POLITECNICO DI TORINO
Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Orchestrazione di servizi SDN/NFV su una infrastruttura geografica

Relatore
prof. Fulvio Risso

Candidato
Francesco Lubrano

Supervisore Aziendale
ing. Matteo D'Ambrosio

Ottobre 2017

Alla mia famiglia

Sommario

Con l'introduzione e lo sviluppo delle Software Defined Network(SDN) e delle Network Function Virtualization(NFV) il mondo delle reti ha subito dei grandi cambiamenti. La necessità di gestire al meglio e in modo automatizzato queste nuove risorse ha portato alla progettazione dei cosiddetti orchestratori, dei software in grado di gestire, attraverso le opportune interfacce, le risorse (domini di rete, domini openstack, etc.) che gli vengono configurate. In questa tesi analizzerò uno di questi orchestratori, il Multidomain Orchestrator (MdO), attraverso la disamina della sua architettura, delle funzionalità e delle potenzialità che è in grado di fornire, e infine presenterò i risultati ottenuti.

Indice

Elenco delle figure

Capitolo 1

Introduzione

Negli ultimi anni il mondo dei servizi di rete è cambiato profondamente con l'introduzione del paradigma *Network Function Virtualization (NFV)*. L'applicazione di questo nuova tecnologia ha consentito di astrarre le network function trasformandole da applicazioni che venivano eseguite in apparecchi dedicati (ad esempio firewall, load balancer, etc.) a set di immagini software che possono essere istanziate in server generici e che vengono chiamate *Virtual Network Function (VNF)*. Grazie alla loro natura puramente software, le VNF possono essere potenzialmente lanciate in qualunque nodo della rete che abbia delle capacità computazionali.

La tecnologia NFV si è affiancata ad un'altra nuova tecnologia che è stata sviluppata a partire dagli anni 2000, le *Software Defined Networking (SDN)*. Le reti SDN sono dei network in cui il piano di controllo viene disaccoppiato dal piano dati, passando dalla tradizionale architettura a controllo distribuito ad una architettura a controllo centralizzato. In altre parole l'intelligenza della rete si concentra in punti chiave e questo porta allo sviluppo di veri e propri sistemi operativi di rete (*Network Operating System*).

Si rende dunque necessario un dispositivo capace di gestire queste nuove risorse, in modo da sfruttare al meglio la flessibilità e la scalabilità introdotta dalle due nuove tecnologie sopracitate. Nel progetto FP7 UNIFY Project viene delineata un'architettura che definisce un insieme di moduli organizzati secondo una precisa gerarchia, che permettono di gestire in modo dinamico le infrastrutture che stanno alla base, che ad esempio possono essere delle reti SDN, oppure dei domini OpenStack. Questa architettura è organizzata in 3 livelli:

- *Service Layer*: è il livello più alto della gerarchia, permette all'utente di definire i servizi attraverso una dashboard oppure utilizzando un *Service graph*, un formalismo usato per descrivere ad alto livello un servizio(ad esempio un insieme di VNF collegate tra loro) .

- *Orchestration Layer*: è il livello intermedio; il Service Layer ricevuto un Service Graph lo elabora e invia all'Orchestration Layer il risultato di questa elaborazione, chiamato *Forwarding Graph (FG)*. L'Orchestration Layer svolge due fasi importanti per lo sviluppo del servizio. Prima manipola il FG in modo da adattarlo alle funzionalità del livello sottostante e poi decide dove istanziare il servizio richiesto.

- *Infrastructure Layer*: è il livello più passo dell'architettura e racchiude tutte le risorse fisiche in cui è possibile istanziare il servizio. Include domini differenti, ognuno dei quali ha il suo controller, ad esempio il dominio Openstack ha il corrispondente Openstack Controller.

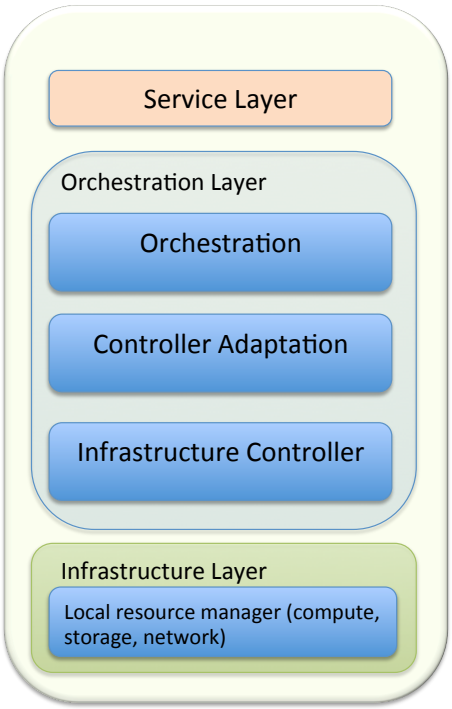

Figura 1.1. Rappresentazione dei livelli di orchestrazione

Questa tesi, svolta in parte presso i Telecom Italia Labs con sede a Torino, ha lo scopo di descrivere i moduli necessari ad implementare l'architettura sopra citata, esaminare nel particolare il *Multi domain Orchestrator (MdO)* e integrarlo nella JOLNet, l'infrastruttura geografica di TIM. Infine la tesi ha come scopo quello di realizzare uno use case in cui un utente accede ad un servizio sulla cloud di JOLNet attraverso una VNF istanziata dinamicamente dall'orchestratore.

La tesi è strutturata come segue:

- **Capitolo 2** spiega quali sono i problemi che si vogliono risolvere in questa tesi.
- **Capitolo 3** da al lettore una descrizione dettagliata dei software usati.
- **Capitolo 4** descrive in modo accurato l'architettura, le funzionalità e i limiti attuali del MdO.
- **Capitolo 5** descrive il software utilizzato per interfacciare il MdO con il Frog4.
- **Capitolo 6** descrive l'applicazione che si occupa di inviare un grafo di servizio al MdO.
- **Capitolo 7** descrive il testbed creato sull'infrastruttura JOLNet e lo use case.

- **Capitolo 8** riporta i risultati ottenuti.
- **Capitolo 9** conclude la tesi e traccia la strada per uno sviluppo futuro.

Capitolo 2

Problemi

Le tecnologie SDN e NFV hanno introdotto nel mondo del networking un nuovo livello di flessibilità. Le tecnologie sopracitate, che si stanno diffondendo, hanno permesso di superare la staticità e la limitatezza dei vecchi sistemi hardware dedicati. Come già detto nell'introduzione, al momento sono stati sviluppati dei software capaci di gestire uno o più domini eterogenei, con l'assunzione che questi domini siano sotto il controllo di uno stesso dominio amministrativo. Per superare questo limite è stato pensato e sviluppato un nuovo tipo di orchestratore multidominio, il MdO la cui caratteristica principale e innovativa è la possibilità di interagire con MdO di altri operatori in modo da rendere possibile la comunicazione tra elementi appartenenti a differenti sistemi amministrativi.

2.1 Interazione del MdO con la JOLNet

Il primo obiettivo di questa tesi è riuscire a far interagire il MdO con la rete JOLNet. La JOLNet è un'infrastruttura geografica disposta lungo la penisola Italiana. È composta da una serie di nodi, ognuno dei quali è formato da uno switch SDN e un compute node OpenStack. La natura distribuita dei compute node rende complicato interfacciare il MdO con il controller Openstack poichè quest'ultimo ha bisogno di ricevere delle informazioni relative al compute node su cui lanciare la Virtual Machine, informazioni che il MdO non è in grado di fornire. Per risolvere questo problema e per separare gerarchicamente meglio i vari livelli di orchestrazione sono stati inseriti tra il MdO e i controller Openstack e SDN, gli orchestratori *FROG4 Global Orchestrator*, il *FROG4 Openstack-Domain Orchestrator* e *FROG4 SDN Domain Orchestrator*. Questi due orchestratori, compatibili con le necessità della JOLNet, in passato sono già stati usati per orchestrare servizi sui domini Openstack e SDN di JOLNet. Posizionando in questo modo il FROG4 Orchestrator il problema di comunicazione tra l MdO con il controller Openstack viene risolto, ma allo stesso tempo se ne crea un altro. Infatti il l MdO e il FROG4 Orchestrator non sono perfettamente compatibili. Nonostante entrambi gli orchestratori siano stati sviluppati seguendo le linee guida del progetto FP7 Unify - https://www.fp7-unify.eu/, i *Network Function - Forwarding Graph NF-FG*, ovvero i formalismi che ogni orchestratore si aspetta di ricevere sulla sua interfaccia, sono differenti.

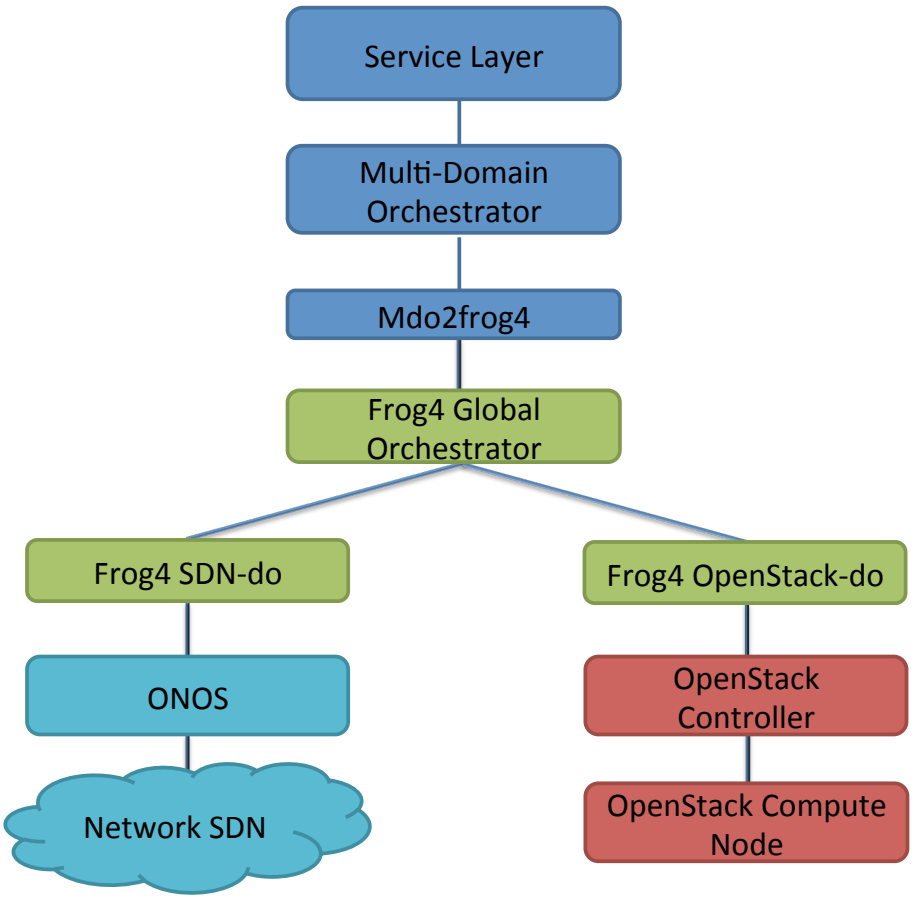

Figura 2.1. Schema architetturale

2.2 Interazione del MdO con il FROG4

Come già detto nel paragrafo precedente, il NF-FG che il MdO invia verso i moduli inferiori non è compatibile con quello che il FROG4 si aspetta di ricevere. È necessario inserire un ulteriore modulo software che si interpone fra i due orchestratori ed esegue la traduzione. Questo modulo è stato chiamato *Mdo2frog4*.

Il Mdo2frog4 si occupa di intercettare i NF-FGs che vengono inviati dal MdO, li traduce in NF-FGs compatibili con la libreria che utilizza l'orchestratore FROG4, fa l'autenticazione e invia i NF-FGs al FROG4. Il Mdo2frog4 verrà trattato in modo completo nel **Capitolo 4**.

Capitolo 3

Background

In questo capitolo vengono introdotte le principali tecnologie e i software che sono stati utilizzate per realizzare l'architettura proposta in questa tesi. Le prime due tecnologie presentate sono le Software Defined Networking e la Network Functions Virtualization che qui verranno trattate in modo approfondito. Gli altri paragrafi saranno invece dedicati ai software utilizzati e non modificati durante la tesi, ovvero gli orchestratori Frog4 Orchestrator e il Frog4 Openstack-Domain Orchestrator, Frog4 SDN-Domain Orchestrator, Onos, OpenStack, DoubleDecker e Docker.

3.1 Software Defined Networking

Il Software Defined Networking (SDN) è un approccio moderno che ha lo scopo di affrontare e colmare l'assenza di dinamicità, scalabilità e capacità di computing richiesti negli ambienti informatici attuali come i data center o le infrastrutture dei Telecom operators. Il concetto fondamentale di questa architettura è il disaccoppiamento tra il *control plane* e *data plane*, ovvero la separazione tra il piano di controllo e il piano dati.

Il control plane è il sistema che si occupa della gestione del traffico, quindi la logica che decide dove devono essere inviati i pacchetti in arrivo. Tale processo porta alla realizzazione di una struttura che immagazzina tutte le decisioni prese per l'inoltro dei pacchetti, come ad esempio una tabella di routing.

Il data plane è il sistema sottostante che si occupa di instradare il traffico verso il *next hop* (nodo successivo) utilizzando le informazioni contenute nella struttura generata dal control plane.

Allo stato attuale questo disaccoppiamento si traduce in dei dispositivi di rete semplici che si occupano solamente del forwarding dei pacchetti e che quindi implementano solamente il data plane. Invece il control plane viene implementato via software, quindi diventa completamente programmabile e viene chiamato controller. Il controller si interfaccia con il dispositivo attraverso una interfaccia di *southbound*, con la quale invia le informazioni necessarie al corretto funzionamento dello switch SDN e inoltre esporta delle informazioni in forma di *API* attraverso una interfaccia di *northbound*, grazie alla quale le applicazioni di livello superiore riescono ad avere una visione astratta dell'infrastruttura sottostante e ne possono manipolare la configurazione, lo stato e il comportamento.

Nell'architettura SDN i flussi sui dispositivi di rete possono essere istanziati seguendo due differenti paradigmi:

- *paradigma reattivo* : all'arrivo di un pacchetto su un dispositivo, se la tabella di quest'ultimo non contiene nessuna regola che combacia col pacchetto in ingresso, questo viene mandato al controller, che lo elabora (a seconda di una qualche logica applicativa) e genera le regole corrette istanziandole in tutta la rete; in questo modo da quel momento in poi tutti i pacchetti di quel tipo verranno trattati tramite quelle regole di flusso, senza avere nuovamente la necessità di recapitare ciascuno di essi al controllore;

- *paradigma proattivo* : le tabelle dei flussi vengono popolate in anticipo, invece che all'arrivo del singolo pacchetto, con delle regole capaci di gestire tutti i match di traffico che si prevede possano attraversare la rete. Questo approccio è simile a quello utilizzato fino ad ora nelle reti tradizionali (si pensi alla tipica tabella di routing). Il risultato è che tutti i pacchetti vengono trasmessi alla massima velocità consentita dall'apparato di rete, ma diventa scomodo introdurre logiche particolarmente complesse e che richiedono una certa dinamicità.

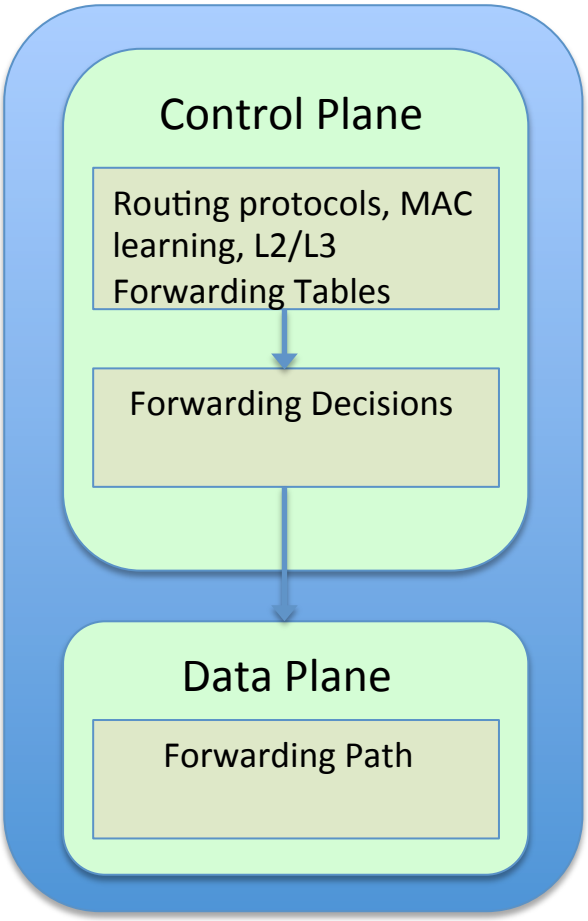

Figura 3.1. Logica SDN

La natura software e programmabile dei controller SDN introduce di fatto un nuovo livello di libertà. Ora è possibile creare delle applicazioni che vengono eseguite sul controller utilizzando delle interfacce e delle librerie standard aperte e non più proprietarie ed è questa la vera rivoluzione introdotta dalla tecnologia SDN.

3.2 Network Functions Virtualization

La tecnologia Network Functions Virtualization (NFV) apporta un grande cambiamento nel mondo del networking, introducendo un cambio di paradigma nel modo in cui sono realizzate le reti di telecomunicazioni, rendendo il software indipendente e libero dall'hardware. Se all'inizio la virtualizzazione riguardava solo i server per estendersi poi allo storage, oggi coinvolge anche le funzioni di rete.

La criticità che viene risolta dall'introduzione della NFV risiede nel fatto che finora il mondo delle reti è costituito da sistemi di controllo e piattaforme di servizio che sono realizzati su dispositivi proprietari, nei quali è molto forte il legame fra l'hardware e il software. In pratica nell'infrastruttura si ha un dispositivo hardware per ogni componente software che si vuole utilizzare. Ciò ha portato nel tempo ad avere una situazione di alta complessità soprattutto nel mondo dei Telecom operators, che hanno nella loro infrastruttura migliaia di questi apparati che hanno costi di gestione molto elevati, basti pensare alla necessità di eseguire gli aggiornamenti che a volte portano anche alla modifica fisica del dispositivo.

I vantaggi derivanti da questo approccio sono innumerevoli, dalla flessibilità e riduzione dei costi fino ad una migliore integrazione tra il mondo delle telecomunicazioni e quello delle tecnologie informatiche, ma il principale beneficio è l'elevata scalabilità introdotta dalla nuova natura virtuale di queste funzioni di rete.La NFV consente in pratica di lanciare delle Virtual Network Functions su dei server generici, allocando un certo quantitativo di risorse computazionali e di storage. Queste risorse possono essere modificate, aumentate o diminuite, in base alle necessità e all'evoluzione della rete (Figura 3.2).

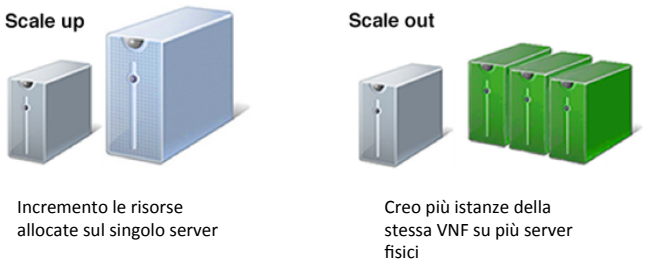

Figura 3.2. Scalabilità verticale e scalabilità orizzontale

Più formalmente, le VNFs, ma in generale ogni tipo di virtual machine, può scalare in due modi:

- *scalabilità verticale* : si modifica la quantità di risorse allocate alla singola macchina virtuale sul server fisico;

- *scalabilità orizzontale* : si attivano più istanze della stessa VNF in modo distribuito su più server fisici.

3.3 OpenFlow

OpenFlow è un protocollo di comunicazione che permette di realizzare la separazione tra il data plane e il control plane. Questo protocollo si colloca al livello più basso di astrazione previsto dall'architettura SDN.

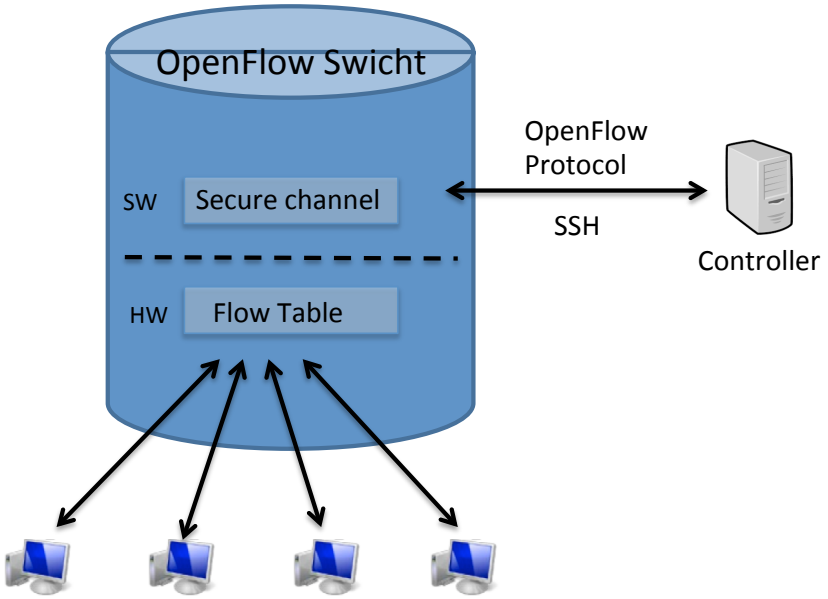

Figura 3.3. Schema di principio di un nodo OpenFlow.

Lo scopo di OpenFlow è quello di presentare all'esterno un modello di nodo generale e unificato, rendendo gli strati più alti dell'architettura di rete SDN indipendenti dall'implementazione del particolare vendor delle tecnologie impiegate nel piano di forwarding. Infatti OpenFlow utilizza un set di istruzioni standard, che sono comuni a qualsiasi dispositivo, controller o switch, sia abilitato o compatibile con OpenFlow. Questo porta a due grandi benefici:

- l'utente che va ad utilizzare questo protocollo non ha dipendenze particolari sul vendor del dispositivo, il che significa che ha la possibilità di avere una rete pienamente compatibile utilizzando dispositivi di vendor differenti;
- i vendor possono aggiungere agevolmente il protocollo OpenFlow ai loro apparati senza dover divulgare il funzionamento interno degli apparati stessi

L'idea alla base di OpenFlow è quella di rendere programmabili in senso generale le tabelle di classificazione ed instradamento dei pacchetti presenti negli apparati di networking (siano essi router o switch). In questo modo il contenuto *(entry)* può essere configurato in modo flessibile dalle applicazioni, attraverso un piano di controllo esterno al dispositivo che comunica con esso mediante un'opportuna interfaccia.

Le tabelle sopracitate, che nel modello di nodo OpenFlow vengono chiamate Flow Tables, specificano le regole di trattamento associate a ciascun flusso *(flow)*. Gli elementi contenuti nella Flow Table specificano come devono essere gestiti i diversi flussi di pacchetti.

Le azioni base che un nodo OpenFlow deve supportare sono le seguenti:

- in caso di un flusso già stabilito inoltrare il pacchetto verso una determinata porta d'uscita;
- in caso di un flusso nuovo, quindi primo pacchetto del flusso, incapsulare il pacchetto e spedirlo al controller attraverso l'interfaccia di comunicazione;
- scartare i pacchetti appartenenti ad un determinato flusso.

Il controller usato in questo lavoro di tesi è *ONOS* che verrà descritto nel paragrafo successivo.

3.4 ONOS

ONOS è l'acronimo di *Open Network Operating System*. ONOS fornisce il control plane di una rete SDN e si pone come sistema operativo di rete, gestendo i componenti della rete, come gli switches e i links, e ha la capacità di eseguire applicazioni che forniscono i *communication services* agli utenti oppure ai network collegati.

Le applicazioni possono essere caricate, attivate e disattivate dinamicamente, tramite API REST o GUI, e senza la necessità di eseguire nuovamente il boot dei dispositivi di rete. Ciascuna applicazione può usufruire di una serie di servizi messi a disposizione da ONOS per poter apprendere lo stato della rete, controllare il flusso di traffico che l'attraversa, consentire la propria configurazione dall'esterno tramite API REST, CLI o GUI, richiedere che particolari pacchetti le vengano recapitati, effettuare operazioni sui pacchetti stessi e così via.

Come si evince dalla Figura 3.4, ONOS può essere lanciato come un sistema distribuito su più server. Ciò gli consente di utilizzare le risorse in termini di CPU e memoria di più macchine fisiche contemporaneamente e questo lo rende resistente ai guasti dei singoli apparati fisici e potenzialmente offre supporto ad aggiornamenti live senza interrompere il traffico sul network. Nella figura inoltre si notano bene le due interfacce del controller descritte nel paragrafo dedicato al SDN. Il protocollo che si interpone tra gli apparati fisici e gli *adapters* nel caso della JOLNet è OpenFlow.

Nel lavoro svolto durante la tesi, il controller ONOS è stato utilizzato per gestire i flussi di traffico e in particolare mi ha permesso di instradare in modo differente i flussi di traffico provenienti dall'utente e destinati ad Internet rispetto ai flussi di traffico provenienti dall'utente e diretti verso la sua VNF.

3.5 Double Decker

DoubleDecker è un sistema di messaggistica gerarchico distribuito basato su *ZeroMQ* che può essere utilizzato per consentire lo scambio di informazioni tra processi situati sulla stessa macchina o su macchine multiple. È gerarchico nel senso che i message broker (che si occupano dello smistamento dei messaggi) sono connessi tra loro in

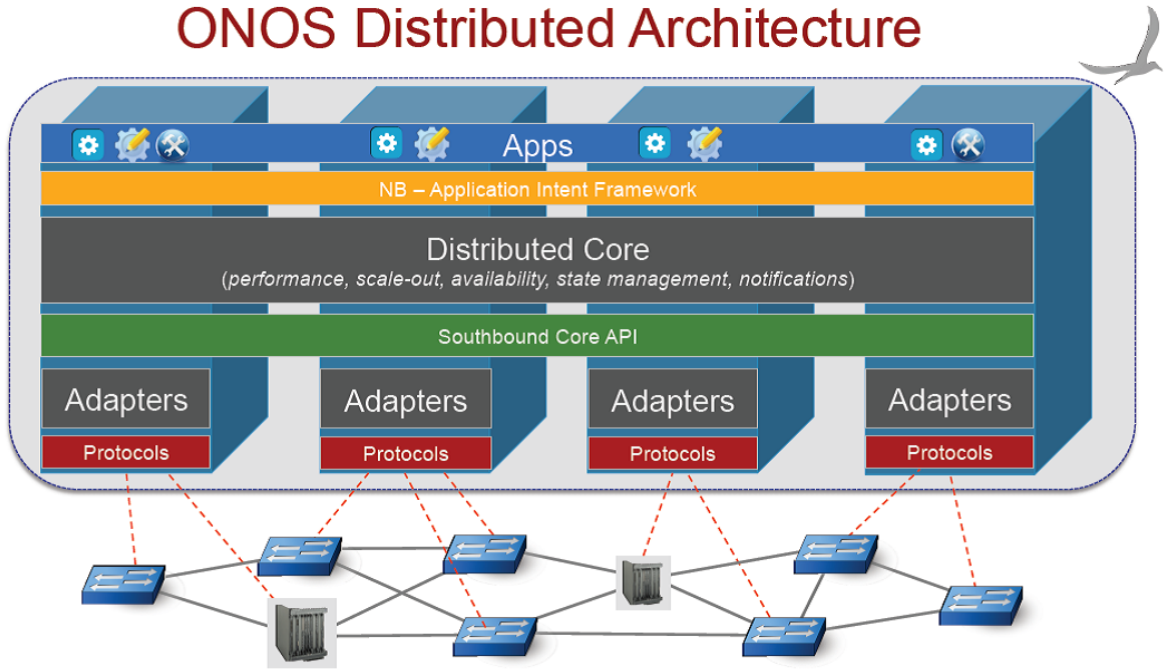

Figura 3.4. Architettura distribuita di ONOS. Immagine tratta da www.onosproject.org

una topologia ad albero ed inviano i messaggi verso l'alto nel caso in cui il client di destinazione non si trovi sotto di essi(Figura 3.5) .

Attualmente DoubleDecker supporta due tipi di messaggistica, *Notifications*, cioè messaggi punto punto da un client ad un altro, e *Publish/Subscribe* su un topic. Il meccanismo di Pub/Sub permette anche di restringere lo scope della sottoscrizione. Questo comporta che un client può decidere di ricevere solo i messaggi di un determinato topic che rientrano in un determinato scope, ad esempio inviati da client collegati direttamente al broker stesso, ad uno specifico o a gruppi di essi.

Sono supportati gruppi multipli di client, detti *tentant*, l'appartenenza ai quali è verificata tramite un'autenticazione a chiave asimmetrica. I messaggi appartenenti a client di un determinato tenant sono cifrati e non possono passare da un tenant all'altro. Esiste un tenant speciale detto *public* che può oltrepassare questo limite, ed essere utilizzato per connettere nodi il cui scopo èquello di fornire/utilizzare un servizio pubblico.

Il paradigma *Publish/Subscribe* si presenta particolarmente scalabile, in quanto evita che un produttore del dato con l'intenzione di comunicare una informazione debba preoccuparsi di chi effettivamente necessita di ricevere quei dati e delle sue coordinate. Dunque è il consumatore del dato che notifica il suo interesse verso un particolare argomento, mentre il produttore deve solo preoccuparsi di etichettarlo con il topic corretto e di pubblicarlo sul bus, lasciando che il broker si occupi di smistarlo a tutti coloro che lo devono ricevere.

In questa tesi DoubleDecker è stato utilizzato come singolo message broker con sistema di messaggistica *Publish/Subscribe*, in modo da gestire le comunicazioni tra i vari moduli software utilizzati e che verranno presentati nei paragrafi seguenti.

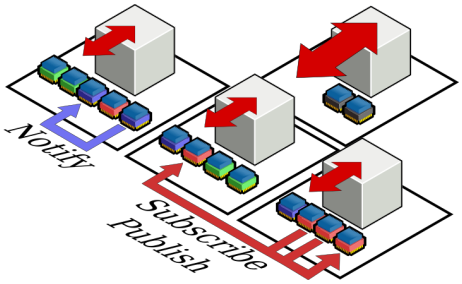

Figura 3.5. Architettura gerarchica di Double Decker e principali operazioni. Immagine tratta da Acreo/DoubleDecker

3.6 Frog4 Orchestrator

Il Frog4 è un sistema di orchestrazione distribuito sviluppato presso il Politecnico di Torino dal gruppo di ricerca *Netgroup*. È una parte dell' *orchestration layer*, descritto nel capitolo 2 di questa tesi, definito nel progetto europeo Unify. Nello specifico il Frog4 implementa i primi due *sub-layer* definiti nell'orchestration layer, ovvero *Orchestration* e *Controller Adaptation*.

In questo paragrafo ne descriverò l'architettura e le funzionalità, attraverso un'analisi top-down.

3.6.1 Global Orchestrator

Il *Global Orchestrator* implementa la trasformazione del forwarding graph, ricevuto dal *service layer* attraverso la sua interfaccia di *northbound*, e lo scheduling tramite un approccio indipendente dalla tecnologia fisica, senza quindi preoccuparsi dei dettagli relativi alla particolare infrastruttura. Questa parte del Frog4 realizza l'orchestration sub-layer.

La trasformazione del forwarding graph consiste nel manipolare il NF-FG in modo da permettere che esso possa essere istanziato nell'infrastruttura o nelle infrastrutture sottostanti. Più precisamente esegue le seguenti due operazioni:

- VNFs expansion : per ogni VNF specificata nel NF-FG, recupera il corrispondente template. Infatti ogni VNF è associata ad un template che la descrive, ad esempio indicando la quantità di risorse necessarie ad istanziarla, le porte disponibili, ecc. Il recupero del template può avvenire in locale (l'orchestratore ha una copia in locale del template della VNF), oppure il template viene ottenuto tramite una richiesta alla *VNF repository* . La tipologia di recupero del template viene specificata nel file di configurazione del Frog4.

- VNFs consolidation : le VNFs che implementano il forwarding L2 e che sono connesse insieme nel NF-FG vengono accorpate in un'unica VNF in modo da limitare le risorse necessarie per implementare le LANs nell'infrastruttura fisica.

Il processo di scheduling, che è responsabile delle decisioni riguardanti la scelta delle infrastrutture nelle quali istanziare le componenti che implementano il servizio richiesto, può basarsi su differenti classi di parametri:

- informazioni riguardanti le VNF, come CPU e memoria richieste;
- politiche di alto livello e requisiti sulla qualità del servizio forniti con il forwarding graph;
- risorse disponibili sull'infrastruttura fisica, come la presenza di un determinato acceleratore hardware in un certo nodo, così come il carico attuale dei nodi stessi.

3.6.2 Domain Orchestrator

Il *Domain Orchestrator* è diverso per ogni tipo di infrastruttura ed implementa una logica ad hoc per ciascuna di esse. Si occupa di tradurre il forwarding graph nell'appropriato set di chiamate verso le API di *northbound* del relativo infrastructure controller, che costituisce la parte inferiore dell'*orchestration layer*. In questa tesi è stato impiegato il Frog4 Openstack domain-orchestrator che si interfaccia con il controller Openstack. Questa parte del Frog4 realizza il *Controller adaptation* sub-layer.

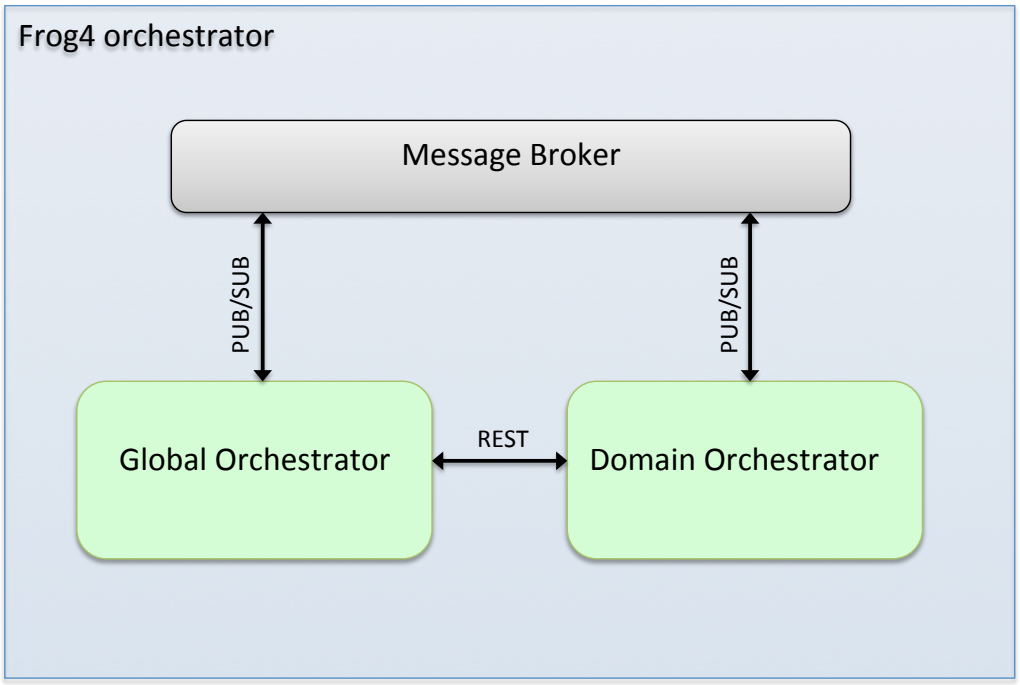

Figura 3.6. Schema che rappresenta il sistema di comunicazione tra il Global e il Domain orchestrator

Come si nota dalla Figura 3.6 le due parti del Frog comunicano attraverso il message broker Double Decker quando devono scambiare messaggi riguardanti la descrizione delle risorse, dei servizi e delle capabilities, mentre le altre informazioni vengono scambiate attraverso l'interfaccia REST.

La comunicazione attraverso il broker è preferibile rispetto a quella che usa le interfacce REST in quanto utilizza un'unica interfaccia sia per leggere che per scrivere e quindi comunicare i dati.

3.7 Tipologie dei grafi delle funzioni di rete

In questo paragrafo vengono chiariti meglio i concetti relativi ai formalismi utilizzati dai vari layer della architettura definita nel progetto Unify per comunicare tra loro.

3.7.1 Service Graph

Il **Service Graph (SG)** è una rappresentazione ad alto livello del servizio che include sia aspetti relativi alle network functions che implementano il servizio (come ad esempio quali sono e come devono essere interconnesse tra loro) sia aspetti legati alla configurazione di queste funzioni. Gli elementi base che costituiscono il SG (mostrati in Figura 3.7) sono descritti di seguito.

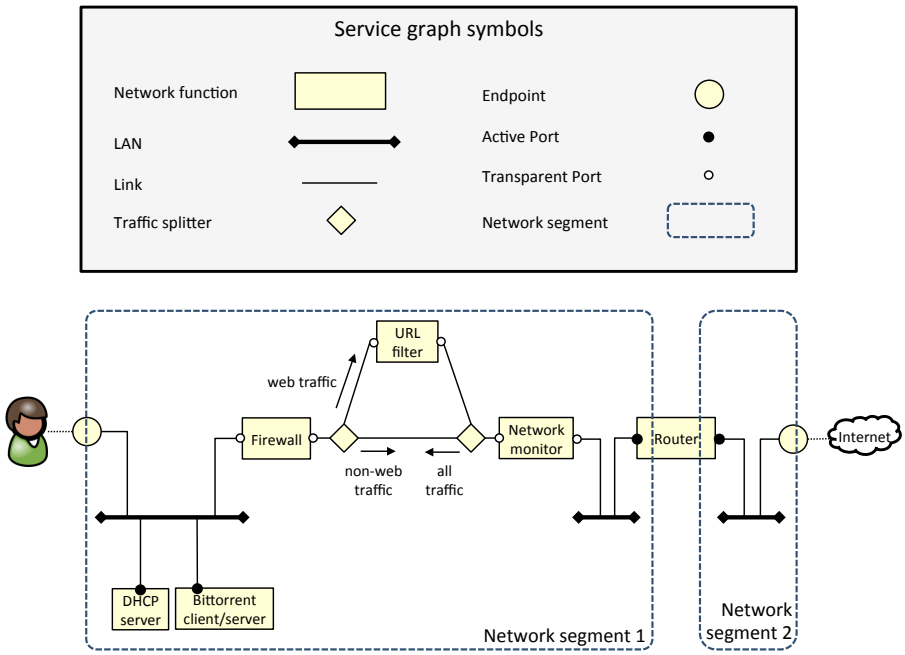

Figura 3.7. Esempio di service graph e insieme dei componenti base

Quello detto **network function (NF)** è un blocco funzionale che verrà in seguito tradotto in una (o più) VNF.

Le **porte** costituiscono le interfacce delle NF, e possono essere **attive** (se richiedono un indirizzo di rete) o **trasparenti**.

La **LAN** rappresenta il mezzo logico di comunicazione di tipo broadcast. La disponibilità di questa primitiva facilita la creazione di servizi complessi che includono non solo NF trasparenti, ma anche tradizionali servizi host-based.

Il **link** (point-to-point) definisce il collegamento logico tra differenti componenti e può essere utilizzato per connettere tra loro porte, endpoint, LAN, eccetera.

Il **traffic splitter/merger** è un blocco funzionale che permette di suddividere il traffico in una serie di flussi in base a delle regole specifiche, o di unificare quello proveniente da link differenti.

Infine, l'**endpoint** rappresenta un punto esterno a cui è agganciato il SG. Può trattarsi sia di una entità logica, sia di una porta specifica (ad esempio una NIC

fisica/virtuale, un tunnel di rete, etc.), purchè sia attiva su un dato nodo dell'infrastruttura fisica. Un endpoint può essere utilizzato per collegare il SG ad internet, ad un dispositivo dell'utente finale, ma anche all'endpoint di un altro service graph qualora occorresse collegarne più di uno in cascata per creare un servizio più complesso.

Come accennato in precedenza, il SG include anche aspetti relativi alla configurazione delle network functions, i quali rappresentano importanti parametri dello strato di servizio che vanno definiti assieme alla topologia e che possono essere utilizzati dal piano di controllo dell'infrastruttura di rete per configurare in maniera opportuna il servizio. Questi parametri riguardano aspetti di rete come gli indirizzi da assegnare alle porte delle NF, così come configurazioni specifiche delle funzioni di rete stesse.

3.7.2 Forwarding Graph

Il formalismo definito dal service graph è di alto livello e non adatto alla fase di allocazione del servizio sull'infrastruttura fisica, in quanto non include tutti i dettagli di cui questo ha bisogno per operare [**frog**]. Dunque si ha la necessità di trasformarlo in una rappresentazione più orientata alle risorse, cioè nel **Forwarding Graph (FG)**, attraverso un processo detto di espansione le cui fasi sono riportate in Figura 3.8.

Nel primo step, detto di **espansione di controllo e di gestione**, viene aggiunta una LAN di configurazione a cui vengono collegate tutte le NF che presentano una vNIC dedicata identificata come *interfaccia di controllo*; questa rete ha il compito di garantire la possibilità di monitorare e configurare opportunamente le NF e può essere un dettaglio non importante per l'utente che richiede il servizio (e quindi non presente nell'astrazione del SG), ma richiesto affinchè il servizio stesso sia in grado di operare.

Nel secondo step avviene la **LAN expansion**, la quale traduce l'elemento LAN astratto definito nel SG in un appropriato set di VNF che emulano il mezzo di comunicazione broadcast. I primi due step sono visibili nel passaggio da Figura 3.8(a) a Figura 3.8(b).

La fase di **service enrichment** analizza ed arricchisce il grafo con quelle funzioni che non sono state inserite nel SG dall'utente, ma sono comunque richieste per la corretta implementazione del servizio. In Figura 3.8(c) l'analisi del grafo determina che il segmento di rete che connette il grafo utente ad Internet non presenta alcun servizio di NAT e routing, e che non è presente nessun server DHCP all'interno della rete locale; il set appropriato di NF viene automaticamente aggiunto.

La **NFs expansion** può sostituire una network function con altre equivalenti, o con un intero sottografo equivalente, in modo da implementare il servizio richiesto. In Figura 3.8(c) il firewall è stato sostituito con un sottografo i cui endpoint coincidono con le porte della vecchia NF.

La fase di **NFs consolitation** analizza il FG cercando eventuali funzioni ridondanti ed ottimizzando il grafo dove possibile. Per esempio, la Figura 3.8(d) mostra come due switch software connessi insieme siano stati rimpiazzati da un unica NF, riducendo le risorse richieste per implementare il grafo sull'infrastruttura fisica.

La **endpoint translation** converte gli endpoint del grafo nelle appropriate porte fisiche del nodo in cui il servizio deve essere istanziato, nei tunnel di rete (ad esempio

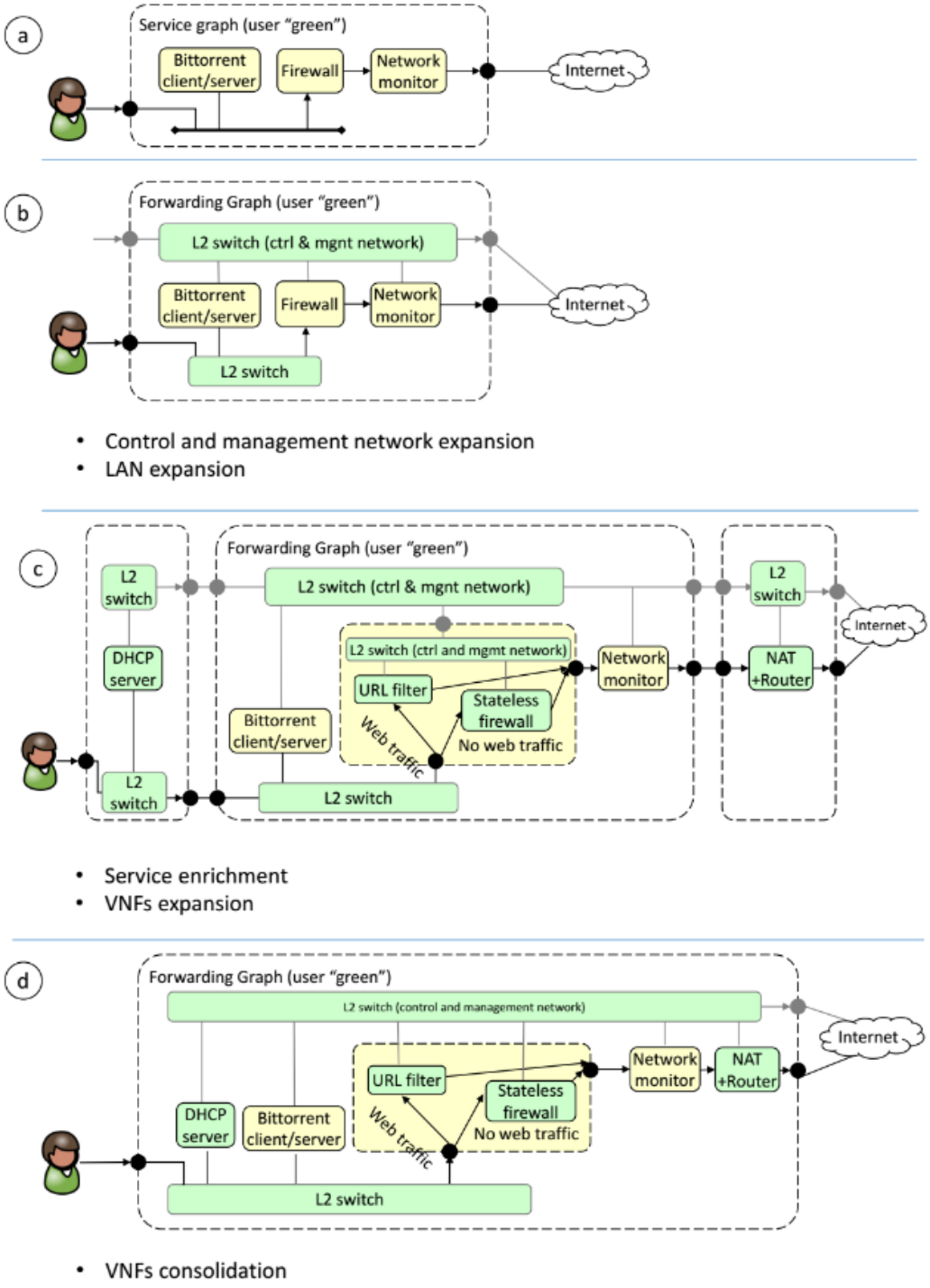

Figura 3.8. Dal service graph al forwarding graph: il *processo di espansione*.

GRE) che connettono il grafo ad un altro istanziato in un dominio esterno, e così via.

Lo step finale è quello della **flow-rules definition**, in cui le connessioni tra NF, le regole definite negli splitter/merger del SG e le regole di ingresso associate ad ogni endpoint vengono tradotte in una serie di coppie match-azione, seguendo un paradigma SDN-based.

3.7.3 Infrastructure Graph

La rappresentazione finale del servizio da istanziare è detta **Infrastructure Graph (IG)**; esso è semanticamente equivalente al FG, ma sintatticamente diverso. L'IG è ottenuto tramite il processo di **riconciliazione**, il quale mappa il FG sulle risorse disponibili nel dominio, e consiste nella sequenza di comandi che devono essere eseguiti sull'infrastruttura fisica in modo da istanziare, configurare e connettere correttamente tra loro tutte le VNF richieste.

3.8 OpenStack

OpenStack è un progetto IaaS cloud computing. Consiste in una serie di progetti legati fra loro e il suo compito è amministrare e controllare le risorse di elaborazione, di storage e di rete in un data center.

OpenStack è un sistema operativo cloud, il cui scopo principale è quello di esportare un'astrazione dell'hardware fisico che controlla, ma anzichè comandare direttamente l'hardware attraverso i drivers come avviene ne caso di un sistema operativo tradizionale, ha la capacità di interfacciarsi con un numero arbitrario di agents che risiedono sopra l'hypervisor; in altre parole si comporta come un manager di hypervisor. I sistemi operativi cloud semplificano molto la gestione dei data center, nei quali si fa largo uso delle virtual machine, fornendo un'astrazione dei server fisici che in questo modo possono essere controllati e gestiti da un singolo punto.

Il progetto OpenStack ha lo scopo di fornire una interfaccia centralizzata che da agli amministratori del data center una visione completa delle risorse disponibili, di quelle utilizzate e dello stato, e permette agli utenti di istanziare macchine virtuali e collegarle fra loro senza che si debbano preoccupare della configurazione e delle risorse disponibili degli apparati fisici sottostanti che le ospiteranno.

Tutta questa automazione fornita da OpenStack ha dei costi in termini di risorse fisiche, che sono facilmente ragionevoli se si tratta di utilizzo in un data center. Nel caso in cui si abbiano pochi server fisici, l'installazione di questo sistema è sconsigliata.

3.8.1 Architettura

OpenStack è formato da tre entità logiche separate che possono essere eseguite in una o più macchine fisiche e a con cui si interagisce attraverso la Dashboard oppure le APIs. Queste tre entità sono generalmente indicate come segue:

- **Controller node**
- **Compute node**

- **Network node**

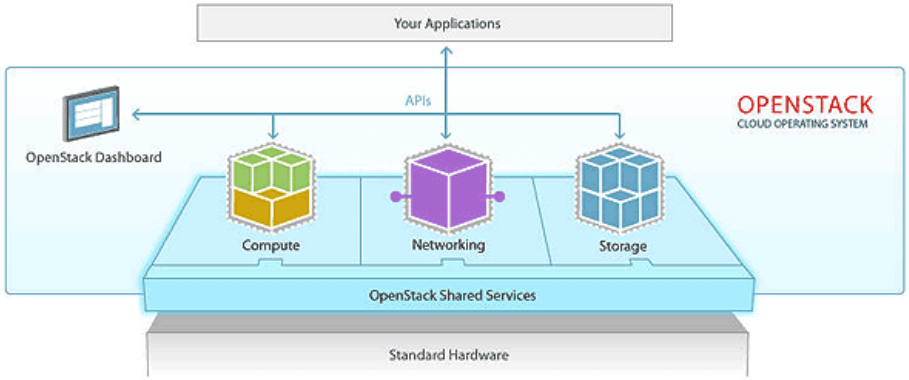

Figura 3.9. Architettura generale OpenStack. Immagine tratta da http://www.openstack.org

Controller node

Il **Controller node** è costituito da dei componenti di core, da servizi di supporto e da una serie di features opzionali. Come si nota in Figura 3.10, i componenti che costituiscono il core del Controller node sono tutti quei moduli che forniscono le funzionalità di base per il corretto funzionamento di OpenStack e che ne costituiscono il fulcro delle capacità di management.

I servizi di supporto non sono da considerarsi propriamente componenti di OpenStack ma sono necessari per il suo funzionamento. Questi servizi sono il sistema di gestione del database, *MySQL* o *MariaDB*, che permette la persistenza dei dati e il message broker, *RabbitMQ* che consente ai moduli di intercomunicare.

Opzionalmente nel controller possono essere installate dei componenti che offrono funzionalità aggiuntive, come ad esempio *Ceilometer*, un servizio di telemetria di OpenStack che raccoglie informazioni sull'uso delle risorse per utente, le quali vengono utilizzate durante il processo di fatturazione.

Il controller node deve poter comunicare direttamente con tutti gli altri nodi in modo da poter inviare direttamente i comandi e riceve le informazioni aggiornate.

Compute node

Il **Compute node** è il nodo che ospita le virtual machines. Questo nodo viene ospitato da un singolo server fisico ma, come nel caso della JOLNet, è possibile avere più compute node controllati da uno stesso controller. I moduli presenti in questo nodo sono pochi e ciò favorisce il risparmio di risorse a favore delle virtual machine che verranno ospitate. Gli unici software che sono richiesti in questo nodo sono un agent che comunica con l'hypervisor locale e un agent, opzionale, che si occupa di accettare e gestire comandi particolari per fornire traffic isolation tra gli user. Per poter comunicare con il controller node, il compute node ha un'interfaccia

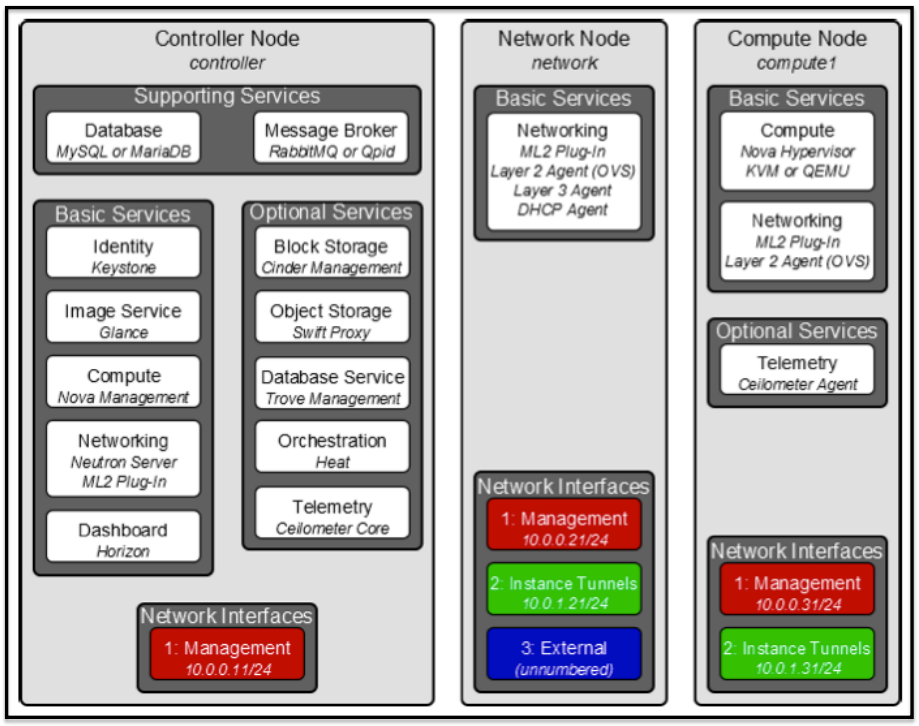

Figura 3.10. Organizzazione interna dei moduli. Immagine tratta da http://www.openstack.org

network dedicata, indicata in rosso nella Figura 3.10. Il traffico prodotto dagli utenti delle virtual machine invece viene fatto passare da un'altra interfaccia, indicata in verde nella Figura 3.10.

Network node

il **Network node** è la frontiera tra il dominio OpenStack e il mondo esterno; in particolare, è l'unico nodo che permette ad una virtual machine di poter comunicare con il mondo esterno, gestendo il traffico in ingresso e in uscita.

3.9 Docker

Docker è una piattaforma per lo sviluppo, la distribuzione e l'esecuzione di applicazioni; permette di separare le applicazioni dall'infrastruttura consentendone una più semplice gestione. Sfruttando Docker per la distribuzione, il trasporto ed il test del codice in modo rapido, è possibile ridurre in modo significativo il ritardo tra la scrittura del codice stesso e la messa in produzione.

Le applicazioni girano in un ambiente isolato chiamato *container*: l'isolamento e la sicurezza permettono l'esecuzione di diversi container simultaneamente nello stesso host, in numero maggiore rispetto alle *virtual machines* grazie alla natura leggera dei container (che girano senza il carico extra di un hypervisor).

Docker mette a disposizione dei *tool* ed una piattaforma per la gestione del ciclo di vita dei container permettendo l'incapsulamento delle applicazioni, la distribuzione, il trasporto e l'installazione in ambiente di produzione (sia in data center che nel cloud).

La parte principale del sistema, che prende il nome di "Docker Engine" (in figura 3.11), è un'applicazione client/server composta da:

- un server di tipo *long time execution* chiamato processo demone: si occupa di creare e gestire i container, le immagini la rete e il volume dati;
- un'interfaccia client a linea di comando (CLI): l'utente usa la CLI per interagire con il client docker che ha come compito quello di ricevere i comandi, interpretarli e gestire lo scambio dei messaggi da e verso il demone. Un client può comunicare con diversi demoni;
- una REST API: definisce l'interfaccia che il client deve utilizzare per poter parlare con ed istruire il demone.

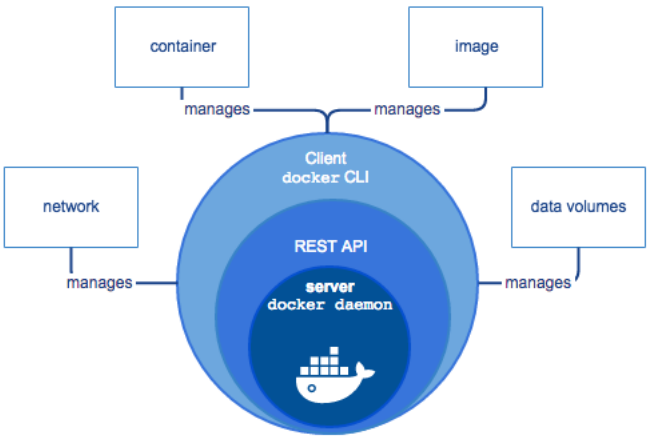

Figura 3.11. Docker Engine. Immagine tratta da www.docker.com

Un container è un'istanza in esecuzione di un'immagine, rappresentata da un template *read-only* contenente le istruzioni per la creazione dei container. Ad esempio un'immagine potrebbe contenere un sistema operativo Ubuntu in cui gira un *server web* Apache ed una *web application* scritta dall'utente.

Le immagini vengono memorizzate in un registro, che può essere pubblico o privato e può trovarsi sulla stessa macchina che ospita il demone o il client o su un server separato.

3.9.1 Docker Compose

Docker Compose è un tool messo a disposizione da Docker che permette di definire ed eseguire applicazioni Docker multi-container. Con Compose è possibile creare un file di configurazione in formato **.yml** nel quale vengono definiti i servizi da attivare, le caratteristiche ad essi correlate e le modalità di esecuzione degli stessi. Ad esempio

è possibile assegnare un nome al container che conterrà il servizio, definire delle variabili d'ambiente etc. È anche possibile definire le dipendenze tra un servizio e l'altro, stabilendo un ordine di avvio.

Una volta creato il file sarà possibile mandare in esecuzione i vari servizi ospitati nei container utilizzando un unico comando.

Questo facilita le fasi di testing ed esecuzione di programmi complessi che vengono eseguiti in più container, come nel caso del MdO che essendo un software costituito da servizi che vengono eseguiti su più container, utilizza questo tool per rendere più semplice e veloce la sua esecuzione. Il MdO verrà trattato in modo più approfondito nel **capitolo 4**.

Capitolo 4

5GEX Multi Domain Orchestrator

In questo capitolo verrà descritto nel dettaglio il Multi Domain Orchestrator (MdO). Il MdO è un software di orchestrazione progettato con lo scopo di essere l'elemento chiave per la gestione dell'orchestrazione dei servizi e delle risorse in ambienti multi-dominio. L'obiettivo del MdO è in prima istanza raccogliere le informazioni sulle risorse e sui servizi che può gestire nel suo dominio amministrativo, attraverso le sue interfacce di southbound, ed esportarle ad altri MdO attraverso una rappresentazione chiamata *BigSwitch with BigSoftware (BiS-BiS)*. La versione del MdO che verrà descritta in questo capitolo è la prima release chiamata P1.

Le operazioni principali supportate da questa versione sono:

- orchestrazione di risorse su un singolo dominio;
- discovery di altri MdO;
- creazione di *slice* per il supporto all'orchestrazione di risorse in ambiente multi dominio amministrativo.

4.1 Rappresentazione Bis-Bis

Il concetto di rappresentazione *BiS-BiS* è fondamentale per rendere possibile la condivisione delle risorse di un dominio verso un altro.

Con questa rappresentazione le risorse di un dominio vengono astratte e ne viene creata una visione ad alto livello. In questo modo altri domini hanno la capacità di capire quali sono le risorse che un certo MdO può orchestrare e potranno all'occorrenza allocarle senza conoscere i dettagli di come esattamente quelle risorse sono organizzate e gestite. Di fatto la rappresentazione BiS-BiS descrive un dominio come un big-switch con molteplici interfacce e con all'interno la capacità di eseguire delle *Network Function*. Un dominio nella rappresentazione Bis-Bis viene chiamato *Bis-Bis node*.

Il Bis-Bis node è una rappresentazione capace di unire insieme le risorse di computing con quelle di networking. Come si può vedere dalla figura 4.1 ogni nodo può possedere delle risorse computazionali, quindi la capacità di mandare in esecuzione delle Network Function, e delle risorse di networking, rappresentate dalle interfacce SAP collegate al nodo.

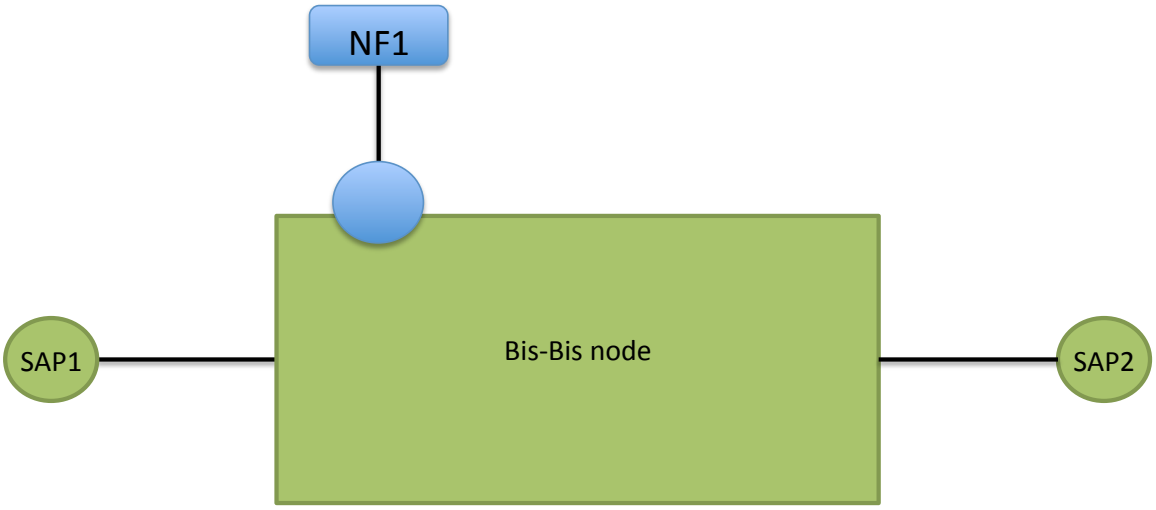

Figura 4.1. Bis-Bis node

4.2 Software già sviluppati ed integrati nel 5GEx MdO

La filosofia alla base della costruzione del MdO è stata quella di riutilizzare ed adattare alcuni moduli sviluppati in progetti precedenti, come il *Netphony Topology*, alcune parti del *T-NOVA marketplace* e lo *Unify Resource Orchestrator* creati ed utilizzati in altri progetti e a disposizione dei partners di 5GEx. (aggiungere reference ai tre moduli citati)

4.2.1 Netphony Topology

La Netphony Topology è una delle librerie principali che compongono la suite di orchestrazione di rete Netphony. Questa libreria consta di alcuni plugin, il più importante dei quali è una implementazione del BGP-LS speaker, che usa il protocollo BGP-LS implementato in un'altra libreria della suite, la Netphony Network Protocols. Quest'ultima libreria supporta la maggior parte dei messaggi BGP-LS.

Questo software viene utilizzato nel MdO durante il processo di discovery degli altri MdO e per il successivo scambio di informazioni di base sulla disponibilità delle risorse controllate dagli orchestratori.

4.2.2 T-NOVA Marketplace

Nel lavoro svolto in questa tesi non si è fatto uso del modulo T-NOVA marketplace, che realizza una sorta di livello di servizio posto nel punto più prossimo all'utente e dotato di un'interfaccia grafica che semplifica la costruzione del grafo che poi viene elaborato dal MdO.

I moduli del T-NOVA marketplace implementati nel MdO sono il VNFStore, il Catalogue Mgmt e la GUI. Questi tre moduli collaborano per realizzare un'interfaccia grafica che permette all'utente di selezionare delle VNF tra quelle presenti nel

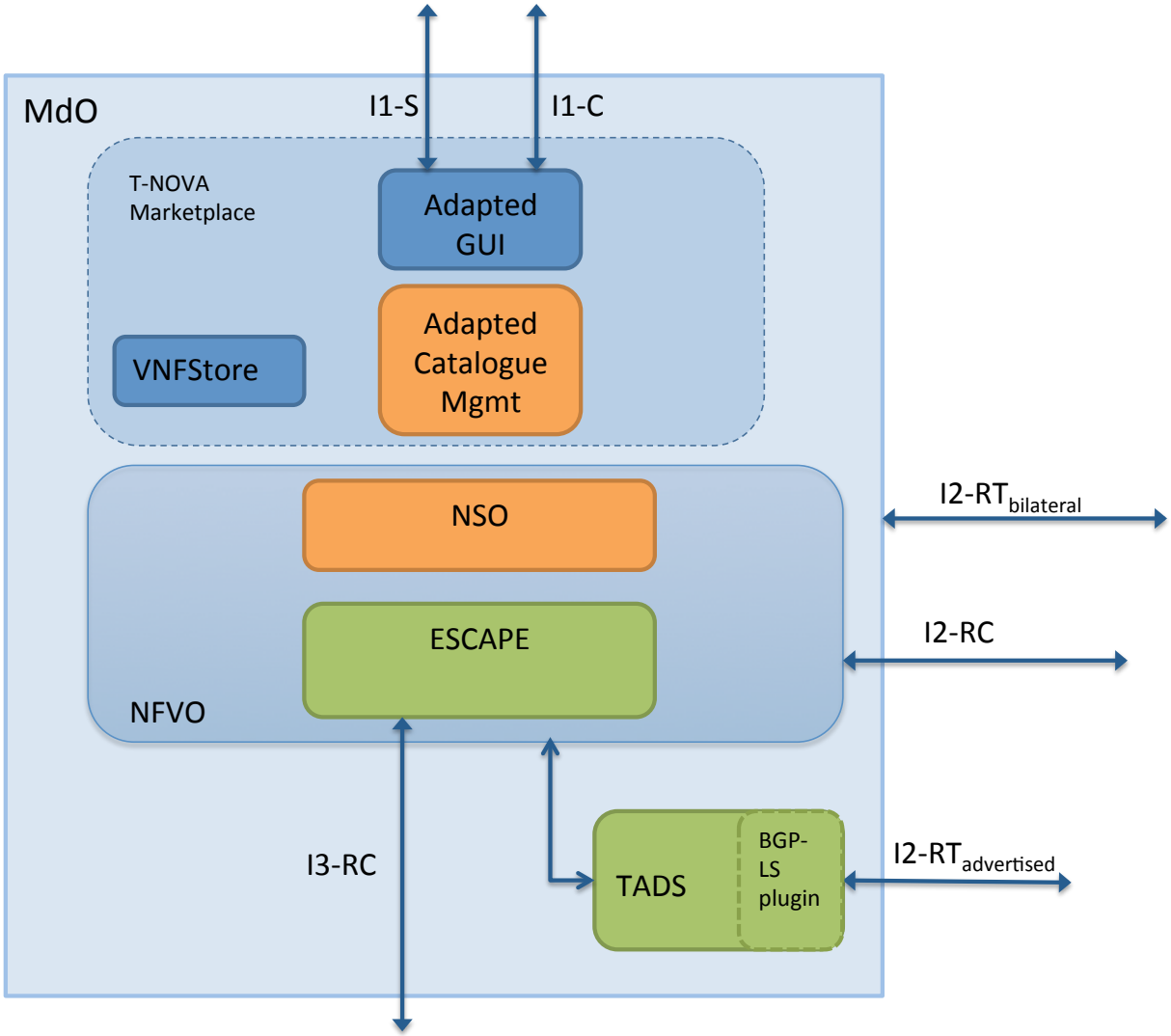

Figura 4.2. Schema architetturale con i moduli riutilizzati messi in evidenza

VNFStore, di definire uno o più *SLA (Service Level Agreement)* e infine di stabilire i collegamenti tra le VNF scelte in modo che queste possano comunicare tra loro o con l'esterno. Questa procedura porta alla definizione di un primo grafo di servizio basato su un modello chiamato *NSD (Network Service Descriptor)*. Una volta creato il NSD, questo viene mandato al modulo sottostante, il *NFVO (NFV Orchestrator)*.

4.2.3 Unify Resource Orchestrator

All'interno del NFVO si trova la componente principale del MdO, il *Resource Orchestrator*, ovvero il modulo che si occupa del processo di orchestrazione delle risorse. Anche in questo caso è stato riutilizzato un software già esistente, *ESCAPE (Extensible Service Chain Prototyping Environment)*. ESCAPE è definito un **global orchestrator**, ovvero un orchestratore che ha la capacità di controllare più domini tecnologici differenti, che però fanno parte dello stesso dominio amministrativo. Per questo ESCAPE si può definire di per sè un orchestratore multidominio.

ESCAPE include diverse interfacce ed adapters per diversi tipi di tecnologie. La sua interfaccia più importante, la quale è stata anche utilizzata per collegare il MdO con il FROG4, è l'interfaccia *Unify Virtualizer*, che permette il controllo della rete e delle risorse IT. Contestualmente a questa interfaccia è stata sviluppata la *Virtualizer library*, una libreria che definisce un modello di grafo in accordo con le linee guida del progetto FP7 Unify.

4.3 Architettura del MdO

Il MdO è un software modulare composto da più parti, che vengono eseguite in container separati e che comunicano tra di loro attraverso delle interfacce interne e con l'esterno attraverso delle interfacce dedicate.

Come già detto il modulo fondamentale del MdO è il NFVO, ovvero il modulo che si occupa del processo di orchestrazione. La descrizione dell'architettura del MdO partirà da questo modulo centrale e successivamente verranno descritti i moduli per il discovery, la comunicazione e lo scambio di informazioni con gli altri MdO mettendo in evidenza i protocolli e le interfacce che vengono utilizzate per questo tipo di comunicazioni. Saranno infine descritte le rimanenti interfacce che il MdO mette a disposizione.

4.3.1 NFV Orchestrator

Il NFVO è il servizio che si occupa dell'orchestrazione delle risorse. È composto dal *Network Service Orchestrator (NSO)* e dal Resource Orchestrator e costituisce la parte più 'intelligente' del MdO, quella che si occupa di gestire le risorse sottostanti e di comunicare con i vari orchestratori di dominio, ottenendo una visione completa delle risorse che dovrà orchestrare.

NSO

Il NSO si occupa di ricevere un NSD attraverso l'interfaccia *I1-S(Interface1-Service)*, ne estrapola il *VNFFG (VNF Forwarding Graph)*, la parte del descriptor che contiene le VNF richieste dall'utente, le cerca nel VNFStore e attraverso un modulo di traduzione chiamato *SV2R (Service to Resource translator)*, traduce il VNFFG in un NFFG coerente con il modello richiesto da ESCAPE.

Il modulo di traduzione SV2R è stato completamente sviluppato in questo progetto in modo da evitare di dover modificare l'interfaccia di northbound di ESCAPE. Evitando questa modifica è possibile continuare ad usare ESCAPE in modo indipendente dai moduli soprastanti permettendo così di poter comunicare direttamente con l'orchestratore.

RO

Il Resource Orchestrator ha il compito di orchestrare le risorse. Ciò significa che una volta ricevuto il grafo sulla sua interfaccia di northbound, ESCAPE cerca di splittare il grafo in uno o più sottografi, ognuno dei quali verrà inviato ai Domain Orchestrator sottostanti, oppure ad altri MdO. In particolare il RO ha il compito

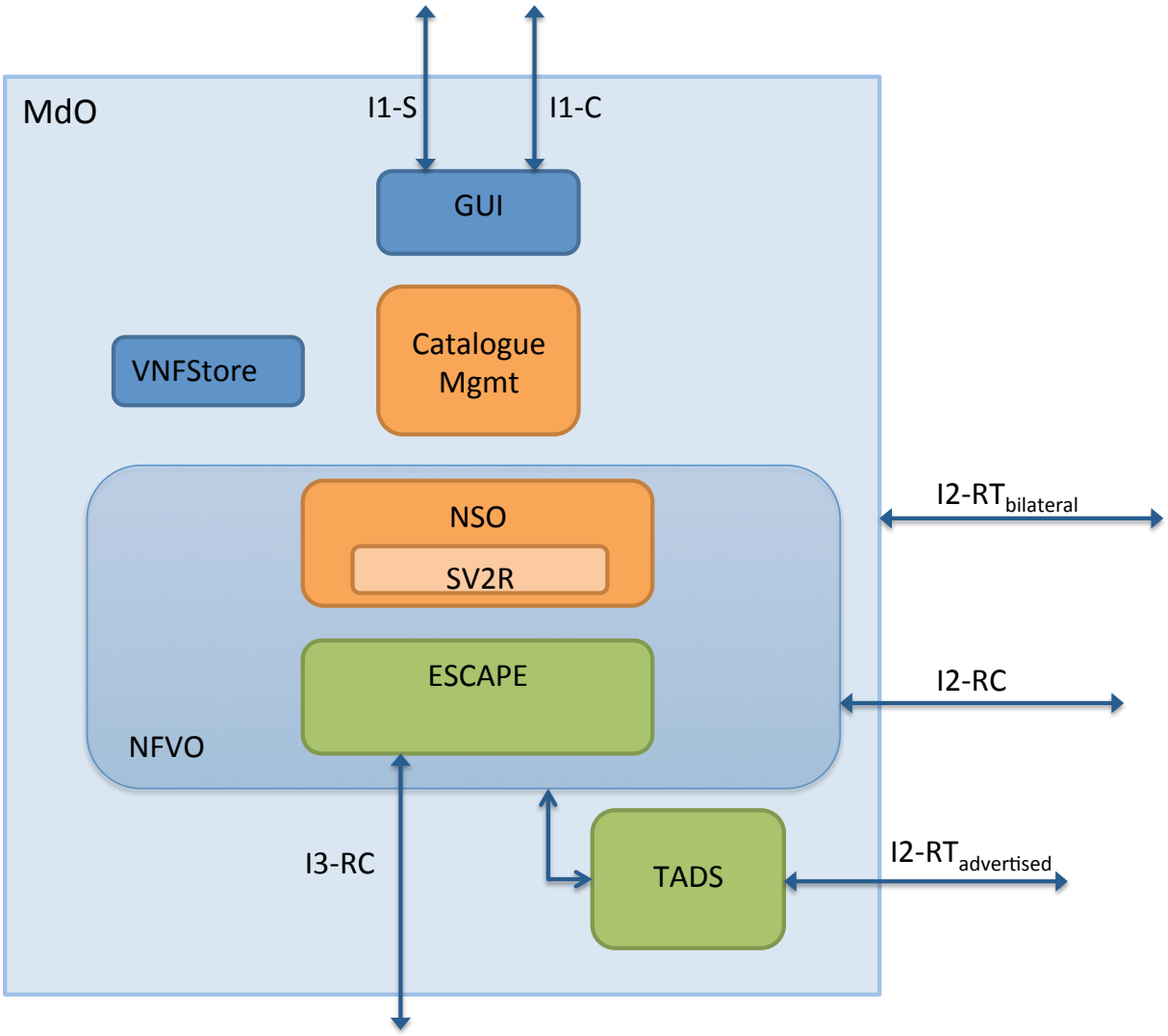

Figura 4.3. Schema architetturale

di ricevere le informazioni ottenute dai Domain Orchestrator che controlla e dagli altri MdO che comunicano con lui, e decidere dove allocare le risorse necessarie a soddisfare la richiesta giunta con il grafo attraverso l'interfaccia di northbound.

Dunque è il RO che decide come partizionare il grafo basandosi sulle informazioni che gli vengono inviate dai vari DO e dagli altri MdO, come risorse disponibili, interfacce, endpoint, etc.

Il MdO comunica con i Domain Orchestrator attraverso l'interfaccia I3-RC e con gli altri MdO attraverso l'interfaccia I2-RT$_{\text{bilateral}}$. Ovviamente la comunicazione con gli altri MdO è possibile solo dopo il processo di discovery.

In particolare ESCAPE, che è di fatto l'implementazione del RO, ha lo scopo di ricevere attraverso la sua API REST un grafo con la descrizione delle risorse da allocare e far partire il processo di orchestrazione che attraverso un algoritmo dedicato di resource mapping, crea dei sottografi partendo da quello ricevuto e li invia sia agli orchestratori di dominio sia agli altri MdO, in base alla disponibilità delle risorse nei vari domini.

Una volta ricevuto il sottografo i vari orchestratori inizieranno ad allocare le risorse richieste.

4.3.2 Topology Abstraction and Discovery Subsystem

Il *TADS(Topology Abstraction and Discovery Subsystem)* è il sottosistema che si occupa del processo di discovery degli MdO degli altri provider e scambia con loro informazioni riguardo al tipo e alla quantità di risorse disponibili.

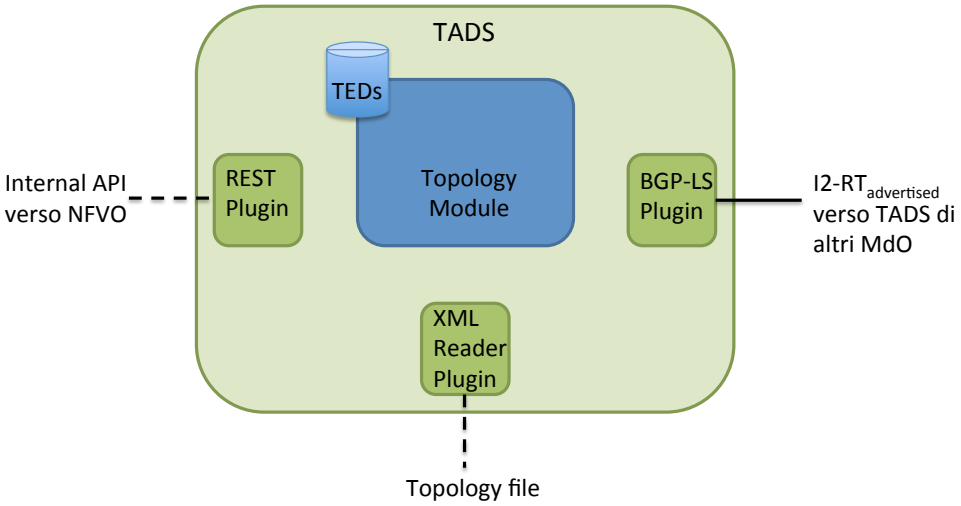

Figura 4.4. Architettura del TADS

Le informazioni ricevute dagli altri MdO vengono salvate in un database interno (TEDs) e vengono inviate al modulo di orchestrazione NFVO, utilizzando il plugin REST, che le userà durante la fase di orchestrazione delle risorse, per decidere se e come dividere il grafo in sottografi. Questo scambio di informazioni tra orchestratori avviene durante la fase di discovery. Un orchestratore invia tre tipi di informazioni:

- l'URL che rappresenta l'entry point attraverso il quale l'orchestratore si rende raggiungibile;
- una rappresentazione astratta ed aggiornata delle sue risorse di rete;
- una rappresentazione astratta ed aggiornata delle sue risorse IT (e.g. CPUs, storage, memory).

Il TADS conserva nel database anche la rappresentazione astratta dello stato delle risorse locali, ovvero quelle che il MdO gestisce direttamente. Come si evince dalla figura 4.4 il TADS è composto da alcuni sottomoduli, ognuno dei quali implementa una determinata caratteristica.

TEDs

I TEDs (Traffic Engineering Databases) sono dei database che contengono la descrizione dello stato delle risorse dei vari MdO. Viene creato un database per ogni MdO scoperto nel processo di discovery.

Topology Module

Il Topology Module è il sottomodulo che si occupa di fornire le informazioni conservare nei TEDs al NFVO attraverso l'interfaccia REST interna e utilizzando il REST Plugin.

REST Plugin

Il REST Plugin si occupa di raccogliere i dati presenti in ogni TED, di convertirli in un opportuno data model e renderli disponibili al NFVO attivando un web server che utilizza il protocollo RESTCONF. Il data model in cui vengono convertite le informazioni è il modello BiS-BiS, descritto al paragrafo 4.1.

Il REST Plugin in questa versione del MdO è read only, dunque il NFVO è in grado solo di leggere le informazioni che gli vengono esposte dal REST Plugin e non può in alcun modo modificarle direttamente, ovvero non è in grado di modificare la disponibilità delle risorse di rete o modificare il totale delle risorse IT disponibili.

XML Reader Plugin

Come detto precedentemente nel database interno sono anche salvate le informazioni riguardanti le risorse locali (e.g. l'entry point del MdO e le risorse di rete e IT presenti nel dominio locale). Queste informazioni sono importate all'interno di un TED dal XML Reader Plugin, che legge un file XML.

Questo file XML è un file di configurazione del MdO chiamato **network.xml** e prima di fare il deploy del MdO è necessario modificare in modo appropriato questo file, inserendo le informazioni concernenti l'identificativo e le risorse complessive del dominio.

BGP-LS Plugin

Il BGP-LS Plugin è il plugin che si occupa del processo di discovery degli MdO. I BGP-LS Speakers dei vari MdO sono capaci di comunicare tra di loro attraverso l'interfaccia I2-$RT_{advertised}$ utilizzando una versione estesa del protocollo BGP-LS. Grazie a questa estensione gli MdO possono scambiarsi sia gli advertisement sulle traffic engineering metrics sia le informazioni sulle risorse che controllano.

Più formalmente il BGP-LS Plugin si occupa di esportare il modello astratto delle risorse locali, e di importare le rappresentazioni astratte delle risorse degli altri MdO connessi, utilizzando il data model di Unify.

4.3.3 Interfacce

In questa sezione verranno elencate e descritte le interfacce più importanti del MdO, in particolare le interfacce I2-RT, I2-RC, I3-RC, che permettono di realizzare i processi di discovery ed orchestrazione.

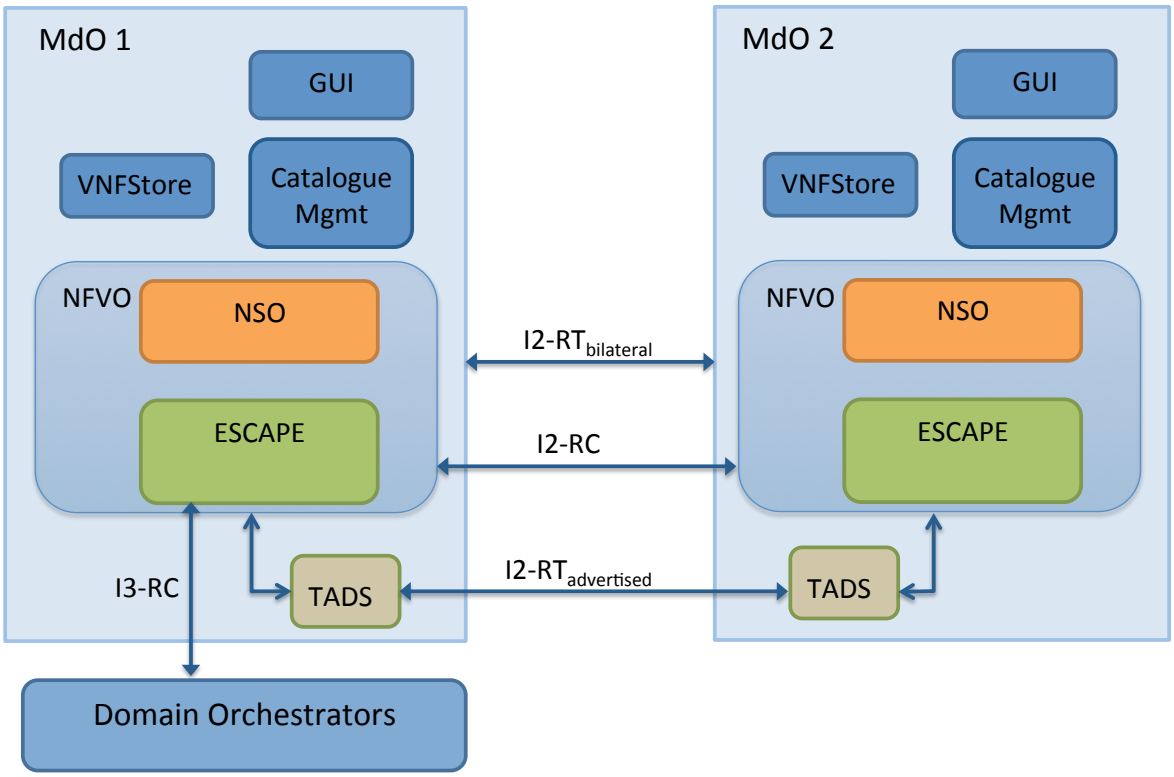

Figura 4.5. Comunicazione tra due MdO e tra MdO e DOs

Interfaccia I2-RT$_{\text{advertised}}$

Questa interfaccia rende possibile il processo di discovery degli altri MdO ed utilizza il protocollo BGP-LS per inviare e ricevere informazioni. In particolare le informazioni sui costi dei collegamenti e sulla topologia della rete possono essere raccolti dagli orchestratori e condivisi verso altri attraverso uno specifico formato di encoding del protocollo di routing BGP, il *BGP Network Layer Reachability Information*. È anche possibile adottare differenti policies nella distribuzione di queste informazioni, infatti il protocollo BGP-LS permette di inviare messaggi di advertisement differenti verso gli MdO adiacenti, decidendo in base alle policies inserite. In questo modo le informazioni non vanno semplicemente in flooding verso tutti gli MdO, ma è possibile scegliere quali e quante informazioni inviare.

Nel protocollo BGP-LS sono definiti alcuni attributi per rappresentare e descrivere i link, come ad esempio gli indirizzi IP locale e remoto, gli identificatori delle interfacce locali e remote, la larghezza di banda massima e quella che può essere riservata, le TE metrics etc.

Nell'ambito del progetto 5GEx il protocollo BGP-LS è stato anche esteso in modo da consentire il passaggio di altre due informazioni per rendere più completo il processo di discovery:

- l'URL del entry point del MdO;

- le risorse IT complessive, in termini di numero di CPUs, quantità di memoria volatile e quantità di storage;

Interfaccia I2-RT$_{\text{bilateral}}$

Questa interfaccia mette in comunicazione in modo diretto due MdO in maniera tale che possano scambiarsi informazioni concernenti la topologia delle risorse senza coinvolgere altri MdO. In pratica tramite questa interfaccia due MdO si possono scambiare gli UNIFY Virtualizer data models, ovvero dei modelli che al loro interno contengono un elenco nodi rappresentati come nodi BiS-BiS e un elenco con i tipi di VNF che possono eseguire. La descrizione del nodo BiS-BiS contiene:

- le informazioni riguardanti la quantità di risorse IT disponibili;
- la flow-table, una tabella contenente una serie di record chiamati flow-entry che rappresentano le connessioni tra le interfacce dei nodi BiS-BiS;
- l'elenco dei *SAPs (Service Access Points)*, che costituiscono le interfacce fisiche con le quali un nodo BiS-BiS può comunicare con l'esterno;
- l'elenco delle VNF supportate da quel nodo BiS-BiS.

Lo scambio di informazioni utilizzando questa interfaccia, deve essere fatto utilizzando uno YANG data model. Grazie a questo formalismo qualunque protocollo basato su questo data model può comunicare utilizzando questa interfaccia. Nella versione P1 del MdO l'interfaccia I2-RT$_{\text{bilateral}}$ è stata implementata con il protocollo NETCONF.

Interfaccia I2-RC

Questa interfaccia consente di mettere in comunicazione i Resource Orchestrators di due MdO con lo scopo di consentire ad un RO che fa da client di allocare risorse di rete ed IT presenti in un altro dominio amministrativo e gestite da un altro RO che funge da server. Nella versione P1 del MdO l'interfaccia I2-RC utilizza la UNIFY Virtualizer API, che consente di recuperare la descrizione delle risorse disponibili e allo stesso tempo permette di allocare una parte di quelle risorse.

Anche in questo caso la descrizione delle risorse disponibili avviene utilizzando il modello BiS-BiS node e come per l'interfaccia I2-RT$_{\text{bilateral}}$ viene utilizzato il protocollo NETCONF. Questa interfaccia viene utilizzata nel modo seguente:

- il RO client invia una richiesta NETCONF di tipo get-config al RO server del dominio remoto. La risposta del RO server a questa richiesta contiene uno o più nodi BiS-BiS, con i rispettivi link e le corrispondenti risorse;
- il RO client, quando ha necessità di allocare o deallocare risorse che appartengono al RO server, invia una richiesta NETCONF di tipo edit-config che contiene la risposta alla get-config e le opportune richieste per creare una o più istanze di VNF e per creare le flow-entry nella flow-table.

Interfaccia I3-RC

L'interfaccia I3-RC è simile alla precedente per quanto riguarda l'implementazione, ma con la grossa differenza che quest'ultima serve per consentire la comunicazione tra il RO e i Domain Orchestrators che si trovano nel dominio amministrativo di cui fa parte il MdO. La I3-RC viene utilizzata per acquisire le informazioni sulla disponibilità delle risorse locali e per allocare queste risorse quando richiesto, istanziando VNF e creando dei link sulla rete SDN.

L'interfaccia I3-RC è basata sull'Unify Virtualizer API e utilizza il modello BiS-BiS node per rappresentare le risorse disponibili. Attraverso la richiesta NETCONF di tipo get-config si ottiene una visione dettagliata delle risorse e con il comando edit-config è possibile effettuare il deploy di un grafo che dal MdO passa al DO.

4.4 Interazione tra MdO e FROG4

Il lavoro svolto in questa tesi prevede che sotto il MdO non siano direttamente presenti degli orchestratori di dominio, ma bensì un orchestratore globale, il FROG4 Global Orchestrator. La motivazione è che il FROG4 e i suoi orchestratori di dominio sono compatibili con la rete JOLNet, l'ambiente in cui è stato testato lo use case che sarà presentato in seguito. Come detto precedentemente il Resource Orchestrator del MdO utilizza un modello definito nella Unify Virtualizer Library per poter comunicare con gli orchestratori sottostanti. Il FROG4 invece utilizza una libreria differente chiamata *NF-FG Library*.

Nonostante le due librerie siano state create seguendo le linee guida del progetto Unify, differiscono in alcune parti e non sono nativamente compatibili tra loro. In altre parole i grafi prodotti dal RO non vengono compresi dal FROG4. Inoltre il FROG4 utilizza delle REST API non compatibili con quelle di ESCAPE che usa il protocollo NETCONF. Dunque si è reso necessario inserire un modulo di traduzione che effettua la conversione da un NFFG conforme alla Virtualizer Library in un NFFG che utilizza il modello compatibile con il FROG4. Questo modulo è stato chiamato *Mdo2frog4*. Il Mdo2frog4 comunica con il MdO attraverso l'interfaccia I3-RC attraverso la quale invia una rappresentazione del dominio sottostante, secondo il modello BiS-BiS, e riceve gli NFFG contenenti le NF da eseguire e una serie di flow-entry che rappresentano le varie interconnessioni e le azioni da compiere. Il Mdo2frog4 una volta ricevuto il grafo lo traduce nel formato compatibile con il modello descritto dalla NF-FG Library e attraverso la sua interfaccia di southbound lo invia al FROG4. A questo punto il FROG4 si occuperà di splittare il grafo in più sottografi e inviandoli ai vari orchestratori di dominio. Il modulo di traduzione Mdo2frog4 verrà descritto in modo più approfondito nel capitolo 5.

4.5 Limiti della versione P1

Il MdO utilizzato durante questa attività è identificato come P1 e corrisponde alla prima versione rilasciata di questo software. Essendo la prima versione alcuni aspetti e funzionalità presenti hanno dei limiti.

4.5.1 Aspetti tecnici

Nella versione P1 il MdO assume che i Domain orchestrators e gli altri MdOs siano già configurati e che esportino già una rappresentazione delle loro risorse attraverso le interfacce I3-RC e I2-$RT_{bilateral}$. Una volta terminato il processo di avvio il MdO attraverso la sua interfaccia verso i Domain Orchestrators, acquisisce tutte le informazioni sui domini sotto il controllo del provider locale. Finita questa prima fase di acquisizione, il MdO crea una rappresentazione virtuale delle risorse utilizzando le informazioni fornite dai DOs. Questo processo viene chiamato *Slice composition*.

Da questo punto in poi non sarà più possibile modificare le risorse descritte in questa rappresentazione, ad esempio non sarà possibile inserire nuovi template per le VNF o aggiungere o rimuovere i SAP. Prima di inviare un nuovo NFFG verso gli orchestratori sottostanti, il MdO esegue una chiamata NETCONF get-config con cui riceve l'attuale ed aggiornata rappresentazione delle risorse, ma ancora non è presente una funzione che vada ad aggiornare i vari campi con i nuovi dati. Questo mancato aggiornamento è limitato alle modifiche riguardanti le risorse disponibili, infatti nel momento in cui viene fatto il deploy di un NFFG, il MdO aggiorna il suo modello riportando le VNFs istanziate e le interconnessioni stabilite. Questo permette quindi di fare il deploy di un NFFG, tenerne traccia, ed eventualmente cancellarlo.

4.5.2 Aspetti riguardanti la sicurezza

Nella versione P1 del MdO non sono stati implementati meccanismi di sicurezza, però sono stati analizzati alcuni aspetti critici legati a questo tema che hanno in parte indirizzato lo sviluppo dell'architettura descritta precedentemente. Gli aspetti analizzati sono:

- integrità dei dati e non ripudio;
- autenticazione e controllo degli accessi;
- privacy dei dati;

Come in tutti i software i punti più critici dal punto di vista della sicurezza sono le interfacce con l'esterno. Le interfacce del MdO descritte nel paragrafo 4.3.3 utilizzano dei protocolli (BGP-LS REST e NETCONF) che supportano l'uso di protocolli standardizzati che assicurano che i tre aspetti di sicurezza elencati sopra vengano osservati. In particolare i protocolli REST e NETCONF supportano l'uso di TLS (aggiungere reference per tls) mentre il BGP-LS supporta l'autenticazione dei peer utilizzando un canale IPsec.

In conclusione i meccanismi di sicurezza possono essere implementati nelle prossime versioni del MdO senza apportare cambiamenti o modifiche rilevanti alla sua architettura.

Capitolo 5

Mdo2frog4

Il Mdo2frog4 è un modulo software intermedio che opera tra il FROG4 Global Orchestrator e il Resource Orchestrator del MdO. Possiede due interfacce differenti, una di northbound che comunica con il RO del MdO e utilizza il protocollo NETCONF, l'altra di southbound è una REST che comunica con il FROG4.

5.1 Virtualizer Library

Il RO e il Mdo2frog4 comunicano attraverso dei formalismi definiti nella Virtualizer Library. La Virtualizer Library è un'implementazione di riferimento scritta in Python del Virtualizer data model. Questa libreria permette di validare ed eseguire il parsing di un file XML conforme al Virtualizer YANG model, il modello utilizzato per definire i NFFG in ESCAPE. Con questo stesso modello vengono descritti i nodi BiS-BiS che poi verranno esaminati dal RO durante la procedura di orchestrazione.

Listing 5.1. Rappresentazione ad albero del Virtualizer model

```
1  +--rw virtualizer
2     +--rw id?      string
3     +--rw name?    string
4     +--rw nodes
5     |  +--rw node* [id]
6     |     +--rw id              string
7     |     +--rw name?           string
8     |     +--rw type            string
9     |     +--rw ports
10    |     |  +--rw port* [id]
11    |     |     +--rw id             string
12    |     |     +--rw name?          string
13    |     |     +--rw port_type?     string
14    |     |     +--rw capability?    string
15    |     |     +--rw sap?           string
16    |     +--rw links
17    |     |  +--rw link* [src dst]
18    |     |     +--rw id?          string
19    |     |     +--rw name?        string
20    |     |     +--rw src          ->
21    |     |     +--rw dst          ->
22    |     |     +--rw resources
23    |     |        +--rw delay?       string
24    |     |        +--rw bandwidth?   string
25    |     +--rw resources
26    |     |  +--rw cpu        string
27    |     |  +--rw mem        string
28    |     |  +--rw storage    string
29    |     +--rw NF_instances
30    |     |  +--rw node* [id]
31    |     |     +--rw id           string
32    |     |     +--rw name?        string
33    |     |     +--rw type?        string
```

```
34      |       |       +--rw ports
35      |       |       |  +--rw port* [id]
36      |       |       |     +--rw id              string
37      |       |       |     +--rw name?           string
38      |       |       |     +--rw port_type?      string
39      |       |       |     +--rw capability?     string
40      |       |       |     +--rw sap?            string
41      |       |       +--rw links
42      |       |       |  +--rw link* [src dst]
43      |       |       |     +--rw id?           string
44      |       |       |     +--rw name?         string
45      |       |       |     +--rw src           ->
46      |       |       |     +--rw dst           ->
47      |       |       |     +--rw resources
48      |       |       |        +--rw delay?       string
49      |       |       |        +--rw bandwidth?   string
50      |       |       +--rw resources
51      |       |          +--rw cpu         string
52      |       |          +--rw mem         string
53      |       |          +--rw storage     string
54      |       +--rw capabilities
55      |       |  +--rw supported_NFs
56      |       |     +--rw node* [id]
57      |       |        +--rw id            string
58      |       |        +--rw name?         string
59      |       |        +--rw type?         string
60      |       |        +--rw ports
61      |       |        |  +--rw port* [id]
62      |       |        |     +--rw id              string
63      |       |        |     +--rw name?           string
64      |       |        |     +--rw port_type?      string
65      |       |        |     +--rw capability?     string
66      |       |        |     +--rw sap?            string
67      |       |        +--rw links
68      |       |        |  +--rw link* [src dst]
69      |       |        |     +--rw id?           string
70      |       |        |     +--rw name?         string
71      |       |        |     +--rw src           ->
72      |       |        |     +--rw dst           ->
73      |       |        |     +--rw resources
74      |       |        |        +--rw delay?       string
75      |       |        |        +--rw bandwidth?   string
76      |       |        +--rw resources
77      |       |           +--rw cpu         string
78      |       |           +--rw mem         string
79      |       |           +--rw storage     string
80      |       +--rw flowtable
81      |          +--rw flowentry* [id]
82      |             +--rw id            string
83      |             +--rw name?         string
84      |             +--rw priority?     string
85      |             +--rw port          ->
86      |             +--rw match         string
87      |             +--rw action        string
88      |             +--rw out?          ->
89      |             +--rw resources
90      |                +--rw delay?       string
91      |                +--rw bandwidth?   string
92      +--rw links
93         +--rw link* [src dst]
94            +--rw id?           string
95            +--rw name?         string
96            +--rw src           ->
97            +--rw dst           ->
98            +--rw resources
99               +--rw delay?       string
100              +--rw bandwidth?   string
```

Nella rappresentazione ad albero riportata sopra si notano bene tutti i campi del NFFG, ovvero il file xml che viene letto e modificato dal RO attraverso l'interfaccia I3-RC. Il NFFG è costituito da un header contenente le informazioni di base e da un elenco di BiS-BiS nodes.

I nodi BiS-BiS sono identificati da un id, e caratterizzati da un nome e un tipo. Ogni nodo possiede:

- un campo **ports** in cui è possibile descrivere quali sono le porte, il cui tipo può essere o SAP o abstract, utilizzabili per le interconnessioni.

- Un campo **resources** in cui sono presenti le risorse disponibili nel nodo in termini di CPU, memoria e storage.

- Un campo **NF_instances** in cui sono presenti le NFs di cui è già stato fatto il deploy o che sono in fase di deploy. Sfruttando la natura ricorsiva del Virtualizer model, all'interno di ogni nodo di questo campo, si trovano nuovamente i campi ports e resources che sono analoghi a quelli descritti in precedenza, con la differenza che qui si riferiscono alle interfacce che avrà la VNF e alle risorse che le verranno riservate.

- Un campo **capabilities** che contiene la lista delle Network Function disponibili e che possono essere utilizzate. Ogni NF disponibile è descritta come un nodo e quindi ha anch'essa tutti i campi necessari a caratterizzarne id, nome, tipo, porte e risorse richieste. È possibile creare istanze di NF solo se queste sono presenti tra le capabilities di un nodo BiS-BiS.

- Un campo **flow table**, una tabella che contiene una lista di **flow entry**. Ogni flow entry è identificata da un id univoco all'interno del NFFG e possiede un nome e una priorità. Una flow entry è una regola che stabilisce un percorso che può essere compiuto da un flusso di pacchetti e al quale possono essere applicati dei filtri e delle azioni. I campi **port**, **out**, **match** e **action** servono a stabilire e caratterizzare questo percorso. Il campo port indica qual è la porta di input, ovvero la porta sulla quale arrivano i pacchetti. Il campo out indica qual è la porta alla quale devono essere mandati i pacchetti, dunque la porta di output. Il campo match serve per indicare una discriminante per filtrare i pacchetti (e.g. il source MAC address). Il campo action permette di eseguire delle azioni sui pacchetti prima che questi vengano mandati sulla porta di output.

Nel modello ad albero della listing 5.1 in ogni nodo è presente anche un campo **links**. Quest'ultimo serve, nel caso si trovi all'interno di un nodo BiS-BiS, per definire dei link virtuali ai quali è possibile associare determinate caratteristiche. Ad esempio è possibile specificare il delay su un link tra due porte. È anche possibile specificare dei links nella struttura più esterna del NFFG, in modo da definire dei link tra porte di differenti nodi BiS-BiS.

5.2 Interfaccia verso il FROG4

Il Mdo2frog4 e il FROG4 comunicano tra loro attraverso una interfaccia REST. Il FROG4 Global Orchestrator espone diverse API grazie alle quali è possibile eseguire le principali azioni necessarie al deploy di un NFFG. Nello specifico il Mdo2frog4 quando deve inviare al FROG4 un grafo esegue prima una chiamata *POST* alla risorsa "login" esposta dall'orchestratore. Le credenziali per poter fare questo login sono contenute nel file di configurazione del Mdo2frog4. Se la procedura di login va a buon fine l'orchestratore restituisce un token di sessione, che viene inserito

dal Mdo2frog4 nell'header dei successivi messaggi che verranno scambiati con il FROG4. Una volta eseguito il login ed ottenuto il token il Mdo2frog4 può inviare all'orchestratore un NFFG concorde con il modello definito nella NF-FG Library utilizzando il metodo *PUT* oppure può inviare una richiesta di cancellazione di un grafo utilizzando il metodo *DELETE* e inserendo nel path della richiesta l'id del grafo. Al momento l'id del NFFG non viene definito dall'utente come parametro del grafo ma viene restituito dal FROG4 come risposta alla *PUT* quando questa ha esito positivo.

5.2.1 NF-FG Library

Analogamente alla Virtualizer Library, la NF-FG Library definisce un modello per descrivere il grafo utilizzato dal FROG4. Questo modello è costituito principalmente da tre parti:

- una lista contenente le VNFs. Per ogni VNF è specificato l'id, il nome, la lista delle porte, il dominio e il *vnf_template*, un template che contiene tutte le informazioni legate all'immagine della VNF che deve essere istanziata, come le risorse minime richieste, il numero delle porte e l'URI. Nello specifico caso affrontato in questa tesi, l'URI corrisponde ad un URL che punta all'immagine della VNF contenuta nel controller node di Openstack e gestita dal image service *Glance*.

- una lista contenente gli Endpoint. Per ogni Endpoint è specificato l'id, il nome, il dominio e il tipo. L'infrastruttura JOLNet consente di utilizzare soltanto endpoint di tipo vlan, per cui ad ogni Endpoint è anche associato un campo vlan che contiene a sua volta un *vlan_id*, un *if_name* e un *node_id*. Il vlan_id è l'identificativo della vlan, l'if_name corrisponde al identificativo di una porta e il node_id è utilizzato per specificare l'id del dispositivo a cui appartiene la porta. E.g. nel caso in cui un Endpoint sia associato alla porta di uno switch SDN, il node_id corrisponderà all'id dello switch e il if_name all'id della porta appartenente a quello switch.

- un Big-Switch, ovvero una lista contenente delle flow-rule. Ogni flow-rule è caratterizzata da un id, una priorità, una lista di match e una lista di actions.

5.3 Configurazione Mdo2frog4 e rappresentazione del dominio

Nel Capitolo 4 è stato spiegato come il MdO all'avvio esegua una chiamata getconfig verso il Mdo2frog4 per ottenere il file xml con la rappresentazione BiS-BiS del dominio sottostante. Questo file viene creato in modo automatico dal Mdo2frog4 al suo avvio, partendo dalle informazioni contenute nei suoi file di configurazione, più precisamente nei file **template.xml** e **port_info.xml**.

Nel file template.xml devono essere inserite tutte le VNFs che sono istanziabili dal dominio. Questo file è conforme alla descrizione delle VNFs secondo la struttura del modello della Virtualizer Library.

Nel file port_info.xml devono essere inserite tutte le porte utilizzabili nel dominio. Queste porte verranno inserite nel campo ports del nodo BiS-BiS, quindi anche in questo caso viene seguito il modello della Virtualizer Library.

Le informazioni contenute nei due file sopracitati verranno inserite nel file xml, validate utilizzando il modello della Virtualizer Library ed inviate al MdO quando quest'ultimo ne farà richiesta.

Ai fini del processo di traduzione, le informazioni sulle porte devono essere inserite in un secondo file, coerente con il file port_info.xml, utilizzato per associare ad ogni porta il dominio a cui appartiene (e.g. OpenStack) e il vlan_id. Queste informazioni saranno necessarie durante la traduzione da porta sap ad Endpoint.

5.4 Processo di traduzione

Il Mdo2frog4 rende possibile la comunicazione tra MdO e FROG4 mappando i campi del NFFG della Virtualizer Library, nei corrispettivi campi del NFFG coerente con il modello usato dal FROG4. Pur essendo stati sviluppati partendo dalle indicazioni stabilite durante il progetto FP7 Unify, come si evince dalla descrizione fatta precedentemente, i due modelli differiscono abbastanza l'uno dall'altro e questo crea delle criticità durante il processo di traduzione.

La differenza più marcata è che mentre il FROG4 gestisce più NFFGs separati tra loro, ESCAPE utilizza un unico NFFG nel quale sono inserite tutte le VNF e tutte le flow-entry appartenenti ad un dato nodo BiS-BiS. Questo è dato dal fatto che ESCAPE utilizza il protocollo NETCONF e dunque va ad editare un unico file che rappresenta lo status aggiornato del dominio.

Un secondo problema da affrontare e gestire durante il processo di traduzione consiste nel fatto che il MdO non ha una visione completa del dominio sottostante, ma vede i vari domini (OpenStack e SDN) come un unico nodo BiS-BiS. Questo comporta delle criticità nel momento in cui traducendo si devono specificare i domini in cui fare il deploy di una VNF o di un Endpoint, in quanto non è possibile specificare queste informazioni nel grafo da inviare al MdO. Infatti il MdO possiede una rappresentazione ad alto livello del dominio sottostante. In particolare conosce solamente quali sono i SAP, le risorse complessive e le VNF che sono supportate ed istanziabili.

All'arrivo di un NFFG sull'interfaccia di northbound, il Mdo2frog4 procede ad arricchire il contenuto del NFFG con le informazioni sulle porte del nodo BiS-BiS. Infatti il MdO non inserisce tali informazioni nel NFFG al momento dell'invio, quindi si rende necessario immetterle in modo da poter eseguire il processo di validazione, che altrimenti fallirebbe. Nella listing 5.2 è possibile vedere un esempio del grafo che il Mdo2frog4 riceve dal MdO.

Listing 5.2. Esempio di un grafo ricevuto dal Mdo2frog4

```
1  <?xml version="1.0" ?>
2  <virtualizer>
3     <nodes>
4        <node>
5           <id>UUID11</id>
6           <NF_instances>
7              <node operation="create">
8                 <id>2</id>
9                 <name>mca-fw1</name>
10                <type>mca-fw1</type>
11                <ports>
```

```
12                      <port>
13                          <id>1</id>
14                          <name>ovs1_eth0</name>
15                          <port_type>port-sap</port_type>
16                      </port>
17                      <port>
18                          <id>4</id>
19                          <name>ovs1_eth1</name>
20                          <port_type>port-sap</port_type>
21                      </port>
22                  </ports>
23                  <resources>
24                      <cpu>1.0</cpu>
25                      <mem>1.0</mem>
26                      <storage>0.0</storage>
27                  </resources>
28              </node>
29          </NF_instances>
30          <flowtable>
31              <flowentry operation="create">
32                  <id>21</id>
33                  <priority>100</priority>
34                  <port>../../../NF_instances/node[id=2]/ports/port[id=0]</port>
35                  <out>../../../ports/port[id=1]</out>
36                  <resources>
37                      <bandwidth>0.0</bandwidth>
38                  </resources>
39              </flowentry>
40              <flowentry operation="create">
41                  <id>22</id>
42                  <priority>100</priority>
43                  <port>../../../ports/port[id=1]</port>
44                  <match>source_mac=fa:16:3e:3d:f8:96</match>
45                  <out>../../../NF_instances/node[id=2]/ports/port[id=0]</out>
46                  <resources>
47                      <bandwidth>0.0</bandwidth>
48                  </resources>
49              </flowentry>
50              <flowentry>
51                  <id>23</id>
52                  <priority>100</priority>
53                  <port>../../../NF_instances/node[id=2]/ports/port[id=1]</port>
54                  <out>../../../ports/port[id=4]</out>
55                  <resources>
56                      <bandwidth>0.0</bandwidth>
57                  </resources>
58              </flowentry>
59              <flowentry>
60                  <id>24</id>
61                  <priority>100</priority>
62                  <port>../../../ports/port[id=4]</port>
63                  <out>../../../NF_instances/node[id=2]/ports/port[id=1]</out>
64                  <resources>
65                      <bandwidth>0.0</bandwidth>
66                  </resources>
67              </flowentry>
68          </flowtable>
69      </node>
70  </nodes>
71 </virtualizer>
```

Il processo di validazione, compiuto dalla Virtualizer Library, oltre a validare la sintassi del grafo controlla che i collegamenti tra le porte espressi nelle flow-entry della flow-table siano validi. Una volta completata questa parte il NFFG è pronto per essere parsificato e tradotto. La traduzione avviene in due fasi, nella prima vengono recuperate le informazioni sulle VNFs e nella seconda quelle sulle flowrules.

La traduzione delle VNFs consiste in un parsing del NFFG utilizzando le funzioni rese disponibili dalla Virtualizer Library. In tal modo si ottiene un oggetto contenente una lista di NF. A questo punto vengono seguiti i passi descritti nell'**algoritmo 1**.

Partendo dalla lista di VNFs ricavata dalla parsificazione del NFFG, l'algoritmo per ogni VNF estrapola id, nome, tipo in modo diretto. In seguito per ogni porta

Algorithm 1 Traduzione delle NF_instances.

```
1: procedure extract_vnf(nfs)
2: for all nf ∈ nfs do
3:    if nf.operation = 'delete' then
4:       graphid := nf.id
5:    else if nf.operation = 'create' then
6:       graphid := nf.id
7:       vnfName :=nf.name
8:       vnfType := nf.type
9:       for all port_id ∈ nf.ports.port do
10:         port := nf.ports[port_id]
11:         if port.address.l3 ≠ 0 then
12:            unify_ip := port.address.l3
13:         end if
14:         mac := port.address.l2
15:         port_id_nffg := port_id
16:         port_list += Port(port_id_nffg, unify_ip, mac)
17:      end for
18:      nf_instances += VNF(nf.id, vnfName, vnfType, port_list)
19:   end if
20: end for
21: return nf_instances
```

ottiene l'id, fa un check su un eventuale indirizzo IP presente, inserisce l'indirizzo MAC e crea l'elemento **Port** (linea `#15`) aggiungendolo alla lista delle porte da inserire nel NFFG da inviare al FROG4. Finite le iterazioni sulle porte viene creato l'elemento **VNF**(linea `#17`) ed aggiunto all'elenco delle VNF. Infine viene ritornato l'elenco delle VNF coerenti col modello descritto dalla NF-FG Library. L'id con cui verrà identificato il grafo in questo caso corrisponde all'id della VNF e una volta che il grafo sarà inviato al FROG4, si creerà una corrispondenza tra l'id salvato e quello restituito dal FROG4. In questo modo sarà possibile cancellare il NFFG a seguito della richiesta di cancellazione della VNF.

Terminato il processo di traduzione delle VNFs viene richiamata la funzione che traduce le flow-entry in flow-rule. Questa funzione riceve il contenuto intero del NFFG e ne esegue la parsificazione ricavandone la flow-table.

Ottenuta la flow-table, l'**algoritmo 2** per ogni flow-entry che deve essere istanziata, crea una flowrule e le assegna l'id e la priorità corrispondenti a quelli della flow-entry. Se ad una flowentry è associato un match, questo viene tradotto nell'equivalente del modello della NFFG Library e inserito nell'elenco dei match associati alla nuova flowrule (linee `#6 - #11`).

A questo punto viene preso in esame il contenuto del campo port della flow-entry (linea `#12`). Se in questo contenuto non è presente la parola chiave 'NF_instances' significa che la porta in esame è una porta del nodo BiS-BiS. In questo caso verrà quindi tradotta come un endpoint. Dunque si procede andando ad recuperare l'id dell'endpoint corrispondente a quella porta, si recuperano le informazioni riguardo al dominio e al vlan_id e si aggiunge l'endpoint alla lista degli endpoint da istanziare per quel NFFG. Se nel nome della porta è presente la stringa 'of', l'endpoint viene considerato come un endpoint appartenente al dominio SDN e associato ad una

interfaccia di uno switch. In caso contrario viene considerato un endpoint associato al dominio OpenStack e caratterizzato di conseguenza (linee #13 - #25). Infine l'endpoint appena creato viene aggiunto alla flowrule come porta in input nel campo match (linea #26). Nel caso in cui la porta corrisponda ad una porta di una VNF, questa viene aggiunta alla flowrule come porta in input nel campo match(linea #29), ma chiaramente non viene creato nessun endpoint.

In seguito viene esaminato il campo actions della flow-entry e nel caso vi siano delle azioni, queste vengono tradotte nel loro equivalente del modello della NFFG Library e aggiunte alle action della flowrule.

Successivamente viene esaminato il contenuto del campo out della flow-entry (linea #37). In questo caso le operazioni non sono riportate ma sono analoghe a quanto visto per il campo port, con l'eccezione che stavolta la porta viene inserita come porta di output tra le actions della flowrule.

Infine la flowrule viene aggiunta alla lista delle flowrules da istanziare e viene restituita insieme alla lista degli endpoint al termine di questo algoritmo. A questo punto si hanno tutti gli elementi per poter costruire il NF-FG, che sarà inviato al FROG4 Global Orchestator. Nella listing 5.3 è possibile vedere un esempio del grafo che il Mdo2frog4 riceve dal MdO.

Listing 5.3. Esempio di un grafo tradotto dal Mdo2frog4

```
1  {
2    "forwarding-graph": {
3      "VNFs": [
4        {
5          "ports": [
6            {
7              "id": "port:0"
8            },
9            {
10             "id": "port:1"
11           }
12         ],
13         "functional-capability": "mca-fw1",
14         "id": "2",
15         "name": "mca-fw1"
16       }
17     ],
18     "big-switch": {
19       "flow-rules": [
20         {
21           "priority": 100,
22           "actions": [
23             {
24               "output_to_port": "endpoint:1"
25             }
26           ],
27           "id": "21",
28           "match": {
29             "port_in": "vnf:2:port:0"
30           }
31         },
32         {
33           "priority": 100,
34           "actions": [
35             {
36               "output_to_port": "vnf:2:port:0"
37             }
38           ],
39           "id": "22",
40           "match": {
41             "source_mac": "fc:4d:e2:56:9f:19",
42             "port_in": "endpoint:1"
43           }
44         },
45         {
46           "priority": 100,
47           "actions": [
48             {
```

```
49                "output_to_port": "endpoint:4"
50              }
51            ],
52            "id": "23",
53            "match": {
54              "port_in": "vnf:2:port:1"
55            }
56          },
57          {
58            "priority": 100,
59            "actions": [
60              {
61                "output_to_port": "vnf:2:port:1"
62              }
63            ],
64            "id": "24",
65            "match": {
66              "port_in": "endpoint:4"
67            }
68          }
69        ]
70      },
71      "name": "NF-FG",
72      "end-points": [
73        {
74          "domain": "openstack",
75          "vlan": {
76            "if-name": "4",
77            "vlan-id": "271"
78          },
79          "type": "vlan",
80          "id": "4",
81          "name": "External_port_2"
82        },
83        {
84          "domain": "onos_domain",
85          "vlan": {
86            "if-name": "5100",
87            "vlan-id": "270",
88            "node-id": "of:000064e9505a82c0"
89          },
90          "type": "vlan",
91          "id": "1",
92          "name": "of:000064e9505a82c0/5100"
93        }
94      ]
95    }
96  }
```

5.5 Aggiornamento della rappresentazione del dominio

Come illustrato nel paragrafo precedente il Mdo2frog4 genera un file xml partendo dalla sua configurazione iniziale, che contiene la rappresentazione BiS-BiS del dominio sottostante. Questo file deve aggiornarsi in modo coerente alle operazioni di create e delete che vengono effettuate tutte le volte che un NFFG giunge sulla sua interfaccia di northbound. L'aggiornamento di questo file consiste nell'inserire o nel cancellare le NF o le flow-entry, a seconda delle richieste provenienti dal MdO. Questo processo di aggiornamento è importante per il fatto che il file xml contenuto nel Mdo2frog4 e quello che si costruisce internamente il MdO devono corrispondere. Dunque MdO e Mdo2frog4 devono sempre essere sincronizzati. La funzione che si occupa dell'aggiornamento del file xml sul Mdo2frog4 esegue il parsing del file non aggiornato e del NFFG arrivato dal MdO, ne calcola le differenze e aggiunge o cancella le NF o le flow-entry. Ovviamente questa funzione viene eseguita soltanto dopo che il NFFG è stato processato con successo dal FROG4 orchestrator.

Algorithm 2 Traduzione della Flow Table.

```
1: for all flowentry ∈ flowtable do
2:   if flowentry.operation = 'create' then
3:     flowrule = FlowRule()
4:     flowrule.id := flowentry.id
5:     flowrule.priority := flowentry.priority
6:     if flowentry.match ≠ NULL then
7:       for all match ∈ flowentry.match do
8:         newMatch := convertToEquivalentMatch(match)
9:         flowrule +=newMatch
10:      end for
11:    end if
12:    port := flowentry.port
13:    if 'NF_instances' ∋ port.path then
14:      port_name := physicalPortsVirtualization(port.name)
15:      port_id = findEndPointId(port_name)
16:      e_domain := endpoints_domain[port_id]
17:      e_vlanid := endpoints_vlanid[port_id]
18:      if 'of' ∈ port_name then
19:        token := split(port_name)
20:        node_id = token[0]
21:        interface = token[1]
22:        endpoints += EndPoint(port_id, port.name, type_vlan, e_vlanid,
           interface, node_id, e_domain)
23:      else
24:        endpoints += EndPoint(port_id, port.name, type_vlan, e_vlanid,
           e_domain)
25:      end if
26:      match.port_in := 'endpoint:' + endpoints[port_name]
27:    else
28:      vnf_id = port.getVnf().id
29:      match.port_in := 'vnf:' + vnf_id + ":port:" + port_id
30:    end if
31:    if flowentry.action ≠ NULL then
32:      for all action ∈ flowentry.action do
33:        newAction := convertToEquivalentAction(action)
34:        flowrule.action += newAction
35:      end for
36:    end if
37:    port := flowentry.out
38:    if 'NF_instances' ∈ port.path then
39:      ...
40:      flowrule.action += Action(output = 'endpoint:' + endpoints[port_name]
41:    else
42:      vnf_id = port.getVnf().id
43:      flowrule.action += Action(output = 'vnf:' + vnf_id + ":port:" + port_id)
44:    end if
45:  end if
46:  flowrules += flowrule
47: end for
48: return flowrules, endpoints
```

Capitolo 6

Service Layer

Il Service Layer è il modulo posto alla cima dell'architettura finora presentata. Comunica direttamente con il MdO e si occupa di generare un grafo di servizio per un utente che ne fa richiesta. Partendo da un grafo salvato in un file e specifico per un determinato utente, il Service Layer aggiunge delle regole che identificano il device che l'utente sta utilizzando e instradano il traffico in modo che possa raggiungere l'esterno della rete.

6.1 Interfaccia verso l'utente

Il Service Layer espone un'interfaccia alla quale è possibile inviare una serie di dati organizzati in un JSON tramite il metodo PUT. Nell'intestazione del messaggio devono essere fornite le credenziali dell'utente che viene autenticato dal Service Layer stesso, il quale possiede un database con l'elenco degli utenti registrati. Nel messaggio JSON sono presenti le informazioni necessarie alla creazione delle regole da inserire nel grafo di servizio, nello specifico la posizione dell'utente e il MAC address del dispositivo che l'utente sta utilizzando per fare la richiesta di login.

Nel caso specifico di un utente della JOLNet, la posizione viene identificata dall'ID e dalla porta dello switch SDN a cui l'utente è connesso. Di seguito un esempio del body del messaggio che il Service Layer si aspetta di ricevere.

Listing 6.1. Body del messaggio da inviare al Service Layer

```
{
  "session": {
    "device": {
      "mac": "fc:4d:e2:56:9f:19",
      "port": "of:000064e9505a82c0/5100"
    }
  }
}
```

6.2 Interfaccia verso il MdO

Attraverso la sua interfaccia di southbound il Service Layer comunica con il MdO. A differenza della versione precedente, in cui il Service Layer comunicava direttamente con il FROG4 Global Orchestrator, in questo caso l'interfaccia permette di eseguire solo le due operazioni permesse dal paradigma NETCONF, quindi delle POST alle risorse */get-config* ed */edit-config*. Con la prima il Service Layer è in

grado di ottenere la configurazione del MdO, ricevendo un XML che contiene oltre alla rappresentazione del dominio, le NF e le regole attive in quel momento. Con la seconda invece è possibile inviare un grafo al MdO, sempre seguendo il formato definito nella Virtualizer Library. Il grafo che deve essere istanziato, per essere valido deve contenere oltre a tutte le risorse da istanziare, anche i riferimenti a tutte le NF e alle regole che sono già attive nel dominio.

Per rispondere a questa necessità il Service Layer una volta generato il grafo di servizio, ne estrapola gli elementi da istanziare e li inserisce nel XML ottenuto in precedenza. Infine invia la nuova configurazione così ottenuta al MdO.

6.3 Procedura per la generazione del grafo di servizio

Dalle informazioni contenute nel JSON ricevuto dall'interfaccia di Northbound, il Service Layer è in grado di generare un flusso bidirezionale che permette al traffico dell'utente di raggiungere la network function che ha richiesto di istanziare. All'arrivo sull'interfaccia di northbound del JSON descritto in precedenza, il Service Layer una volta autenticato l'utente, inizia la procedura di creazione del grafo di servizio. Questo processo inizia con una interrogazione al database interno, per trovare il nome del file contenente una prima versione del grafo dell'utente (listing 6.2), nel quale sono presenti una VNF e due regole, con le quali sarà possibile raggiungere l'esterno della rete. Viene eseguito un primo parsing del grafo, in modo da ottenere l'identificativo della Network Function. Questo ID, che deve essere univoco in tutto il dominio, viene utilizzato per generare gli ID delle regole necessarie a realizzare il flusso bidirezionale.

Listing 6.2. Esempio di un grafo di servizio prima della procedura di aggiunta delle regole

```
<?xml version="1.0" ?>
<virtualizer>
   <id>SingleBiSBiS</id>
   <name>Single-BiSBiS-View</name>
   <nodes>
      <node>
         <id>SingleBiSBiS</id>
         <name>SingleBiSBiS</name>
         <type>BiSBiS</type>
         <ports>
            <port>
               <id>sap4</id>
               <name>External_port_2</name>
               <port_type>port-sap</port_type>
               <sap>sap4</sap>
            </port>
         </ports>
         <NF_instances>
            <node operation="create">
               <id>2</id>
               <name>mca-fw1</name>
               <type>cirros</type>
               <ports>
                  <port>
                     <id>0</id>
                     <name>eth0</name>
                  </port>
                  <port>
                     <id>1</id>
                     <name>eth1</name>
                  </port>
               </ports>
               <resources>
```

```
34                      <cpu>1.0</cpu>
35                      <mem>1.0</mem>
36                      <storage>0.0</storage>
37                  </resources>
38              </node>
39          </NF_instances>
40          <capabilities>
41              <supported_NFs>
42                  <node>
43                      <id>cirros</id>
44                      <type>cirros</type>
45                  </node>
46                  <node>
47                      <id>mca-fw1</id>
48                      <type>mca-fw1</type>
49                  </node>
50                  <node>
51                      <id>mca-fw2</id>
52                      <type>mca-fw2</type>
53                  </node>
54              </supported_NFs>
55          </capabilities>
56          <flowtable>
57              <flowentry operation="create">
58                  <id>23</id>
59                  <priority>100</priority>
60                  <port>../../../NF_instances/node[id=2]/ports/port[id=1]</port>
61                  <out>/virtualizer/nodes/node[id=SingleBiSBiS]/ports/port[id=sap4]</
                        out>
62                  <resources>
63                      <bandwidth>0</bandwidth>
64                  </resources>
65              </flowentry>
66              <flowentry operation="create">
67                  <id>24</id>
68                  <priority>100</priority>
69                  <port>/virtualizer/nodes/node[id=SingleBiSBiS]/ports/port[id=sap4]</
                        port>
70                  <out>../../../NF_instances/node[id=2]/ports/port[id=1]</out>
71                  <resources>
72                      <bandwidth>0</bandwidth>
73                  </resources>
74              </flowentry>
75          </flowtable>
76      </node>
77  </nodes>
78 </virtualizer>
```

Il Service Layer a questo punto contatta il MdO per ottenere la configurazione attuale. Dalla risposta del MdO il Service Layer recupera l'elenco delle interfacce appartenenti al nodo BiS-BiS, verifica che la stringa che identifica la posizione dell'utente corrisponda ad una di queste interfacce e ne ricava l'identificativo.

Terminata questa parte, il Service Layer ha tutte le informazioni necessarie alla creazione delle regole. Dall'ID della NF ottenuto in precedenza si calcolano gli IDs delle regole da aggiungere al grafo, si imposta una priorità di default e si generano le stringhe che andranno a costituire la porta di ingresso e la porta di uscita del traffico. Le due porte coinvolte saranno la porta SAP ricavata dalla posizione dell'utente e la porta con id uguale a 0 della Network Function che verrà istanziata. Vengono quindi inserite manualmente queste due regole che rappresentano il flusso bidirezionale che va dall'interfaccia dell'utente a quella della NF che verrà istanziata (algoritmo 3). A questo punto viene effettuato il parsing del nuovo grafo, in modo tale da eseguire una nuova validazione, e in seguito viene aggiunto, nella regola che gestisce l'ingresso del traffico dell'utente sulla rete, il match con il MAC address del dispositivo indicato nel messaggio JSON ricevuto (algoritmo 4).

Al termine del processo di aggiunta delle regole, il grafo di servizio è pronto per essere istanziato e appare come nell'esempio riportato nella listing 6.3.

Algorithm 3 Generazione delle regole

```
1: procedure createRules(mdo_config, nf_id)
2: tree = ET.ElementTree(ET.fromstring(mdo_config))
3: root = tree.getroot()
4: Infrastructure = Virtualizer.parse(root=root)
5: ports = Infrastructure.nodes.node['SingleBiSBiS'].ports
6: for all port ∈ ports do
7:    if port.name.get_value() = user_location then
8:       port_id = port.id.get_value()
9:       id_flow1 = nf_id + 1
10:      id_flow2 = nf_id + 2
11:      nf_port = '../../../NF_instances/node[id=' + nf_id + ']/ports/port[id=0]'
12:      sap_port = '/virtualizer/nodes/node[id=SingleBiSBiS]/ports/port[id=' +
         id_port + ']'
13:      new_mdo_config = add_bidirectional_flow(nffg, id_flow1, id_flow2, nf_port,
         sap_port)
14:      return new_mdo_config
15:   end if
16: end for
```

Algorithm 4 Aggiunta del match con il MAC address

```
1: procedure addMac(new_mdo_config, mac_address)
2: tree = ET.ElementTree(ET.fromstring(new_mdo_config))
3: root = tree.getroot()
4: Infrastructure = Virtualizer.parse(root=root)
5: Flowtable = Infrastructure.nodes.node['SingleBiSBiS'].flowtable
6: match_string = "source_mac=" + mac_address
7: for all flow_entry ∈ Flowtable do
8:    port_path = flow_entry.port.get_value()
9:    if "sap" in port_path then
10:      flow_entry.match.data = match_string
11:      return Infrastructure.xml()
12:   end if
13: end for
```

Listing 6.3. Esempio di un grafo di servizio dopo della procedura di aggiunta delle regole

```
<?xml version="1.0" ?>
<virtualizer>
   <id>SingleBiSBiS</id>
   <name>Single-BiSBiS-View</name>
   <nodes>
      <node>
         <id>SingleBiSBiS</id>
         <name>SingleBiSBiS</name>
         <type>BiSBiS</type>
         <ports>
            <port>
               <id>sap1</id>
               <name>of:000064e9505a82c0/5100</name>
               <port_type>port-sap</port_type>
               <sap>sap1</sap>
            </port>
            <port>
               <id>sap4</id>
               <name>External_port_2</name>
               <port_type>port-sap</port_type>
```

```
21                <sap>sap4</sap>
22             </port>
23          </ports>
24          <NF_instances>
25             <node operation="create">
26                <id>2</id>
27                <name>mca-fw1</name>
28                <type>cirros</type>
29                <ports>
30                   <port>
31                      <id>0</id>
32                      <name>eth0</name>
33                   </port>
34                   <port>
35                      <id>1</id>
36                      <name>eth1</name>
37                   </port>
38                </ports>
39                <resources>
40                   <cpu>1.0</cpu>
41                   <mem>1.0</mem>
42                   <storage>0.0</storage>
43                </resources>
44             </node>
45          </NF_instances>
46          <capabilities>
47             <supported_NFs>
48                <node>
49                   <id>cirros</id>
50                   <type>cirros</type>
51                </node>
52                <node>
53                   <id>mca-fw1</id>
54                   <type>mca-fw1</type>
55                </node>
56                <node>
57                   <id>mca-fw2</id>
58                   <type>mca-fw2</type>
59                </node>
60             </supported_NFs>
61          </capabilities>
62          <flowtable>
63             <flowentry operation="create">
64                <id>21</id>
65                <priority>100</priority>
66                <port>../../../NF_instances/node[id=2]/ports/port[id=1]</port>
67                <out>/virtualizer/nodes/node[id=SingleBiSBiS]/ports/port[id=port-sap
                      1]</out>
68                <resources>
69                   <bandwidth>0</bandwidth>
70                </resources>
71             </flowentry>
72             <flowentry operation="create">
73                <id>22</id>
74                <priority>100</priority>
75                <port>/virtualizer/nodes/node[id=SingleBiSBiS]/ports/port[id=port-
                      sap1]</port>
76                <match>source_mac=fa:16:3e:3d:f8:96</match>
77                <out>../../../NF_instances/node[id=2]/ports/port[id=1]</out>
78                <resources>
79                   <bandwidth>0</bandwidth>
80                </resources>
81             </flowentry>
82             <flowentry operation="create">
83                <id>23</id>
84                <priority>100</priority>
85                <port>../../../NF_instances/node[id=2]/ports/port[id=1]</port>
86                <out>/virtualizer/nodes/node[id=SingleBiSBiS]/ports/port[id=sap4]</
                      out>
87                <resources>
88                   <bandwidth>0</bandwidth>
89                </resources>
90             </flowentry>
91             <flowentry operation="create">
92                <id>24</id>
93                <priority>100</priority>
94                <port>/virtualizer/nodes/node[id=SingleBiSBiS]/ports/port[id=sap4]</
                      port>
95                <out>../../../NF_instances/node[id=2]/ports/port[id=1]</out>
96                <resources>
```

```
97                      <bandwidth>0</bandwidth>
98                  </resources>
99              </flowentry>
100         </flowtable>
101     </node>
102 </nodes>
103 </virtualizer>
```

Il grafo generato può essere rappresentato come in figura 6.1

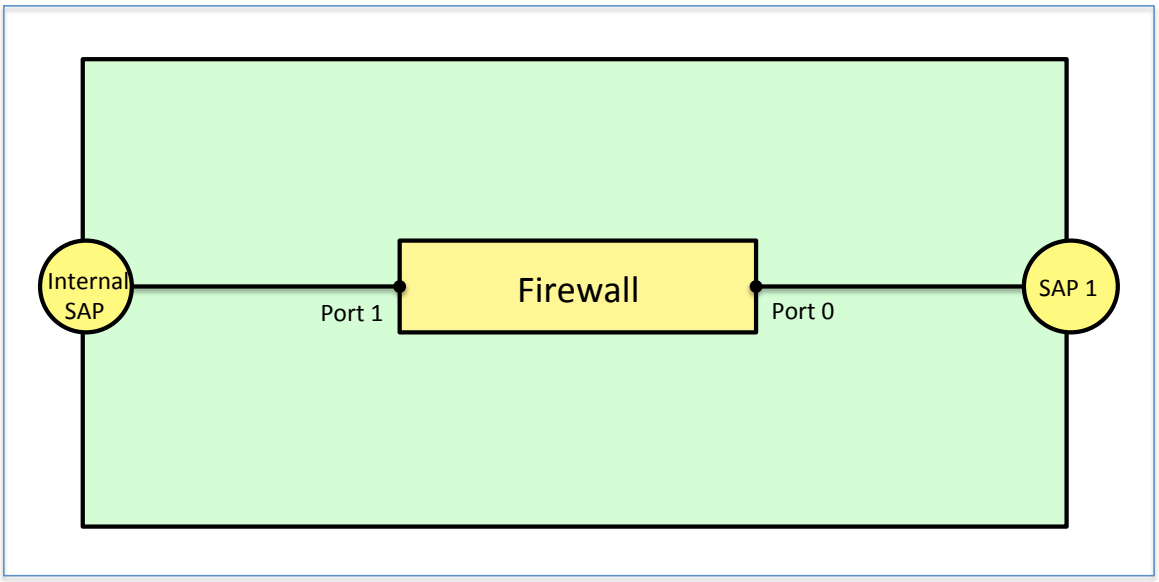

Figura 6.1. Service Graph

Ora che il grafo di servizio è stato generato, viene parsificato utilizzando le funzioni messe a disposizione dalla Virtualizer Library e vengono estrapolati gli elementi da istanziare, come le NFs e le regole. Poichè il MdO si aspetta di ricevere una nuova rappresentazione del nodo BiS-BiS, il Service Layer procede ad eseguire il parsing della configurazione ricevuta in precedenza, ne aggiunge le NFs e le regole ottenute prima dal grafo di servizio e invia la nuova configurazione al MdO.

Capitolo 7

Istanziamento di una VNF sull'infrastruttura JOLNet

In questo capitolo viene descritto l'istanziamento di una VNF partendo dal grafo di servizio, fino ad arrivare ai sottografi che raggiungono le interfacce degli orchestratori di dominio, i quali interagendo con i vari controller effettueranno realmente l'allocazione delle risorse, nel caso di OpenStack istanziando una macchina virtuale, nel caso di ONOS creando i flussi per il passaggio del traffico.

7.1 Topologia dell'ambiente di test

Per validare il lavoro svolto in durante questa tesi, è stata messa a disposizione una slice della JOLNet (per maggiori informazioni si rimanda all'appendice A). Questa slice comprende due nodi, il nodo di Trento e quello di Torino e dieci VLAN, identificate con ID che vanno dal 270 al 279. La prova consiste nel simulare un utente che si collega alla JOLNet dal nodo di Torino e deve raggiungere una risorsa (macchina virtuale) istanziata sul nodo di Trento, attraverso una *service chain*, ovvero una catena di servizi che in questo caso corrisponde ad un firewall che consente il passaggio del traffico di quell'utente soltanto verso la sua risorsa (vedi figura 7.1).

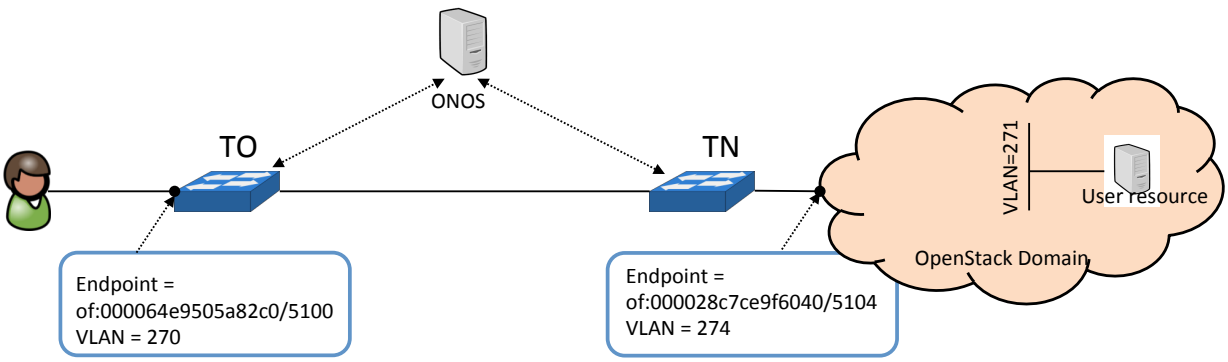

Figura 7.1. Scenario ambiente di test

Nella figura 7.1 l'utente, collegato sulla VLAN 270, deve raggiungere la sua risorsa che ha un'interfaccia sulla VLAN 271. Sono stati evidenziati gli endpoints che consentiranno il passaggio dei pacchetti dell'utente fino al raggiungimento della sua risorsa e viceversa. L'identificativo e la porta dello switch alla quale è connesso l'utente costituisce la sua posizione, che verrà inviata al Service Layer.

7.2 Elaborazione del grafo di servizio dal Service Layer al Mdo2frog4

Come descritto nel Capitolo 6, la posizione dell'utente e il MAC address del dispositivo con cui si connette alla rete, vengono inserite nel grafo di servizio e questo viene inviato verso il MdO. L'orchestratore multidominio vede i domini SDN e OpenStack come un'unico nodo e considerato che nel grafo le uniche risorse da istanziare fanno parte di quel nodo, procede a validare il grafo e lo propaga attraverso la sua interfaccia di southbound verso il Mdo2frog4. Al modulo di traduzione arriva un grafo in formato XML, come quello mostrato nel capitolo precedente nella listing 6.3, che viene tradotto in un NFFG compatibile con la NFFG Library (vedi listing 5.3). A questo punto il NFFG viene inviato al FROG4 Global Orchestrator.

7.3 Suddivisione del NFFG in sottografi da inviare agli orchestratori di dominio

Gli orchestratori di dominio possiedono dei file da configurare, che rappresentano la descrizione del dominio che controllano. In questi file devono essere elencate le varie interfacce presenti, le VLAN disponibili, il quantitativo di risorse, la tipologia di dominio, etc. Queste informazioni vengono fornite, attraverso il Double Decker, al FROG4 Global Orchestrator che le utilizza durante il processo di scheduling per creare un percorso valido che vada, ad esempio, da un endpoint ad una porta di una VNF. Nella realizzazione di questo scenario il Global Orchestrator deve mettere in comunicazione l'endpoint che rappresenta la porta dello switch SDN sul quale è connesso l'utente, con la porta della Virtual Machine che rappresenta la risorsa che l'utente vuole raggiungere. Essendo presenti e coinvolti due domini, lo scheduler del Global Orchestrator genera due sottografi, uno contenente le regole con cui i pacchetti andranno dall'utente all'endpoint del dominio Openstack, e un altro contenente l'istanza della VNF e le regole che portano il traffico dall'endpoint del dominio Openstack alla porta della VNF.

7.3.1 Sottografo del dominio SDN

Il sottografo generato per il dominio SDN presenta l'endpoint corrispondente all'ingresso del traffico dell'utente, posto sulla VLAN 270, e un secondo endpoint, generato dal FROG4 Global Orchestrator, che si posiziona nel punto di collegamento tra i domini SDN e OpenStack. Sono presenti anche regole che permettono il flusso in entrambe le direzioni del traffico tra i due endpoint. Il compito del FROG4

SDN-do consiste nel generare le corrispondenti chiamate alle REST API esposte dal controller ONOS, in modo da realizzare questo collegamento.

Come si nota dal sottografo di esempio riportato nella listing 7.1, il Global Orchestrator genera un endpoint assegnandogli una VLAN presa da una lista di VLANs disponibili fornita dalle descrizioni ricevute in precedenza. Sarà compito dell'orchestratore di dominio generare le regole che modifichino il tag VLAN, passando in questo caso dalla VLAN con ID 270 all'ID 274. Nel caso specifico della JOLNet il tag VLAN non può essere rimosso. Il Flowvisor non consente che sulla rete circolino dei pacchetti non taggati, quindi è necessario fare un *set* del VLAN ID.

Listing 7.1. Esempio di sottografo per il FROG4 SDN domain orchestrator

```
{
  "forwarding-graph": {
    "name": "NF-FG",
    "end-points": [
      {
        "name": "of:000064e9505a82c0/5100",
        "vlan": {
          "node-id": "of:000064e9505a82c0",
          "if-name": "5100",
          "vlan-id": "270"
        },
        "id": "1",
        "type": "vlan"
      },
      {
        "vlan": {
          "node-id": "of:000028c7ce9f6040",
          "if-name": "5104",
          "vlan-id": "274"
        },
        "id": "auto-generated-split_ep.1/nf.2",
        "type": "vlan"
      }
    ],
    "big-switch": {
      "flow-rules": [
        {
          "actions": [
            {
              "output_to_port": "endpoint:1"
            }
          ],
          "id": "21_2",
          "priority": 100,
          "match": {
            "port_in": "endpoint:auto-generated-split_ep.1/nf.2"
          }
        },
        {
          "actions": [
            {
              "output_to_port": "endpoint:auto-generated-split_ep.1/nf.2"
            }
          ],
          "id": "22_1",
          "priority": 100,
          "match": {
         "source_mac": "FA:16:3E:3D:F8:96",
            "port_in": "endpoint:1"
          }
        }
      ]
    }
  }
}
```

7.3.2 Sottografo del dominio OpenStack

Il sottografo generato per il dominio OpenStack presenta, come si nota dalla listing 7.2, la VNF da istanziare, con l'elenco delle porte, l'endpoint generato dal FROG4 Global Orchestrator che rappresenta il punto di ingresso del traffico sul dominio OpenStack e un secondo endpoint che permette alla VNF di raggiungere gli host collegati alla VLAN 271.

Listing 7.2. Esempio di sottografo per il FROG4 OpenStack domain orchestrator

```
{
  "forwarding-graph": {
    "name": "NF-FG",
    "VNFs": [
      {
        "name": "mca-fw1",
        "functional-capability": "mca-fw1",
        "id": "2",
        "ports": [
          {
            "id": "port:0"
          },
          {
            "id": "port:1"
          }
        ]
      }
    ],
    "big-switch": {
      "flow-rules": [
        {
          "actions": [
            {
              "output_to_port": "endpoint:auto-generated-split_nf.2/ep.1"
            }
          ],
          "id": "21_1",
          "priority": 100,
          "match": {
            "port_in": "vnf:2:port:0"
          }
        },
        {
          "actions": [
            {
              "output_to_port": "vnf:2:port:0"
            }
          ],
          "id": "22_2",
          "priority": 100,
          "match": {
            "port_in": "endpoint:auto-generated-split_nf.2/ep.1"
          }
        },
        {
          "actions": [
            {
              "output_to_port": "endpoint:4"
            }
          ],
          "id": "23",
          "priority": 100,
          "match": {
            "port_in": "vnf:2:port:1"
          }
        },
        {
          "actions": [
            {
              "output_to_port": "vnf:2:port:1"
            }
          ],
          "id": "24",
          "priority": 100,
          "match": {
            "port_in": "endpoint:4"
          }
```

```
68            }
69          ]
70        },
71        "end-points": [
72          {
73            "name": "eth1",
74            "vlan": {
75              "if-name": "eth1",
76              "vlan-id": "271"
77            },
78            "id": "4",
79            "type": "vlan"
80          },
81          {
82            "vlan": {
83              "if-name": "az_tn",
84              "vlan-id": "274"
85            },
86            "id": "auto-generated-split_nf.2/ep.1",
87            "type": "vlan"
88          }
89        ]
90      }
91    }
```

7.4 Istanziamento della VNF sul nodo di TN e creazione dei flussi su ONOS tra i nodi di TN e TO

Il risultato del corretto deployment dei due sottografi descritti in precedenza, comporta la creazione dei due flussi sulla rete SDN e l'istanziamento della VNF firewall che consente di inoltrare i pacchetti provenienti dall'utente, dall'interfaccia corrispondente alla porta 0 all'interfaccia corrispondente alla porta 1, che si affaccia sulla VLAN 271 e che può quindi raggiungere la risorsa dell'utente.

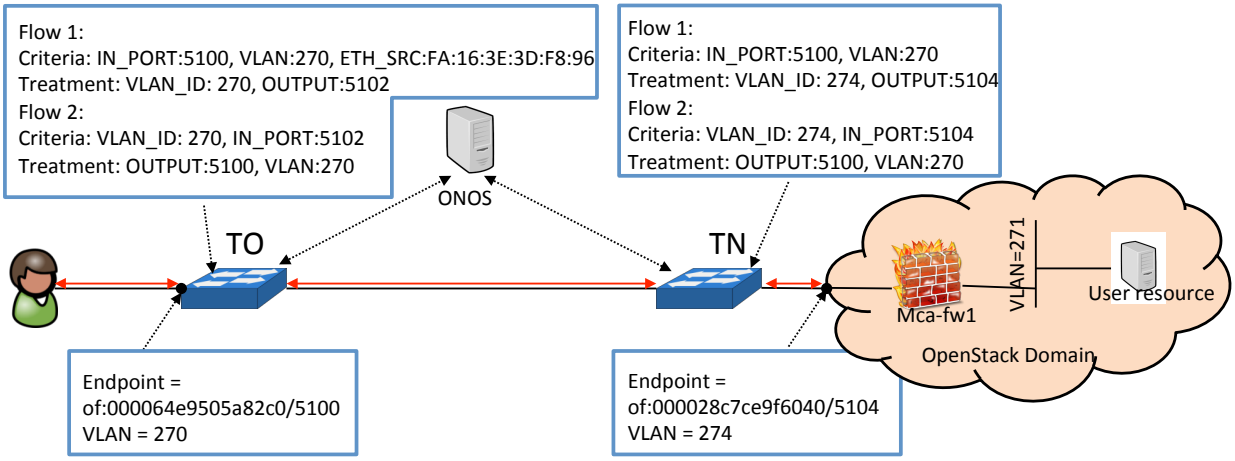

Figura 7.2. Scenario ambiente di test dopo che il grafo è stato istanziato

Come si nota dalla figura 7.2 il controller ONOS provvede ad istanziare i flussi sugli switch SDN. Sullo switch appartenente al nodo di TO vengono creati due flussi:

- il primo flusso prende i pacchetti provenienti dalla porta 5100 e appartenenti alla VLAN 270 e il inoltra sulla porta 5102 dopo aver verificato la corrispondenza con il MAC address;
- il secondo flusso esegue il percorso inverso inoltrando i pacchetti provenienti dalla porta 5102 verso la porta 5100.

Sullo switch appartenente al nodo di TN invece vengono create i flussi seguenti:

- il primo flusso prende i pacchetti provenienti dalla porta 5100 e appartenenti alla VLAN 270 e il inoltra sulla porta 5104 dopo aver settato il tag VLAN al valore 274;
- il secondo flusso esegue il percorso inverso inoltrando i pacchetti provenienti dalla porta 5104 verso la porta 5100 dopo aver impostato il tag VLAN al valore 270.

A questo punto l'utente potrà raggiungere la sua risorsa. I flussi generati permetteranno in realtà all'utente di raggiungere qualsiasi virtual machine presente sul nodo di Trento e con una interfaccia sulla VLAN 274. Considerando però solo le istanze create dal Global Orchestrator, quest'ultimo non utilizzerà più quella VLAN che d'ora in poi verrà segnata come occupata, dunque l'utente potrà raggiungere solamente la sua VNF.

Capitolo 8

Validazione

In questa sezione si analizzeranno le tempistiche e i ritardi introdotti dai moduli posti al di sopra del FROG4 Global Orchestrator rispetto al deployment di un NFFG inviato direttamente all'orchestratore sopracitato.

8.1 Tempi di istanziamento di un grafo di servizio

Per eseguire i test sulle tempistiche si è fatto riferimento al processo di istanza di un grafo presentato nel capitolo precedente. In particolare sono stati calcolati i tempi necessari al Service Layer, al MdO e al Mdo2frog4 per eseguire correttamente le loro funzioni. I tempi sono stati presi singolarmente per ogni modulo, dal momento in cui riceve un messaggio sulla sua interfaccia, al momento in cui termina l'elaborazione e lo propaga.

8.1.1 Service Layer

	test 1	test 2	Media
Tempi (s)	0,386	0,314	0,350

8.1.2 Multi Domain Orchestrator

	test 1	test 2	Media
Tempi (s)	0,320	0,278	0,299

8.1.3 Mdo2frog4

	test 1	test 2	Media
Tempi (s)	0,258	0,247	0,253

8.1.4 Ritardo medio introdotto dai 3 moduli

	Service Layer	MdO	Mdo2frog4	Tempo complessivo
Media dei tempi(s)	0,350	0,299	0,253	0,902

8.2 Valutazioni

Dall'analisi effettuata si evince come venga introdotto un ritardo di circa 1 secondo rispetto al tempo totale necessario ad istanziare un grafo che in media è di circa 10 secondi. Il ritardo aggiunto è coerente con i tempi di elaborazione necessari ad effettuare le varie operazioni. È importante specificare anche che il MdO in questo caso non ha dovuto contattare altri MdO per allocare risorse in altri domini amministrativi. In questo nuovo scenario i tempi di elaborazione del MdO potrebbero allungarsi ulteriormente ed introdurre un ritardo maggiore rispetto a quanto calcolato. Si ricorda infine che la versione utilizzata del Multi Domain Orchestrator è la prima rilasciata, dunque è probabile che con le successive versioni i tempi, soprattutto nello scenario multi dominio amministrativo, non aumentino in modo eccessivo.

Capitolo 9

Conclusioni

Nel lavoro svolto in questa tesi è stato raggiunto l'obiettivo di interfacciare il Multi Domain Orchestrator con l'infrastruttura geografica JOLNet, utilizzando gli orchestratori sviluppati dal NetGroup del Politecnico di Torino e sviluppando il modulo che gestisce la traduzione tra i due formalismi utilizzati dagli orchestratori. I test effettuati e le analisi sulle tempistiche hanno dimostrato come la scelta utilizzare degli orchestratori già compatibili con la JOLNet e di introdurre un modulo che esegue la traduzione sia stata una buona soluzione per raggiungere l'obiettivo della tesi.

9.1 Sviluppi futuri

Il lavoro svolto in questa tesi è stato una prima prova per interfacciare il Multi Domain Orchestrator con la JOLNet, proponendo un caso d'uso su un unico dominio amministrativo. Con le successive release del MdO ci si aspetta che le potenzialità di questo software vengano espanse e che la possibilità di creare dei collegamenti tra più domini amministrativi venga implementata. Di conseguenza ci si aspetta che venga continuato lo sviluppo del modulo di traduzione per supportare sia nuove funzionalità che verranno introdotte dal MdO sia per continuare a seguire l'evoluzione dei FROG orchestrators.

Appendice A

Joint Open Lab Network

Questa sezione fornisce una spiegazione più dettagliata riguardo al network sul quale sono stati effettuati i test durante questo lavoro di tesi.

A.1 Il progetto JOLNet

TIM in collaborazione con le maggiori università italiane ha sviluppato una rete SDN basata su switch Cisco che supportano il protocollo OpenFlow, con lo scopo di poter studiare le nuove funzionalità introdotte dai paradigmi SDN e NFV. La JOLNet consente quindi di fare ricerca su argomenti quali, ad esempio, il traffic engineering, il cloud computing e le network function, dando la possibilità di eseguire anche delle valutazioni sugli effettivi vantaggi e sui problemi di queste tecnologie. La JOLNet da un valore aggiunto alla validazioni dei vari progetti di ricerca poichè queste sono fatte non in un ambiente virtuale, ma su un'infrastruttura reale, che si avvicina molto ad una rete di produzione geografica di un Internet Service Provider.

A.2 Topologia della rete

La JOLNet è una rete geografica che si estende lungo tutta la penisola italiana ed è composta da 7 nodi. Ogni nodo è a sua volta composto da 1 server e 2 switch SDN, come si può vedere dalla figura A.1. I due switch si differenziano in quanto uno è identificato come switch di core ed è collegato agli altri switch di core della rete in maglia completa, mentre l'altro è uno switch che va semplicemente ad estendere la rete all'interno di ogni nodo. Gli switch di core costituiscono il backbone della rete e su questi sono collegati i 7 server che ospitano il framework OpenStack. Nel dettaglio i vari nodi non sono collegati direttamente tra loro, ma i link consistono di tunnel creati sulla rete TIM esistente.

La rete fisica è divisa logicamente in più porzioni, in modo da garantire più ambienti di test e sviluppo separati tra loro, nonostante vengano utilizzati gli stessi nodi. Questa suddivisione è resa possibile dal Flowvisor, un network hypervisor che si posiziona tra lo switch SDN e il controller OpenFlow. Il Flowvisor discrimina il traffico sugli switch in base al VLAN ID dei pacchetti indirizzandoli ai vari controller. In questo modo un controller potrà agire sul traffico dei pacchetti che provengono dalle VLAN appartenenti alla partizione di rete che può controllare.

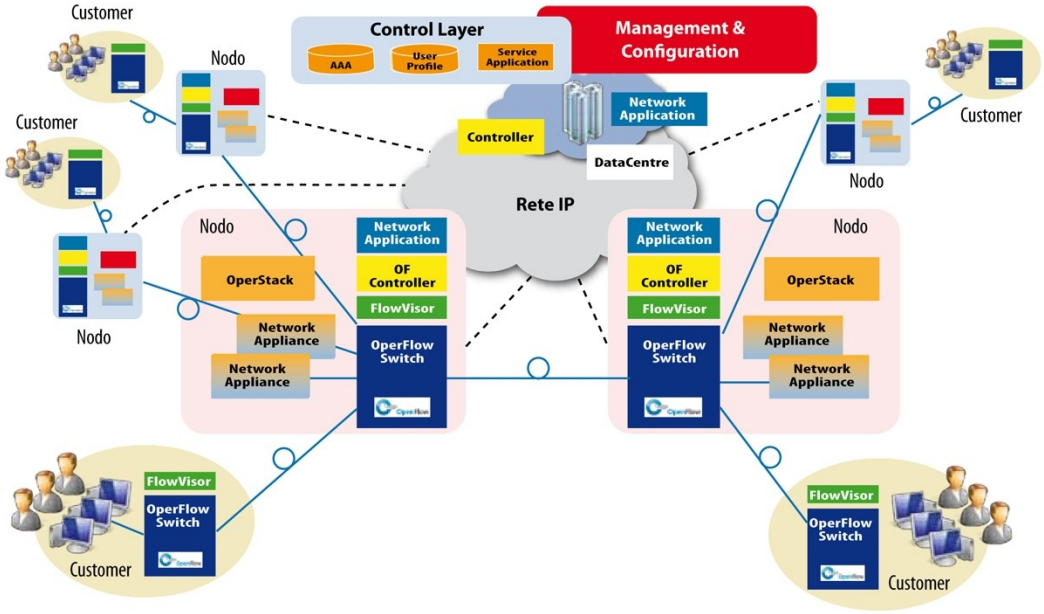

Figura A.1. Nodo JOLNet. Immagine tratta da www.telecomitalia.com/tit/it/notiziariotecnico/numeri/2014-2/capitolo-05

A.3 Collegamento tra network Polito e JOLNet

Dalla rete interna del Politecnico di Torino è possibile accedere direttamente alle macchine della rete JOLNet. È presente infatti un tunnel IPsec che collega l'access point del Politecnico con quello dei laboratori TiLab (vedi figura A.2). Il gateway del Politecnico riconosce i pacchetti indirizzati verso la JOLNet e ne forza il passaggio attraverso il tunnel IPsec, consentendogli di raggiungere la rete di TiLab.

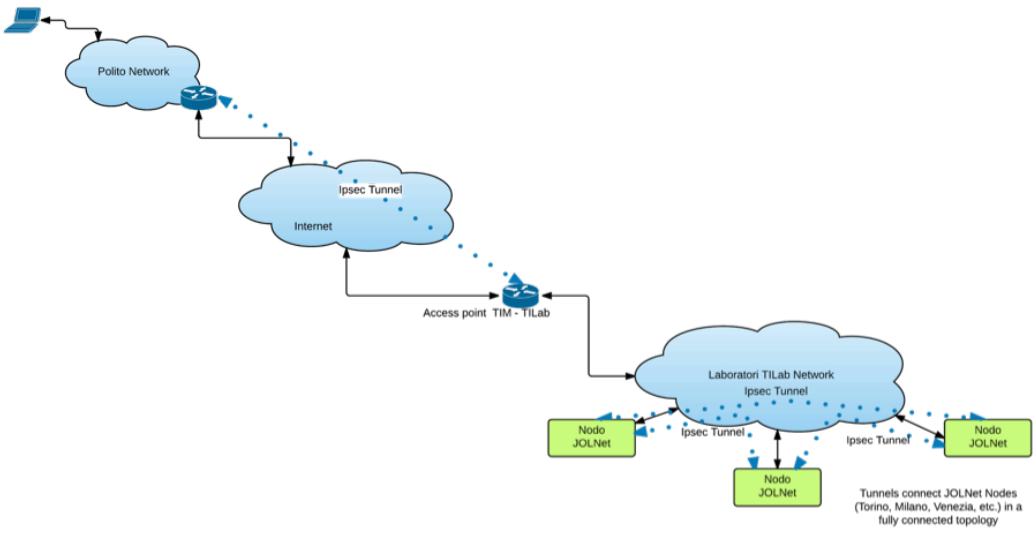

Figura A.2. Collegamento tra Politecnico di Torino e TiLab

Bibliografia

[1] FROG4. url: https://github.com/netgroup-polito/frog4
[2] Open vSwitch. url: http://vswitch.org.
[3] DoubleDecker. url: https://github.com/Acreo/DoubleDecker.
[4] UNIFY website. url: https://www.fp7-unify.eu.
[5] Balàzs Sonkoly et al. "Multi-Domain Service Orchestration Over Networks and Clouds: A Unified Approach". In: SIGCOMM. Aug. 2015.
[6] OpenStack official website. url: http://openstack.org/.
[7] Ivano Cerrato et al. "Towards Dynamic Virtualized Network Services in Telecom Operator Networks". In: Computer Networks 92 (2015), pp. 380–395.
[8] ONOS official website. url: https://onosproject.org/
[9] ESCAPE url: https://sb.tmit.bme.hu/mediawiki/index.php/ESCAPE
[10] 5GEX project url: http://www.5gex.eu/