POLITECNICO DI TORINO

Corso di laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Creazione di servizi virtualizzati in un home-gateway

Relatore
prof. Fulvio Risso

Tutore:
dott. Ivano Cerrato

Candidato
Federica Li Muti

Dicembre 2017

Alla mia famiglia.

Indice

Elenco delle figure

Capitolo 1

Introduzione

La nascita delle prime reti si vede negli anni '70 quando ancora consistevano in apparati non dotati di intelligenza e connessi a dei grossi e potenti mainframe. Il primo progetto di una rete a livello mondiale prevedeva il collegamento di computer di 4 Università americane. Solo dagli anni '80, dopo la definizione del protocollo *TCP/IP* e la nascita dei Personal Computer, si diede il via ad una crescita che portò alla connessione di miliardi di utenti da tutto il mondo così da costituire una vasta infrastruttura sotto la gestione di vari *Internet Service Provider (ISP)*.

Tuttavia, la rete era di tipo statico, ovvero, gli apparati di rete necessitavano una configurazione individuale, c'era una dipendenza legata alle richieste dei vendor per il supporto a nuove capabilities e servizi e molti protocolli che miravano a risolvere solo un problema specifico (QoS, VLAN, ...).

Negli anni '90 vengono sviluppate le *Active Networking*, che rendono la rete più dinamica in quanto si tratta di particolari reti i cui nodi sono programmabili. Ogni nodo, in base al flusso di pacchetti che arriva in ingresso, può essere programmato dinamicamente. Tramite hardware attivo è in grado di far routing, switching o eseguire frammenti di codice: un utente di un'Active Networking può, ad esempio, inviare al nodo dei programmi che mirano a modificare il suo comportamento all'arrivo dei pacchetti in ingresso.

Ad oggi le Active Networks si sono evolute in *Software Defined Networking (SDN)*, le quali disaccoppiano il piano dati dal piano di controllo dei nodi di rete, il quale diventa centralizzato. Nascono inoltre anche i primi *Network Operating*

System che consentono la creazione di applicazioni di rete condivise dagli utenti.

La nuova tecnologia, insieme a *Network Functions Virtualization (NFV)* che aggiunge aspetti di virtualizzazione, favorisce lo sviluppo di servizi personalizzabili della rete, risponde ai cambiamenti dinamici dei pattern di traffico, supporta servizi di cloud privati e pubblici e aiuta ad avere una gestione intelligente e complessiva della rete. In questo modo la rete diventa quindi una vera e propria piattaforma di processamento.

Oggi queste tecnologie si stanno diffondendo molto in ambito data center e nel *core* della rete dell'operatore. Sulla parte periferica della una rete, il cosiddetto *network edge*, invece non ci si è mossi molto in questa direzione, in quanto le sue funzionalità sono state orientate principalmente al controllo d'accesso e alla connettività.

I nodi all'edge della rete negli ultimi anni non hanno visto una crescita significativa, al contrario dei dispositivi utente che invece sono diventati dei sistemi di tipo plug & play e altamente personalizzabili.

Tuttavia, con il passare degli anni, si sta cercando di rendere flessibile e intelligente anche l'edge della rete. In particolare, un utente potrebbe voler personalizzare il proprio home-gateway con dei particolari servizi, così come è in grado di personalizzare il proprio cellulare con le applicazioni che più preferisce.

Oggi, soprattutto in ambito *Internet of Things (IoT)*, la ricerca sta cercando di interconnettere i dispositivi all'edge della rete che stanno a diretto contatto con l'utente finale ai servizi cloud della rete, rendendoli parte integrante del cloud stesso.

Tra i dispositivi interessati ci sono i CPE, come i comuni home-gateway, nei quali virtualizzazione e integrazione con il cloud semplifica e accelera l'installazione di servizi, permettendone e facilitandone anche la modifica e la creazione di nuovi da parte dell'utente finale.

Sebbene una funzione di rete virtuale (*Virtual Network Function - VNF*) può essere ospitata ovunque si abbiano server con risorse disponibili, ha senso ospitare alcune di esse all'edge della rete e quindi sul CPE del cliente. Ci sono diverse ragioni che motivano ciò [1]:

- praticità: sicuramente certe funzioni di rete devono essere ospitate presso il CPE per ragioni pragmatiche. Ad esempio il tema della sicurezza è di certo

caro ad un'azienda e quindi preferirebbe che i suoi dati vengano crittografati prima che escano dall'edificio; dunque in questo caso sarebbe necessaria una Virtual Network Function (VNF) che ha il compito di cifrare/decifrare i dati all'interno del CPE del cliente.

- resilienza: alcune funzioni di rete, come ad esempio *Private Branch Exchange (PBX)* per la telefonia IP, devono essere disponibili anche quando viene a mancare la connettività verso la rete WAN. Se questa funzione di rete è ospitata in un lontano data center e la connessione di rete viene a mancare, l'impresa potrebbe non essere in grado di effettuare chiamate locali, o anche tra stanze dello stesso ufficio;

- performance: certe funzioni di rete funzionano meglio se ospitate all'edge della rete. Esempi sono la gestione della qualità del servizio (QoS) end-to-end e il controllo del rispetto del *Service Level Agreement (SLA)*.

In questa tesi si sfrutta il software open-source dello universal-node, in grado di concretizzare servizi astratti in funzioni e percorsi di rete, grazie al supporto di diversi motori di virtualizzazione. In particolare, sfruttando le tecnologie SDN e NFV, si vuole realizzare quella che è la funzionalità di un home-gateway tramite l'utilizzo di grafi di servizio, permettendo una configurazione ed installazione dinamica di servizi e regole di traffico all'interno del CPE stesso. L'utilizzo di grafi permette di implementare le funzionalità che si vogliono garantendo alta flessibilità e permettendo anche future modifiche migliorative ai servizi così da rendere tutto trasparente all'utente.

In particolare, il lavoro è stato concentrato sul rendere lo universal-node un prototipo utilizzabile in casa dagli utenti e in modo da poter deployare grafi che abbiano una terminazione verso un tipo di connessione PPPoE utilizzata dalla grande maggioranza degli operatori di rete per il collegamento di utenti xDSL verso la centrale.

L'elaborato è organizzato come segue:

- **Capitolo 2:** descrive lo stato dell'arte dell'elemento in questione ed evidenzia le differenze con i progetti simili;

- **Capitolo 3:** fornisce una panoramica delle architetture e dei componenti che costituiscono lo universal-node;
- **Capitolo 4:** descrive in dettaglio lo universal-node, l'architettura e i moduli che lo compongono;
- **Capitolo 5:** espone le modifiche apportate allo universal-node grazie a questa tesi, le quali hanno permesso di raggiungere l'obiettivo che ci si era prefissati;
- **Capitolo 6:** presenta il caso d'uso in questione, mostrando i grafi che compongono il servizio di un home-gateway;
- **Capitolo 7:** esamina le prestazioni del caso d'uso in diversi scenari di utilizzo;
- **Capitolo 8:** espone le conclusioni e presenta alcuni possibili sviluppi futuri sulla base del lavoro svolto in questa tesi.

  Appendice A: descrive una prima soluzione che era stata presa in considerazione per raggiungere gli obiettivi prefissati e perché si è optato per un'altra soluzione.

Capitolo 2

Stato dell'Arte

La necessità di avere una maggiore flessibilità nei CPE che vengono utilizzati in casa e in ufficio dai clienti, è cresciuta nel corso degli anni e si è mostrata sempre più evidente con l'emergere del paradigma NFV. Di recente si sente infatti parlare di CPE virtuali (vCPE) che consistono nel posizionare gran parte (o tutte) le funzioni di un CPE nei data center; degli esempi sono [2] e [3]. All'edge della rete rimane un dispositivo dotato di un hardware minimo, mentre l'intelligenza viene migrata sul cloud e implementata tramite funzioni virtuali.

Un passo mosso verso un CPE totalmente virtualizzato è proposto in [4], che si basa sull'architettura definita dalla *Home Gateway Initiative industry alliance 1.* Si tratta di un'architettura modulare che implementa diverse funzioni del CPE come *Java OSGi bundles*, che possono essere caricati/scartati dinamicamente su richiesta. Le VNF surrogate estendono questo paradigma tramite la definizione di una serie di funzioni OSGi che hanno il ruolo di "proxy" e mantengono la compatibilità con l'architettura esistente, mentre delegano la maggior parte del processamento ad una VNF associata in esecuzione nel cloud.

Tuttavia, tali soluzioni hanno bisogno di una connettività eccellente tra la sede del cliente e il data center e potrebbero anche introdurre eccessiva latenza su servizi sensibili. Inoltre anche se in base al principio NFV un operatore dovrebbe essere in grado di orchestrare i propri servizi sfruttando le risorse offerte dall'intera infrastruttura di rete, l'approccio vCPE non riesce a sfruttare le risorse disponibili su nodo in quanto le VNF vengono spostate sul cloud del data center.

Servizi basati sul lato edge della rete sono proposti in [5], che sfrutta programmi eBPF per creare *data path* programmabili nel CPE, mantenendo invece il piano di controllo sul cloud. Il CPE riesce a gestire il traffico localmente, garantendo in questo modo operatività anche se la connessione verso il cloud viene persa. Questa soluzione è molto efficiente, tuttavia la macchina virtuale eBPF non è Turing equivalente e non supporta alcuni semplici programmi che sono comuni all'edge della rete (come ad esempio il match di una stringa).

Poiché un CPE ha di solito un numero limitato di risorse, un modello di ottimizzazione in grado di selezionare la VNF migliore, ottimizzando così il costo delle VNF istanziate sul CPE, è quello proposto da [6]. Questa soluzione si basa però su tecnologie esistenti per l'implementazione di VNF, come container Linux o macchine virtuali, mentre il CPE in questione è in grado anche di far girare delle VNF come funzioni native, riducendo così l'overhead del sistema.

Questo lavoro parte da un'architettura esistente che vuole proporsi in questo scenario come un prototipo di home-gateway basato su servizi virtualizzati grazie all'utilizzo di grafi che descrivono un insieme di funzioni di rete e di punti di ingresso/uscita del traffico, specificando quali sono i percorsi e le regole che interconnettono gli uni con gli altri. Tuttavia, per poter essere portato a casa dell'utente, deve essere necessario che l'architettura sia compatibile con le tecnologie largamente utilizzate dai fornitori di servizi per il collegamento degli utenti alla centrale.

Capitolo 3

Background

L'architettura di un sistema di telecomunicazioni è composto da tre componenti fondamentali: piani di controllo, piani dati e piani di gestione. Quest'ultimo si occupa del trasporto del traffico amministrativo ed è considerato un sottoinsieme del piano di controllo; entrambi servono il piano dati che smista il traffico che la rete deve trasportare. Nelle reti convenzionali, questi tre piani sono implementati nel firmware di router e switch, invece nella tecnologia "Software Defined Networking (SDN)", il piano di controllo viene rimosso dall'hardware e implementato tramite software, così da fornire più dinamicità e flessibilità per l'amministrazione della rete. L'amministratore di rete è in grado di modellare i flussi del traffico grazie ad una console di controllo centralizzata, senza dover agire individualmente ogni switch. In questo modo è in grado di modificare le regole già presenti sugli apparati con una granuralità molto più ad alto livello, assegnando ad esempio una specifica priorità o bloccando un determinato tipo di pacchetti.

La nascita di "Network Functions Virtualization (NFV)" è legata al sempre più crescente costo degli apparati di rete e alla mancata possibilità di assegnare risorse di computing in modo intelligente (in base all'effettivo utilizzo). Questa nuova tecnologia prevede la trasformazione delle funzioni di rete in software in modo tale che possano girare su macchine virtuali, così da minimizzare e consolidare l'hardware e distribuire le risorse in maniera ottimale.

3.1 Software Defined Networking

L'architettura Software Defined Networking (SDN) prevede un disaccoppiamento tra piano di controllo e piano dati, rendendo il primo direttamente programmabile. L'infrastruttura di rete diventa esclusivamente hardware e inoltra pacchetti in base a delle regole calate tramite una funzione logica centralizzata che rappresenta la piattaforma programmabile e in grado di eseguire le applicazioni che implementano il piano di controllo. In questo modo, hardware, piano di controllo e applicazioni risultano separate e indipendenti tra di loro, inoltre le comunicazioni tra i diversi livelli avvengono tramite delle interfacce ben definite. Le risorse di rete diventano astratte grazie ad una logica programmabile che può essere fornita tramite servizi di rete personalizzabili. L'architettura SDN riesce quindi a fornire programmabilità, automazione e controllo alla rete.

Per poter implementare la tecnologia SDN esistono diversi metodi che mettono in comunicazione il piano di controllo e il piano dati. Tra questi, l'organizzazione no-profit *Open Networking Foundation (ONF)* che promuove SDN e ha standardizzato elementi come il protocollo *OpenFlow*, il quale definisce come il piano di dati e il piano di controllo devono comunicare. OpenFlow tramite il concetto di flusso identifica il traffico di rete in base a delle regole predefinite di *match* programmabili sia staticamente che dinamicamente grazie al controllo software SDN.

3.2 Network Functions Virtualization

Al giorno d'oggi, si ha la necessità di manipolare le reti e processare i pacchetti per diverse ragioni. Nelle reti tradizionali, questo viene fatto aggiungendo alla rete degli apparati autonomi che svolgono degli specifici compiti (come firewall, web cache, NAT, network monitor). L'interconnessione di questi box costituiscono la cosiddetta *chain of service*; ogni cambiamento i questa catena porta spesso a delle difficoltà in quanto bisogna disconnettere e riconnettere fisicamente i box e apportare a volte anche delle modifiche nella configurazione degli apparati gestiti.

Nel Novembre del 2012, per affrontare questi e altri problemi, sotto l'organizzazione "European Telecommunications Standard Institute (ETSI)", è stata formata

una nuova *Industry Specification Groups (ISG)* chiamata "Network Functions Virtualization (NFV)" e guidata da sette operatori di rete leader a livello mondiale. È stata proposta una soluzione che sfrutta l'evoluzione della virtualizzazione IT e mira a consolidare molti tipi di apparecchiature di rete in un'industria standard di server, switch e sistemi di storage.

Questa soluzione prevede che le funzioni di rete vengano implementate in software che sia in grado di girare sull'hardware di server di alta e media fascia e che sia possibile muoverle o istanziarle in qualsiasi parte della rete in base alle necessità. In questo modo, l'operatore ha la possibilità di costruire la propria rete appoggiandosi su un hardware generica di alta qualità al quale può aggiungere servizi flessibili in maniera dinamica e con un ridotto costo. Il sistema risulta in questo modo molto flessibile e favorisce una semplice condivisione delle risorse ed un'efficiente scalabilità. Pochi server fisici possono quindi contenere diverse funzioni, diminuendo la potenza dissipata ed ottimizzando la distribuzione delle risorse.

3.3 Piano di controllo

Il piano di controllo è quella parte della rete che si occupa del trasporto del traffico di segnalazione e del *routing*, ovvero di quelle funzioni necessarie per la configurazione e la gestione degli apparati di rete.

3.3.1 Openflow

OpenFlow (OF) [7] è un protocollo di comunicazione sviluppato originariamente dai ricercatori della "Stanford University" per poter collaudare, in maniera semplice, i protocolli sperimentali sulla rete del campus utilizzando vari dispositivi di rete (come switch, router e punti di accesso). Tale protocollo permette a dei *controller* remoti di determinare il percorso dei pacchetti di rete tramite la gestione degli switch OF-compatibili.

La separazione della parte di controllo da quella di *forwarding* permette di gestire il traffico in maniera più sofisticata e allo stesso tempo semplice rispetto alle apparecchiature preesistenti. OpenFlow cerca anche di unificare la gestione

da remoto di switch eterogenei provenienti da diversi vendor grazie ad un unico protocollo aperto.

In OpenFlow (versione 1.0) veniva utilizzata una tabella dei flussi popolata da regole *Access Control List (ACL)* estese. In questo modo si avevano delle limitazioni, infatti se ad esempio si volevano configurare regole L2 e L3 che non fossero sovrapposte, il risultato erano un elevato numero di regole ottenute dal prodotto cartesiano delle regole iniziali.

Questo limite è stato superato nelle versioni successive in modo da permettere ad uno switch OF di utilizzare fino a 255 tabelle, accessibili in cascata, tra flussi, gruppi (azioni comuni per molti flussi, o di bilanciamento del carico o multicast) e metrica (traffico di *policing*).

Nonostante la versatilità, questa modifica ha portato con se vari problemi ai fornitori di switch e il software risente ancora dei limiti hardware come ad esempio il limite del numero di tabelle (255), l'ordine fisso di tabelle e gruppi (non è possibile saltare da un gruppo ad una tabella) e la pipeline *loop-free* (non è consentito tornare indietro ad una tabella che è già stata visitata).

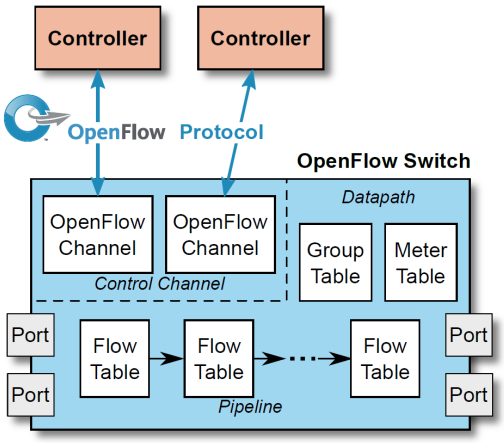

Figura 3.1. Architettura switch OpenFlow

Come si vede dalla figura 3.1, uno switch logico OpenFlow è costituito da una o più tabelle di flusso, una di gruppo e una di metrica, che si occupano del *lookup* e

del *forwarding* dei pacchetti, e uno o più canali OpenFlow verso i controller esterni.

Grazie al protocollo OpenFlow, il controller è in grado di aggiungere, aggiornare e rimuovere i flussi nelle tabelle, in modo reattivo, ovvero in risposta all'arrivo di un pacchetto, o in modo proattivo. All'interno di ogni tabella sono contenute delle regole, ognuna delle quali è formata da uno o più campi di *match*, un contatore e un insieme di istruzioni da applicare ai pacchetti che soddisfano il match. Nel caso in cui non venga trovato alcun match in tabella, il pacchetto verrà inviato al controller OpenFlow, cancellato o consultata la tabella successiva. Ad ogni flusso sono associate delle istruzioni che possono contenere azioni, come operazioni sul pacchetto, o direttive per la modifica del processo di pipeline in modo che il pacchetto possa raggiungere altre tabelle per altri eventuali processamenti.

Il controller OpenFlow interagisce con uno o più switch tramite il protocollo OpenFlow. Tra il controller e l'interfaccia di ogni singolo switch è presente un canale OpenFlow utilizzato per la comunicazione dei due.

Uno dei progetti più interessanti tra le implementazioni di tale protocollo è OpenvSwitch (OVS) [8]. Si tratta di un software switch multilayer in grado di supportare le interfacce standard di gestione e che estende le funzioni di forwarding ad un accesso e un controllo programmatico. È adatto a funzionare come switch virtuale in ambienti VM e oltre ad esporre le interfacce standard di controllo e di visibilità allo strato di rete virtuale, è stato progettato per supportare la distribuzione su più server fisici.

3.4 Piano dati

Il livello che si occupa di smistare il traffico utente è il piano dati (noto anche come piano utente, piano di forwarding o piano di trasporto).

3.4.1 OpenvSwitch

OpenvSwitch [8] è uno switch virtuale *multilayer* rilasciato sotto licenza del progetto open-source Apache 2.0.

È stato progettato per dotare le reti di una massiccia automazione estendendo il livello di programmazione, pur sostenendo le interfacce e i protocolli standard

di gestione (ad esempio NetFlow, sFlow, IPFIX, RSPAN, CLI, LACP, 802.1ag). Esso supporta anche la distribuzione su più server fisici come *vNetwork distributed vswitch* di VMware o *Nexus 1000V* di Cisco.

Poiché gli *Hypervisor* devono riuscire a gestire traffico tra macchine virtuali e con il mondo esterno, nel caso in cui siano basati su Linux, questo si traduce nell'utilizzo dello switch L2 nativo del sistema (Linux Bridge), che risulta essere veloce e affidabile. Il motivo per cui viene però risulta vantaggioso utilizzare Open vSwitch sta nel fatto che questo è destinato a implementazioni multi-server di virtualizzazione, un ambiente nel quale lo stack precedente risulta non adatto, a causa della gestione problematica dell'elevata dinamicità degli end-point.

I punti di forza di Open vSwitch sono messi in risalto dalle seguenti caratteristiche e considerazioni di progettazione:

- Mobilità dello Stato: lo stato della rete associato ad una entità (ad esempio una macchina virtuale) deve essere facilmente identificabile e migrabile tra diversi *host*. Con 'stato della rete' si intende il tradizionale *soft state* (come ad esempio una *entry* in una tabella di *learning* di un L2 switch), lo stato di farwarding di uno switch L3, la politica di routing, ACL, QoS *policy*, il monitoraggio della configurazione (ad esempio NetFlow, IPFIX, sFlow), etc.

  Open vSwitch supporta sia la configurazione che la migrazione delle istanze. Per esempio, se una VM migra tra due end-host, è possibile sia la migrazione della configurazione associata (regole SPAN, ACL, QoS), sia quella di qualsiasi stato della rete (come ad esempio lo stato attuale che potrebbe essere una caratteristica difficile da ricostruire). Inoltre lo stato di Open vSwitch è sostenuto da un modello dati che consento lo sviluppo di sistemi di automazione strutturati.

- Risposta alle dinamiche della rete: alti tassi di cambiamento caratterizzano spesso gli ambienti virtuali. VM che vanno e vengono, macchine virtuali che vanno avanti e indietro nel tempo, modifiche alla rete logica, e così via.

  Open vSwitch supporta un insieme di funzioni tramite le quali un sistema di controllo di rete riesce a rispondere e ad adattarsi ai cambiamenti dell'ambiente. Esso si appoggia ad un database dello stato della rete (OVSDB) che

supporta *trigger* remoti; in questo modo un software di orchestrazione riesce ad vedere i vari aspetti della rete e agire quando si verifica un cambiamento. Ad esempio, questo è molto sfruttato oggi per poter rispondere e tener traccia delle migrazioni delle macchine virtuali.

- Gestione dei *tag* logici: switch virtuali distribuiti (come VMware vDS e Nexus 1000V di Cisco), spesso hanno un contesto logico all'interno della rete grazie all'aggiunta o alla manipolazione dei tag nei pacchetti di rete. In questo modo si riesce per esempio ad identificare una VM in modo univoco (in un modo resistente allo *spoofing* hardware). Il problema più grosso che si ha nella costruzione di uno switch virtuale distribuito è quello che riguarda la gestione in modo efficiente e corretto di questi tag.

  Open vSwitch ha diversi metodi che permettono di specificare e gestire le regole che coinvolgono i tag ed è possibile accedere a tutti tramite un orchestratore remoto. Spesso queste *tagging rules* sono memorizzate in una forma ottimizzata; in queso modo non devono essere accoppiate a nessun dispositivo di rete e permettono, ad esempio, di configurare, cambiare e migrare migliaia di tagging rules.

  Allo stesso modo, Open vSwitch supporta un'implementazione GRE che riesce a gestire migliaia di tunnel allo stesso tempo e supporta anche la configurazione remota per la creazione di questi tunnel, permettendo in questo modo per esempio il collegamento tra reti private di VM in diversi data center.

- Integrazione Hardware: il *forwarding path* di Open vSwitch (il *datapath* nel kernel) è stato progettato in modo che risultasse compatibile con l'*offloading* di elaborazione dei pacchetti del chipset hardware, nel caso in cui venga ospitato da un classico switch hardware o all'interno di un end-host NIC. In questo modo il *control path* di Open vSwitch riesce a controllare sia un'implementazione puramente software, sia uno switch hardware.

  Molte sono le iniziative che mirano a portare Open vSwitch all'interno di chipset hardware. Tra i promotori ci sono diversi commercianti di chipset in silicio (Broadcom e Marvell), così come un certo numero di fornitori di specifiche piattaforme.

Il vantaggio di avere hardware integrato non sta tutto nel guadagno di performance all'interno degli ambienti virtualizzati. Se gli switch fisici espongono anche le astrazioni del controllo di Open vSwitch, gli ambienti di hosting virtuali possono essere gestiti utilizzando lo stesso meccanismo per il controllo della rete automatizzata.

Open vSwitch è stato progettato sviluppando aspetti diversi rispetto i precedenti *hypervisor networking stack*; questo perché si aveva la necessità di ottenere un controllo automatizzato e dinamico dalla rete in ambienti di virtualizzazione su larga scala basati su Linux.

Open vSwitch si pone come obiettivo quello di mantenere codice in-kernel il più piccolo possibile (importante per le prestazioni) e di riutilizzare quando possibile i sottosistemi già esistenti (per esempio OVS usa il QoS di stack esistenti). Dalla versione 3.3 di Linux, Open vSwitch è incluso come parte integrante del kernel e i pacchetti per le *userspace utilities* sono disponibili sulle distribuzioni più popolari.

3.5 Virtualizzazione

La virtualizzazione è il processo di creazione di una rappresentazione virtuale (basata sul software) e non fisica di qualcosa. È possibile virtualizzare applicazioni, server, storage e reti, riducendo completamente e in maniera efficace le spese IT e aumentando al tempo stesso l'efficienza e l'agilità del sistema.

3.5.1 Docker

Docker [9] è una piattaforma per lo sviluppo, la distribuzione e l'esecuzione di applicazioni. Tale piattaforma da la possibilità di separare le applicazioni dell'infrastruttura permettendo in questo modo una gestione più semplice. Sfruttando Docker per la distribuzione, il trasporto ed il test del codice in modo rapido, si riesce a ridurre significativamente il ritardo tra la scrittura del codice stesso e la messa in produzione.

Le applicazioni girano all'interno di un ambiente isolato che viene chiamato container: grazie all'isolamento e alla sicurezza è possibile eseguire più container

contemporaneamente sullo stesso host e in numero superiore rispetto alle virtual machines grazie alla natura leggera di un container (in quanto girano senza il carico extra di un hypervisor).

Docker mette a disposizione dei *tool* ed una piattaforma grazie ai quali è possibile gestire il 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 prende il nome di "Docker Engine" (figura 3.2) ed 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 può utilizzare la CLI per interagire con il client docker che ha il compito di ricevere e interpretare i comandi e gestire lo scambio dei messaggi da e verso il demone (un client può comunicare con diversi demoni);
- una REST API: definisce l'interfaccia tramite la quale il client può parlare con il demone e istruirlo.

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

Le immagine sono memorizzare all'interno di un registro, il quale può essere pubblico o privato e può trovarsi sulla stessa macchina che ospita il demone o il client o su un server separato.

3.5.2 KVM

KVM [10] (Kernel-based Virtual Machine) è una soluzione di virtualizzazione per Linux su hardware x86 e che contiene estensioni alla virtualizzazione (Intel VT o AMD-V). Esso è costituito da un modulo kernel, kvm.ko, che rappresenta la parte

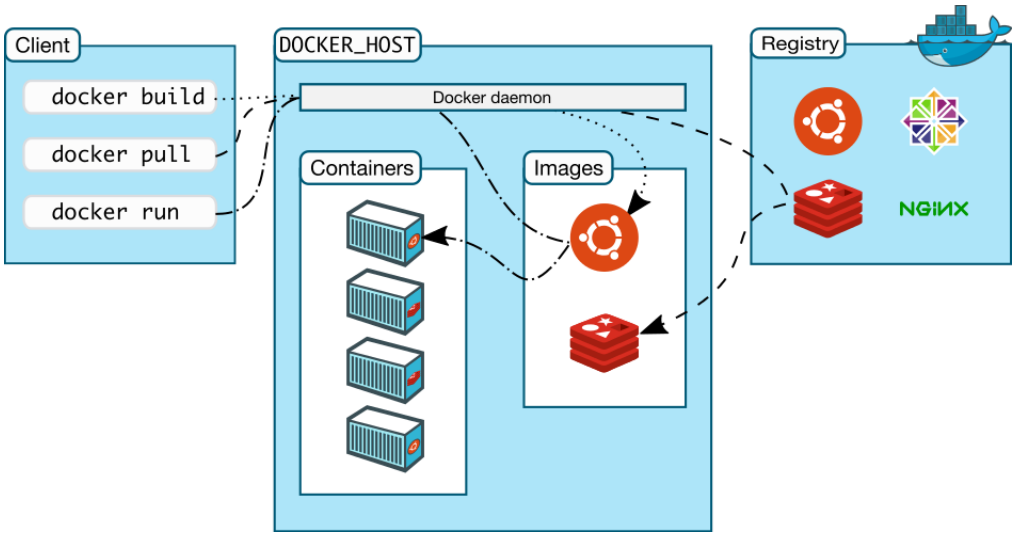

Figura 3.2. Docker Engine

principale dell'infrastruttura di virtualizzazione e da un modulo specifico, kvm-intel.ko o kvm-amd.ko, dipendente dal processore utilizzato. Con l'uso di KVM, si possono eseguire istanze multiple di macchine virtuali in cui girano immagini non modificate di Linux o Windows, assegnando ad ogni macchina virtuale un proprio hardware virtualizzato: una scheda di rete, un disco, un adattatore grafico e così via.

Il sistema operativo vede KVM come se fosse un tipico dispositivo Linux (/dev/kvm) [11] che lo userspace può utilizzare per gestire il ciclo di vita delle macchine virtuali, in modo da poter svolgere operazioni come la creazione e l'allocazione di memoria di una nuova macchina virtuale, la lettura e scrittura dei registri della cpu virtuale e altro.

Il ciclo di esecuzione di una virtual machine, come si vede in figura 3.3, può riassumersi nei seguenti punti:

- a livello più esterno, lo userspace delega il kernel affinché esegua il codice della VM (modalità Guest) fino a quando non incontra un'istruzione di I/O, o fino a quando non si verifica un evento esterno, come l'arrivo di un pacchetto di rete o lo scadere di un timeout;

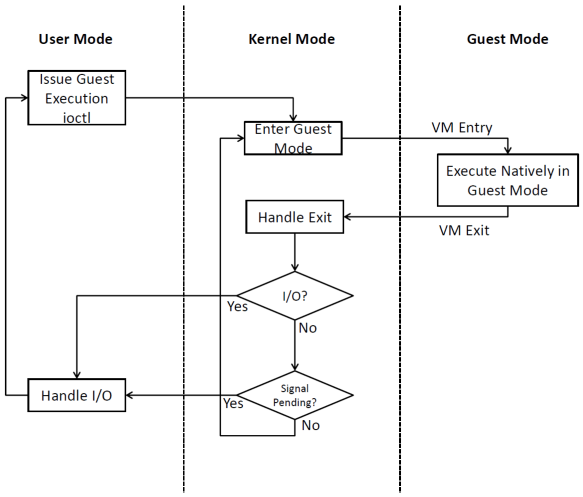

Figura 3.3. KVM - Processamento di una Virtual Machine

- a livello Kernel, si forza l'hardware ad entrare in modalità Guest. Se, a causa di un evento, come ad esempio un interrupt esterno, il processore esce da questa modalità, il kernel chiama la relativa procedura di gestione e, una volta terminata questa, riprende l'esecuzione in modalità Guest. Se l'uscita è stata causata da un'istruzione I/O o dalla ricezione di un segnale da parte del processo, il kernel ritorna allo userspace;
- a livello hardware, il processore esegue il codice Guest fino a quando non incontra un'istruzione per cui è richiesto l'intervento del kernel, un fault o un interrupt esterno.

3.6 PPPoE

Il PPPoE (Point-to-Point Protocol over Ethernet), seppure non sia uno standard Internet, è descritto nell'informativo RFC 2516 [12] ed è stato sviluppato da RedBack Networks, RouterWare, UUNET e altri. Nasce nel 1999 quando con il diffondersi delle tecnologie xDSL rappresentava la soluzione per poter inviare i pacchetti da una rete di host verso la rete IP di un ISP (Internet Service Provider). PPPoE è infatti un protocollo di tunneling punto-punto tra un end user CPE e il BRAS

(Broadband Network Gateway) di un ISP, che permette di incapsulare pacchetti PPP all'interno di frame Ethernet. In questo modo si riescono a fornire le caratteristiche tipiche del protocollo PPP, come ad esempio autenticazione, cifratura e compressione, che soddisfano funzionalità di controllo degli accessi e di fatturazione richieste da un provider.

Per fornire una connessione punto-punto su Ethernet, ciascuna sessione PPP deve imparare l'indirizzo Ethernet del peer remoto, oltre a stabilire un identificatore di sessione univoco. PPPoE include un protocollo di rilevamento che fornisce tale identificatore.

PPPoE ha due fasi distinte: una fase di scoperta (*Discovery stage*) e una fase di sessione PPP (*PPP Session stage*). Quando un host desidera avviare una sessione PPPoE, deve prima eseguire la fase di scoperta per identificare l'indirizzo MAC del peer e stabilire un `SESSION_ID` PPPoE. Mentre la fase di sessione PPP definisce una relazione punto-punto, quella di Discovery è intrinsecamente una relazione client-server. Nel processo di rilevamento, un host (il client) rileva un *Access Concentrator (AC)* (il server). In base alla topologia della rete, è possibile che ci sia più di un AC con cui l'host può comunicare. La fase di Discovery consente all'host di scoprire tutti gli AC e selezionarne uno. Quando la fase di Discovery viene completata con successo, sia l'host che l'AC selezionato dispongono delle informazioni che useranno per creare la connessione point-to-point su Ethernet.

La fase di Discovery rimane stateless finché non viene stabilita una sessione PPP. Una volta stabilita, sia l'host che il concentratore di accesso devono allocare le risorse per un'interfaccia virtuale PPP. Una volta completata la negoziazione e stabilita la connessione l'AC invia occasionalmente dei paccheti *Echo-Request* all'host per determinare lo stato della sessione. Altrimenti, se l'host termina una sessione senza inviare un pacchetto di terminazione (*Terminate-Request packet*), l'AC non sarà in grado di determinare che la connessione è stata chiusa.

3.6.1 Discovery Stage

Ci sono quattro passaggi per la fase di Discovery. Al termine, entrambi i peer conoscono il `SESSION_ID` PPPoE e l'indirizzo Ethernet del peer, che insieme definiscono la sessione PPPoE in modo univoco. I passaggi consistono sono i seguenti:

- l'host che trasmette un pacchetto di inizializzazione (*Initiation packet PADI*);
- uno o più AC inviano pacchetti contenenti un'offerta (*Offer packets PADO*);
- l'host invia un pacchetto unicast di *Session Request* (*PPPoE Active Discovery Request Packet - PADR*) all'AC che ha selezionato (la decisione può dipendere dal nome dell'AC o dal servizio offre);
- l'AC selezionato invia un pacchetto di conferma (*Confirmation Packet PADS*).

Quando un host non riceve un pacchetto PADO entro uno specifico intervallo di tempo, invia nuovamente il pacchetto PADI e raddoppia il tempo di attesa di risposta. Allo stesso modo, se un host è in attesa di un pacchetto PADS è utilizzato un meccanismo di timeout simile per l'invio di nuovo pacchetto PADR. Dopo uno specifico numero di tentativi, l'host invia un nuovo pacchetto PADI.

Quando l'host riceve il pacchetto di conferma, può procedere alla fase di sessione PPP. Allo stesso modo, quando l'Access Concentrator invia il pacchetto di conferma, può procedere alla fase di sessione PPP.

3.6.2 PPP Session Stage

Una volta iniziata la sessione PPPoE, i dati PPP vengono inviati dentro un ulteriore incapsulamento PPP e tutti i pacchetti Ethernet sono unicast. Il payload PPPoE contiene un frame PPP che inizia con Protocol ID `PPP`.

Capitolo 4

Universal Node

Lo universal-node [13] è un nodo di computing altamente performante che permette di istanziare servizi virtualizzati tramite l'uso di diversi motori di virtualizzazione (macchine virtuali, container ed altro). Esso si adatta bene a girare su comuni server localizzati nei data center ma anche su hardware con basse capacità di calcolo (resource-constrained), come ad esempio un home-gateway.

Lo universal-node facilita anche l'orchestrazione orientata ai domini e consente l'ottimizzazione di servizi computazionali e di risorse di rete, caratteristiche che determinano un'architettura unica rispetto alle altre piattaforme che supportano la virtualizzazione dei servizi.

4.1 Overview

Con la tecnologia Network Functions Virtualization (NFV) si sta modificando il modo in cui i servizi vengono forniti, favorendo l'istanziamento di funzioni di rete attorno all'intera rete eterogenea di un ISP (Internet Service Provider), a partire dal CPE a casa degli utenti fino ai server nei data center dell'operatore di rete.

Lo universal-node vuole garantire un deploy e una distribuzione dei servizi in maniera efficiente su queste risorse, in modo che si abbia un'astrazione di alto livello per comporre e realizzare servizi virtualizzati. Inoltre, sfruttando le informazioni locali di cui dispone, ottimizza l'istaziamento dei servizi; ad esempio, valuta le risorse e i vincoli dell'ambiente per scegliere l'implementazione migliore della funzione

che è stata richiesta, in modo che si possono sfruttare a pieno le risorse disponibili.

Il software dello universal-node può essere eseguito su diverse piattaforme hardware, sia server dotati di una CPU Intel x86 e diversi GB di memoria e disco, sia dispositivi con processori ARM o MIPS che hanno risorse hardware limitate e magari anche una versione custom del sistema operativo Linux. Inoltre riesce anche ad eseguire funzioni di rete in diversi ambienti di esecuzione che riescono ad adattarsi bene a differenti contesti operativi.

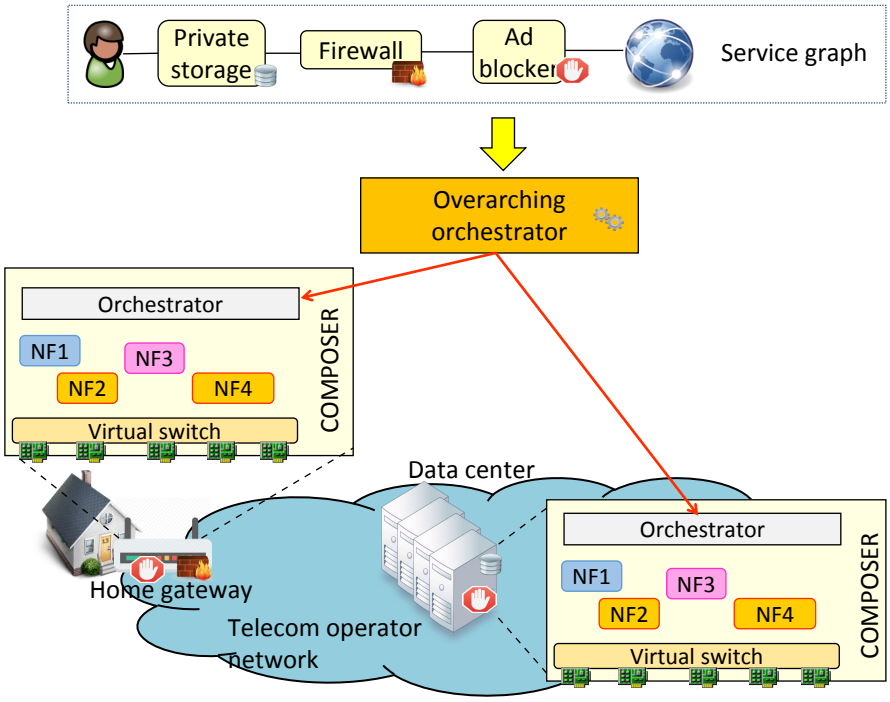

Figura 4.1. Istanziamento di funzioni di rete in un'infrastruttura equipaggiata con lo universal-node

La figura 4.1 mostra come ogni risorsa computazionale disponibile nella rete dell'operatore può far girare il software dello universal-node, ricevere istruzioni da uno o più orchestratori ed eseguire tutte le operazioni che sono richieste per rendere funzionale il servizio.

4.2 Architettura

Nella figura 4.2 viene mostrata l'architettura ad alto livello dello universal-node composta dai seguenti blocchi:

Lo universal-node-orchestrator, che si può vedere in dettaglio nella figura 4.3, costituisce il componente principale del piano di controllo dello universal-node, in quanto fornisce astrazione di rete e di calcolo computazionale, facilita l'orchestrazione orientata ai domini e consente l'ottimizzazione sia della parte computazionale che delle risorse di rete del grafo di servizio (*service graph*). Esso orchestra risorse computazionali e di rete, quindi gestisce il ciclo di vita completo dell'ambiente di esecuzione virtuale e le primitive di rete (ad esempio, le regole di *traffing steering*).

Lo universal-node-orchestrator riceve i comandi attraverso un'interfaccia di *northbound* e si occupa di implementare i servizi sull'infrastruttura del nodo, facendo affidamento sul *VNF Repository* per selezionare la migliore implementazione delle VNF (Virtual Network Function) disponibili per il servizio richiesto. Il VNF Repository può essere anche su un altro server e può essere contattato da più universal-node.

Il piano dati include invece uno switch virtuale (*vSwitch*) che gestisce i percorsi del traffico tra le varie NF, e un numero di motori computazionali che possono eseguire funzioni di rete implementate con diverse tecnologie (ad esempio virtual machine, container e altro).

4.2.1 Interfaccia di Northbound

Lo universal-node-orchestrator interagisce con l'orchestratore o gli orchestratori superiori tramite un'interfaccia di northbound bidirezionale. In particolare, riceve un grafo di servizio descritto secondo il formalismo NF-FG (*Netork-Function-Forwarding-Graph*), ed esporta le informazioni tramite un modello YANG, compatibile con OpenConfig e che descrive il dominio dello universal-node.

Come si vede nella figura 4.3, i comandi di creazione, lettura, aggiornamento e distruzione (operazioni CRUD) correlate al NF-FG sono ricevute attraverso una REST API, mentre la descrizione dello universal-node è esportata attraverso Double

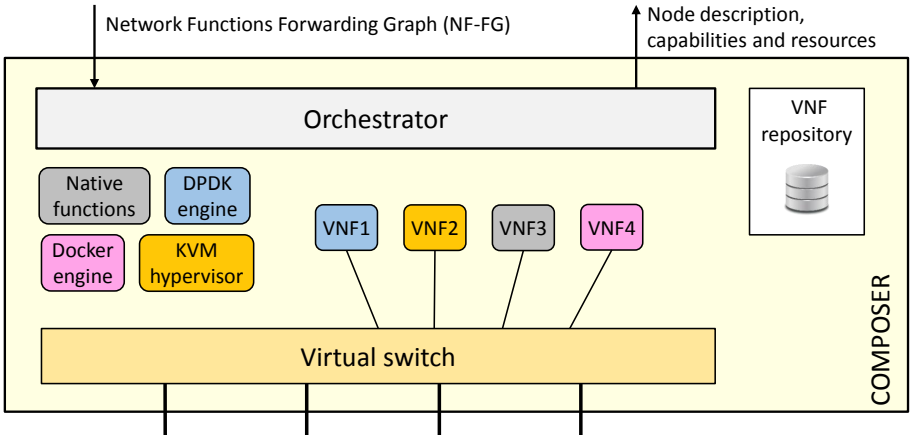

Figura 4.2. Overview architettura dello universal-node

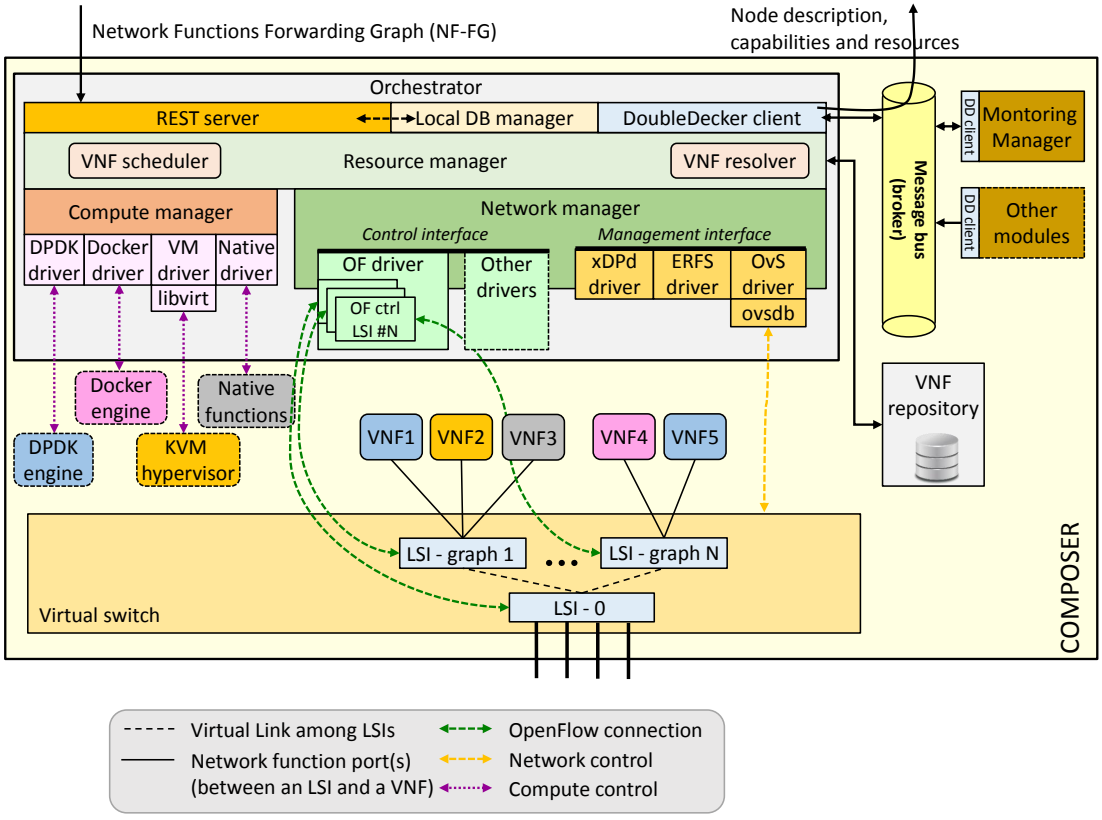

Figura 4.3. Architettura dettagliata dello universal-node.

Decker (DD) [14], un bus gerarchico di messaggi basato su 0MQ [15] e, tra le altre cose, supporta un modello *publish/subscribe*.

Di seguito si spiega il motivo che sta dietro la scelta di tale broker dei messaggi. L'orchestratore globale conosce esattamente l'entità universal-node (cioè l'indirizzo IP dello universal-node) sul quale il grafo di servizio (o parte di esso) deve essere creato/aggiornato/rimosso, quindi un'interfaccia di tipo REST è appropriata per questa funzione. D'altra parte, utilizzando DD, lo universal-node non ha bisogno di tener traccia dei consumatori della descrizione che esporta, poiché potrebbero esserci diverse entità interessate a questa informazione. Tutti gli universal-node pubblicano la loro descrizione attraverso DD utilizzando uno specifico *topic*, permettendo a tutte le entità interessate a questa informazione di sottoscriversi a quel topic. In questo caso, un esempio pertinente è l'orchestratore globale sopra menzionato , che può utilizzare la descrizione dello universal-node per scegliere il nodo dove il servizio deve essere istanziato.

Come descritto in figura 4.3, il server REST interagisce con il *security manager*, un modulo che gestisce l'autenticazione e controlla i permessi delle entità che mandano i comandi allo universal-node. Per esempio, solo l'operatore di rete può avere l'autorizzazione di istanziare gli NF-FG, mentre gli utenti possono avere solo il permesso di leggere i grafi di servizio che operano sulla loro connessione Internet.

Network Functions - Forwarding Graph

Il formalismo Network Functions - Forwarding Graph (NF-FG) descrive il servizio che deve essere istanziato nello universal-node sia dal punto di vista computazionale (la composizione delle funzioni del servizio) che dal punto di vista della rete (regole di traffic steering).

Come mostra il listing 4.1, ogni NF-FG è costituito da tre parti principali. Innanzitutto, la sezione `VNFs` elenca le funzioni che compongono il servizio. In particolare, il NF-FG può richiedere una funzione senza specificare come essa debba essere implementata (`firewall` in figura). In questo caso l'immagine appropriata è selezionata dallo universal-node attraverso l'interazione con il VNF repository. Tuttavia, il NF-FG può anche richiedere una specifica implementazione di una funzione, specificando un *template* che la descriva nello specifico in termini di, ad

esempio, immagine da eseguire, numero di CPU core richiesti, tecnologia da usare per implementare la *virtual network interface cards (vNICs)*, e così via.

Listing 4.1. Esempio di Network Function - Forwarding Graph (NF-FG)

```
{
   "nf-fg": {
      "id": "0x1",
      "name": "example graph",
      "VNFs": [
      {
         "id": "0x1",
         "name": "firewall",
         "ports": [
         {
            "id": "0xa",
            "name": "internal port"
         },
         ....
         ]
      },
      ....
      ]
      "service-access-points": [
      {
         "id": "0x1",
         "type": "interface",
         "interface":  {
            "if-name": "eth1"
         }
      },
      ....
      ],
      "flow-rules": [
      {
         "id": "0x1",
         "priority": 1,
         "match": {
            "port_in": "service-access-point:0x1",
            ....
         },
         "actions":  {
            "output_to_port": "vnf:0x1:0xa",
            ....
         }
      },
      ....
      ]
   }
}
```

La sezione `service-access-points` descrive i punti di ingresso/uscita del traffico del grafo di servizio istanziato nello universal-node. I *Service access points (SAPs)* possono essere usati sia nelle parti del match che in quelle delle azioni delle regole di traffic steering.

Lo universal-node implementa quattro tipi di punti di accesso: *Interface*, *VLAN*, *GRE-tunnel* e *Host-Stack*. L'endpoint *Interface* (Listing 4.1) corrisponde ad una interfaccia fisica dello universal-node, mentre un punto di accesso *vlan* include solamente il traffico associato ad una interfaccia fisica e appartenente ad una specifica

VLAN. Questo significa che lo universal-node garantisce che solamente il traffico con un dato VLAN ID arrivi da questo endpoint (VLAN ID 25 in Listing 4.2), e che tutto il traffico da esso spedito venga taggato (dallo universal-node stesso) con il dovuto VLAN ID.

Listing 4.2. Esempio di un punto di accesso VLAN nel NF-FG

```
{
    "id": "0x2",
    "type": "vlan",
    "vlan":
    {
        "vlan-id": "25",
        "if-name": "eth1"
    }
}
```

Il punto di accesso *GRE-tunnel* (un esempio si vede nel listing 4.3) rappresenta la terminazione di un tunnel GRE.

Lo universal-node garantisce che solo il traffico incapsulato in uno specifico tunnel GRE entri da questo punto di accesso, e che tutto il traffico da essa uscente venga incapsulato nello stesso tunnel GRE. Entrambi i punti di accesso *vlan* e *GRE-tunnel* possono essere usati per pilotare il traffico tra parti diverse dello stesso grafo di servizio istanziato in differenti nodi.

Listing 4.3. Esempio di un punto di accesso GRE nel NF-FG

```
{
    "id": "0x3",
    "type": "gre-tunnel",
    "gre-tunnel":
    {
        "local-ip": "10.0.0.1",
        "remote-ip": "10.0.0.2",
        "gre-key" : "0x1"
    }
}
```

Il punto di accesso *Host-Stack* rappresenta il punto di connessione tra il grafo di servizio e lo stack TCP/IP del dispositivo su cui è in esecuzione lo universal-node. Questo permette ad ogni servizio TCP/IP eseguito sull'host (ad esempio il REST server dello universal-node) di connettersi al mondo esterno attraverso un grafo di servizio, riuscendo a controllare la loro connessioni alla rete (ad esempio, il REST server dello universal-node potrebbe essere protetto da un firewall istanziato in un grafo di servizio).

Infine, la sezione `flow-rules` nel listing 4.1 descrive le interconnessioni tra le porte delle funzioni di rete e i punti di accesso. La semantica utilizzata è simile a quella del protocollo Openflow in quanto ogni connessione è caratterizzata da:

- una priorità;
- un match su un punto di accesso o una porta di una VNF e potenzialmente su campi protocollari (ad esempio IP source);
- un'azione per inviare il traffico verso una specifica porta e che potenzialmente modifica il contenuto dei pacchetti (ad esempio decrementa il valore IPv4 TTL)

Descrizione del dominio

La descrizione del dominio pubblicata dallo universal-node-orchestrator tramite DD include sia caratteristiche computazionali che di rete dell'infrastruttura del nodo.

Dal punto di vista della rete, lo universal-node viene astratto come uno "*big-switch*" con un insieme di endpoint, ognuno caratterizzato dalle seguenti informazioni (opzionali):

- neighbor domain: identifica il dominio di un altro nodo (ad esempio lo universal-node), in presenza di una connessione diretta tra i due;
- IP address: indirizzo IP del nodo in questione;
- supporto al traffico VLAN: se presente, gli endpoint indicano i VLAN ID disponibili;
- supporto per i tunnel GRE.

Dal punto di vista computazionale, lo universal-node esporta invece le *capabilities* funzionali e infrastrutturali, nonché le risorse disponibili. Le capabilities funzionali rappresentano la capacità dello universal-node di implementare specifiche funzioni (ad esempio NAT, firewall), opzionalmente con certe caratteristiche come la possibilità di gestire traffico alla velocità di 1Gbps. Le capabilities infrastrutturali fanno invece riferimento alle caratteristiche di basso livello dello universal-node,

come ad esempio l'architettura della CPU, la capacità di eseguire VM piuttosto che Docker container, ed altro ancora. Infine, le risorse disponibili includono informazioni quali la quantità di memoria libera o la presenza di un acceleratore hardware.

4.2.2 Modello di traffic steering

Come mostrato in figura 4.3, lo universal-node-orchestrator crea i percorsi di rete tra le porte delle VNF e i punti di accesso attraverso 2 livelli di LSI. La base `LSI-0` sta sotto un insieme di LSI (`graph-LSI`), ognuno incaricato di implementare i percorsi attraverso le VNF di grafi differenti.

`LSI-0` è creato al boot e gestisce il traffico scambiato tra le interfacce fisiche e i `graph-LSI`, mentre gli LSI aggiuntivi (ognuno creato quando un nuovo NF-FG deve essere istanziato) implementano i percorsi tra le porte delle VNF, l'*Host-Stack* e i tunnel GRE che appartengono al grafo. Infatti, mentre le interfacce fisiche sono connesse al `LSI-0`, le porte delle VNF, l'*Host-Stack* e i tunnel GRE sono connessi al `graph-LSI` associato.

Poiché ogni `graph-LSI` è connesso alle porte delle VNF, all'*Host-Stack* e ai tunnel GRE di un differente NF-FG, lo universal-node-orchestrator può implementare la *multi-tenancy* e isolare il traffico di diversi tenant, dove `LSI-0` diventa l'unico LSI attraversato da pacchetti appartenenti a multipli tenants/service graph.

Inoltre, la gerarchia degli LSI si prende carico di rimuovere gli incapsulamenti utilizzati per implementare i punti di accesso vlan e GRE, nel caso di traffico che arriva da un tale endpoint. Allo stesso modo, le NF non sono consapevoli del fatto che il traffico che trasmettono sarà inviato, dalla gerarchia degli LSI, attraverso i punti di accesso vlan e/o GRE; anche in questo caso, infatti è la gerarchia LSI che incapsula i pacchetti negli opportuni header, in accordo con le regole descritte nel NF-FG.

Oltre a questo, nessuno degli LSI può essere programmato da un controller SDN esterno, appartenente, ad esempio, al possessore del service graph; gli LSI sono infatti usati dallo universal-node-orchestrator per implementare i percorsi di rete descritti nel NF-FG e sono sotto il completo controllo dello universal-node-orchestrator stesso (attraverso il *network manager*, come dettagliato in seguito).

Infine, gli LSI forniscono allo universal-node il completo controllo sul piano reti delle NF, in quanto esso può implementare le connessioni di rete come definite nel NF-FG. Invece, se lo universal-node usasse il modello di rete offerto dal motore computazionale sfruttato per far eseguire le funzioni di rete (ad esempio, KVM e Docker collegano le VM a i container ad un bridge L2), non avrebbe il controllo totale dei percorsi di rete.

4.2.3 Node Resource Manager

Il *Node resource manager* è il principale modulo dello universal-node-orchestrator, in quanto gestisce i comandi ricevuti tramite la REST API ed esporta la descrizione del dominio al boot del nodo ed ogni volta che avviene qualche cambiamento nella configurazione.

In accordo con quanto mostrato dal diagramma dei messaggi in figura 4.4, quando lo universal-node-orchestrator riceve il comando di creare un nuovo grafo a partire dal NF-FG, il node resource manager:

1. interagisce con il VNF repository per selezionare l'immagine più appropriata di VNF per ogni funzione che fa parte del servizio e che non è esplicitamente associata ad un'immagine nel NF-FG;

2. configura gli vSwitch per creare un nuova istanza di uno switch logico (Logical Switch Instance - LSI) e le porte richieste per connettere il nuovo switch alle VNF da istanziare;

3. istanzia ed avvia le VNF selezionate;

4. configura le tabelle di forwarding degli LSI in accordo con le regole di traffic steering che devono essere implementate.

Allo stesso modo, il node resource manager si fa carico di aggiornare o distruggere un grafo, quando vengono ricevuti i comandi corrispondenti.

La figura 4.3 mostra che lo universal-node-orchestrator include, tra gli altri moduli, il *network manager* ed il *compute manager*, che sono impiegati dal node resource manager per interagire rispettivamente con il vSwitch ed il motore esecutivo al fine di creare i corretti grafi ed avviare le corrette VNFs.

Il *VNF resolver* interagisce con il VNF repository e seleziona la migliore implementazione per la funzione richiesta, in base ad alcuni parametri quali ad esempio la quantità computazionale e le risorse di memoria disponibili nello universal-node.

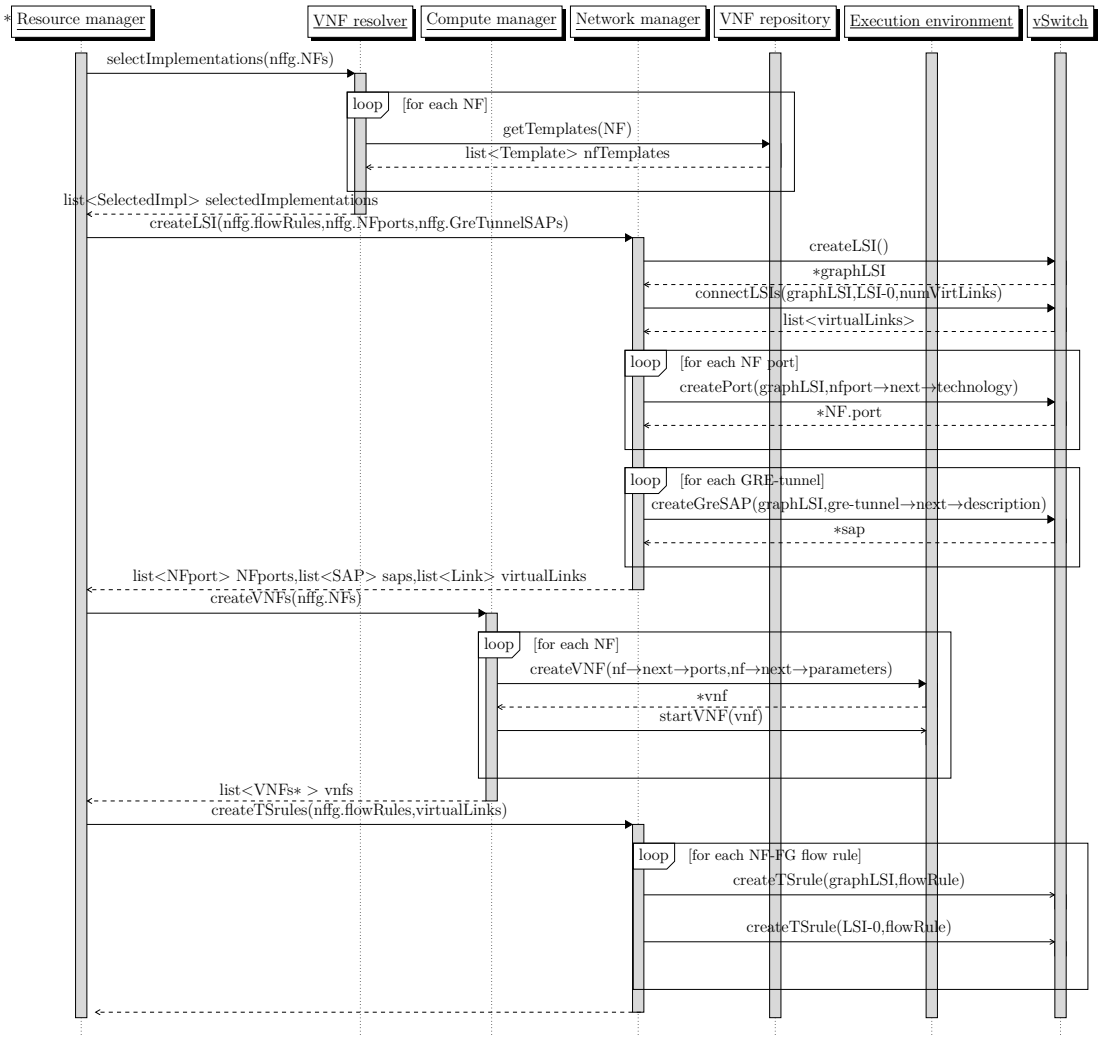

Figura 4.4. Diagramma della sequenza di messaggi che portano all'istanziamento di un nuovo NF-FG.

4.2.4 Network Manager

Il modulo *network manager* gestisce la parte di rete del grafo di servizio. Esso può interagire con diversi vSwitch (con possibili caratteristiche differenti) al fine di

creare l'infrastruttura virtuale di rete che implementa i percorsi descritti nel NF-FG, e che supporta l'istanziamento di multipli service graph (in base al modello di traffic steering presentato nella sezione 4.2.2).

La creazione dei percorsi descritti nel NF-FG richiede l'interazione tra il network manager e il vSwitch per:

1. creare un nuovo `graph-LSI` con le porte virtuali richieste che saranno poi collegate alle VNF per mezzo dell'interfaccia del *management plane*;

2. programmare la forwarding table di `LSI-0` e del nuovo `graph-LSI` così da realizzare il traffic steering attraverso l'interfaccia del *control plane*.

L'interfaccia del management plane consente al network manager di gestire differenti vSwitch (in assenza di informazioni riguardo alla tecnologia dello switch) attraverso un insieme di primitive listate nella tabella 4.1 e implementate da ogni driver specifico per ogni tecnologia.

Function	**Description**
lsi ∗createLSI()	Create a new LSI
void destroyLSI(lsi)	Delete a specific LSI
list<∗link> connectLSIs(lsi1,lsi2,N)	Create `N` virtual links between two LSIs
void destroyVlink(link)	Destroy a virtual link between two LSIs
port ∗createPort(lsi,technology)	Create a (VNF) port with a specific technology on an LSI
void destroyPort(port)	Destroy a specific (VNF) port
sap ∗createSAP(lsi, description)	Create a SAP on an LSI, according to a specific description
void destroySAP(sap)	Delete a specific SAP

Tabella 4.1. Interfaccia del management plane.

Fondamentalmente, queste primitive permettono al network manager di:

1. creare/distruggere un LSI;

2. creare/distruggere una porta che verrà connessa a una VNF;

3. creare/distruggere virtual link (connessioni tra due LSI);

4. creare/distruggere punti di accesso.

Il supporto di diverse tecnologie di switching permette la selezione del miglior vSwitch per lo scenario in questione. In particolare il prototipo dello universal-node, oltre al diffuso Open vSwitch [8], supporta *extensible Data-Path deamon (xDPd)* [16] ed *Ericsson Research Flow Switch (ERFS)* [17]. Infatti, se da un lato xDPd supporta l'offloading delle regole OpenFlow sull'hardware sottostante e si adatta bene nel caso in cui lo universal-node gira su un box con hardware con capabilities di switching, ERFS è molto veloce ma richiede che sia eseguito su server di alta fascia, a causa della sua alta domanda in termini di CPU core. Altri vSwitch possono essere supportati scrivendo il corrispondente driver che implementa le primitive in tabella 4.1.

In modo simile, l'interfaccia del *control plane* consente al network manager di configurare la forwarding table degli LSI tramite l'uso di diverse tecnologie (ad esempio, OpenFlow, eBPF, P4 [18]), nascondendo, al tempo stesso, al network manager la tecnologia usata.

L'insieme di primitive definite dall'interfaccia del control plane è elencato nella tabella 4.2. Le specifiche funzioni in tabella devono essere implementate dai controller specifici della tecnologia usata al fine di permettere al network manager di inserire/rimuovere regole di traffic steering in/da uno specifico LSI.

Function	**Description**
void createTSRule(lsi,rule)	Send a traffic steering rule to the LSI
void deleteTSRule(lsi,rule)	Remove a traffic steering rule from the LSI

Tabella 4.2. Interfaccia del control plane.

Come mostrato in figura 4.3, per ogni LSI vengono creati controller di diverse tecnologie, che controllano la forwarding table del LSI stesso. Il corrente prototipo supporta OpenFlow per il traffic steering; così, come si vede in figura, ogni controller specifico per ogni tecnologia è in realtà un controller OpenFlow, ed i messaggi OpenFlow (*flowmod*) vengono usati per installare le regole di traffic steering.

Anche in questo caso, altre tecnologie per programmare la forwarding table del vSwitch possono essere supportate scrivendo il corrispondente driver che implementa le primitive in tabella 4.2.

Traduzione delle regole del NF-FG in regole di traffic steering

Al fine di implementare i percorsi di rete descritti nel NF-FG, il network manager deve mappare la sezione `flow-rules` nel modello di traffic steering definito nella sezione 4.2.2. In particolare, questo modello richiede che, per ogni NF-FG da istanziare, il network manager:

In particolare, questo modello richiede che, per ogni NF-FG da istanziare, il network manager:

1. connetta il nuovo `graph-LSI` al `LSI-0` attraverso un certo numero di virtual link;

2. cominciando dalla sezione `flow-rules` del NF-FG, origini due insiemi di regole di traffic steering da installare rispettivamente nel `LSI-0` e nel nuovo `graph-LSI`.

In accordo con la tabella 4.3, alcune regole di flusso possono essere implementate su un singolo LSI, qualora match ed azione coinvolgano porte/punti di accesso connessi allo stesso LSI, mentre altre regole devono essere divise in regole di traffic steering per il `LSI-0` e per il `graph-LSI`. Mentre le prime regole di flusso non richiedono nessun virtual link, in quanto tengono il traffico all'interno di un LSI, le altre richiedono la presenza di virtual link per trasferire i pacchetti da un LSI all'altro.

		ACTION: Output to	
		Interface / VLAN SAP	**VNF Port / GRE-tunnel**
MATCH	**Interface / VLAN SAP**	only LSI-0 (no vlink needed)	LSI-0 and graph-LSI
	VNF Port / GRE-tunnel	LSI-0 and graph-LSI	only Graph-LSI (no vlink needed)

Tabella 4.3. LSI coinvolti nell'implementazione delle flow rules con specifici match/action.

In particolare, come mostrato nell'algoritmo 1, per ogni porta di una VNF/punto di accesso che appare come azione nelle regole che coinvolgono entrambi gli LSI

si crea un diverso virtual link (vedi Tabella 4.3), che è poi usato per spostare da un LSI all'altro tutto il traffico che deve essere spedito sulla specifica porta di una VNF/punto di accesso. In accordo con lo pseudo codice, al fine di minimizzare il numero di virtual link, lo stesso virtual link può essere, in realtà, usato in entrambe le direzioni per mandare verso il `graph-LSI` tutto il traffico destinato a una porta di una VNF o a un punto di accesso di tipo tunnel GRE o a un punto di accesso di tipo Host-Stack e verso il `LSI-0` tutto il traffico destinato a una specifica interfaccia o a un endpoint VLAN.

Algorithm 1 Creazione dei virtual link.

```
1: procedure createVlinks(first_id,nffg,lsi0,graphLsi)
2: vlink_to_lsi0 ← vlink_to_graphlsi ← first_id
3: association ← ∅
4: for all r ∈ nffg.bigswitch do
5:    if vlink_needed[r.match.port][r.action.out] and association[r.action.out] ∈ ∅ then
6:       if r.action.out ∈ vnf_port or r.action.out ∈ gre_tunnel then
7:          association[r.action.out] ← vlink_to_graphlsi
8:          vlink_to_graphlsi ← vlink_to_graphlsi+1
9:       else if r.action.out ∈ interface_sap or r.action.out ∈ vlan_sap then
10:         association[r.action.out] ← vlink_to_lsi0
11:         vlink_to_lsi0 ← vlink_to_lsi0+1
12:      end if
13:   end if
14: end for
15: N ← max(vlink_to_lsi0,vlink_to_graphlsi)−first_id
16: return connectLSIs(lsi0,graphLsi,N)
```

Mentre la trasformazione delle flow rule che possono essere implementate in un singolo LSI in regole di traffic steering non richiede nessuna operazione (un esempio è fornito dalla flow rule `#3` del NF-FG in figura 4.5), l'algoritmo 2 mostra come, invece, il network manager derivi le regole di traffic steering corrispondenti alle regole del NF-FG che coinvolgono entrambi gli LSI, dopo aver associato le porte delle VNF/i punti di accesso ad uno dei virtual link appena creati.

In accordo con le linee `#4-#12` dello pseudo codice dell'algoritmo 2, le regole la cui porta di output è connessa al `graph-LSI` generano due regole di traffic steering come segue: il match della regola nel `LSI-0` corrisponde al match della regola originale (linea `#7`), mentre l'azione differisce dall'originale solamente nella porta

Algorithm 2 Creazione delle regole di traffic steering.

```
1: procedure splitRules(association,nffg)
2: for all r ∈ nffg.bigswitch do
3:    if vlink_needed[r.match.port][r.action.out] then
4:       if r.action.out ∈ vnf_port or r.action.out ∈ gre_tunnel then
5:          {The rule brings traffic from LSI-0 to the graph-LSI}
6:          {Create the rule for LSI-0}
7:          rule-LSI0.match ← r.match
8:          rule-LSI0.action.out ← association[r.action.out]
9:          rule-LSI0.action.other ← r.action.other
10:         {Create the rule for the graph-LSI}
11:         rule-graphLSI.match.port ← association[r.action.out]
12:         rule-graphLSI.action.out ← r.action.out
13:      else if r.action.out ∈ interface_sap or r.action.out ∈ vlan_sap then
14:         {The rule brings traffic from the graph-LSI to LSI-0}
15:         {Create the rule for LSI-0}
16:         rule-LSI0.match.port ← association[r.action.out]
17:         rule.LSI0.action.out ← r.action.out
18:         {Create the rule for the graph-LSI}
19:         rule-graphLSI.match ← r.match
20:         rule-graphLSI.action.out ← association[r.action.out]
21:         rule-graphLSI.action.other ← r.action.other
22:      end if
23:   end if
24: end for
```

di output; infatti, l'azione ha come output il virtual link che trasporta al `graph-LSI` tutti i pacchetti verso la porta originale (linea `#8`).

Di conseguenza (linee `#11-#12`) la regola per il `graph-LSI` fa match sul corretto virtual link e manda tutto il traffico nella porta di output della regola originale. Questo comportamento può essere osservato nella flow rule `#1` della figura 4.5, dove alla `port 1` del `NAT` è associato il virtual link `vlink1`.

Le linee `#13-#21` dell'algoritmo 2 gestiscono le flow rules la cui azione manda i pacchetti su una porta connessa al `LSI-0`. Diversamente dal caso precedente, adesso è il match della regola nel `graph-LSI` che corrisponde al match della regola originale (linea `#19`), così come l'azione della regola nel `graph-LSI` è uguale all'azione originale eccetto che per il campo porta di output, che corrisponde al virtual link che porta verso `LSI-0` tutti i pacchetti destinati alla porta di output originale. Infine, le linee `#16-#17` mostrano che la regola creata per il LSI-0 fa match sul

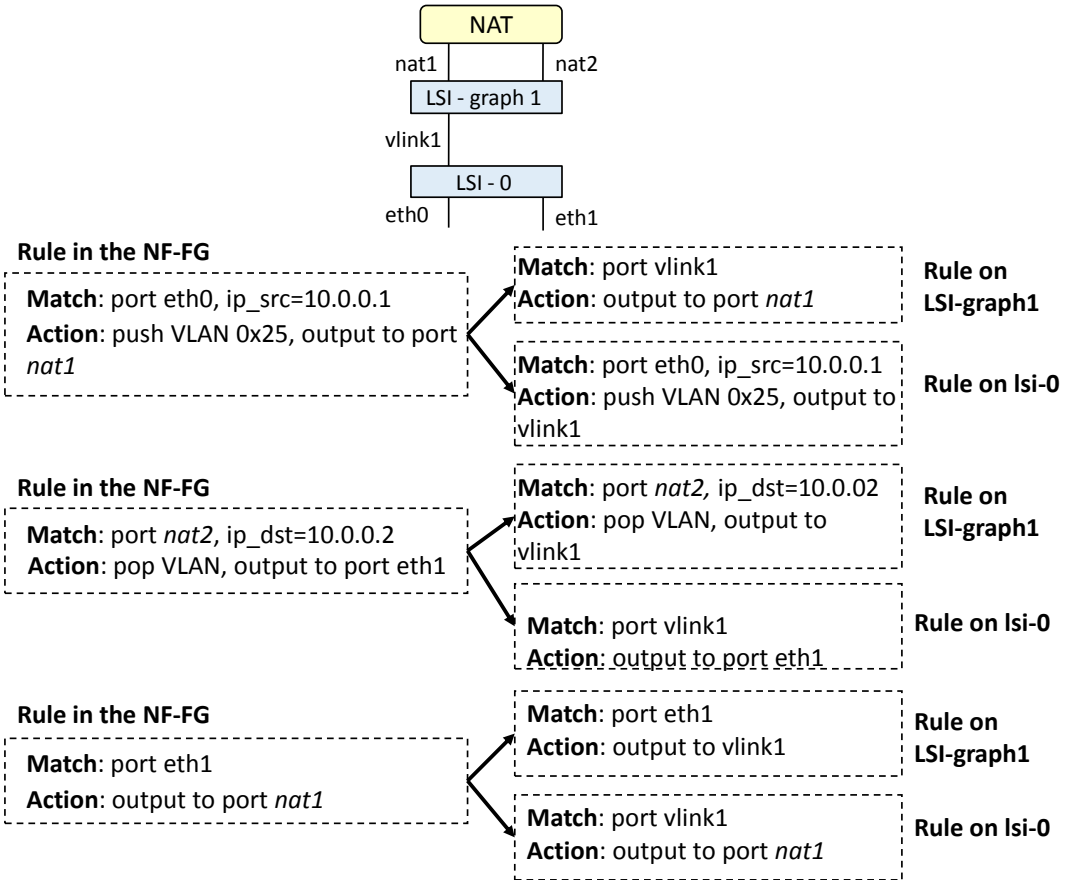

Figura 4.5. Esempio di trasformazione delle flow rules in regole di traffic steering.

corretto virtual link. Un esempio di questa procedura è mostrato nella flow rule `#2` in figura 4.5, dove il virtual link usato per trasportare il traffico verso `eth1` è lo stesso usato per mandare i pacchetti verso la porta del NAT.

4.2.5 Compute Manager

Il *compute manager* interagisce con il motore computazionale (KVM hypervisor, Docker, etc.) e gestisce il ciclo di vita delle *Network Function* (creazione, aggiornamento, distruzione di una NF), incluse le operazioni necessarie per collegare le porte delle NF già create sul vSwitch (dal network manager) alle NF stesse. Il modulo compute manager può interagire con differenti motori di esecuzione e può

anche gestire NF basate su diverse tecnologie, attraverso l'interfaccia di *compute* definita in tabella 4.4.

Function	**Description**
vnf ∗createVNF(ports,other parameters)	Allocate the resources needed by a VNF; download or create a local copy of the VNF image
void destroyVNF(vnf)	Release the resources allocated to the VNF
void startVNF(vnf)	Start a VNF previously created
void stopVNF(vnf)	Stop a VNF, without deallocating resources
void updateVNF(vnf,...)	Update a running VNF (e.g., remove/add network interfaces)
void pause(vnf)	Suspend the execution of the VNF (e.g., for a possible migration)

Tabella 4.4. Interfaccia di compute.

Come mostrato in figura 4.3, questa astrazione è implementata da un insieme di driver, ognuno incaricato di una specifica tecnologia di ambiente di esecuzione. Attualmente, il prototipo supporta l'hypervisor QEMU/KVM, i container Docker, processi basati sul framework DPDK [19], e funzioni di rete native. In particolare, il driver KVM interagisce con l'hypervisor QEMU/KVM attraverso le API di Libvirt, mentre il driver Docker usa la CLI definita da Docker per gestire le VNF eseguite nei container.

Supporto alle Native Network Functions

Diversamente dalle VM e dai container Docker che sono ben conosciuti come ambienti di esecuzione virtuale, le funzioni di rete native (*Native Network Function - NNF*) sono un nuovo concetto introdotto nello universal-node per permettere l'istanziamento dei service graph su un possibile home-gateway domestico (con i vincoli in termini di risorse che possiede).

Una funzione di rete nativa può essere definita come un componente di processamento del piano dati che sfrutta le funzionalità nativamente presenti nella piattaforma, come ad esempio i moduli software o i componenti hardware, e che viene eseguita direttamente sull'host. E' definita come un archivio `.tgz` contenente un insieme di script bash che sono chiamati dal driver *native* (vedi figura 4.2) per implementare le funzioni, definite nella tabella 4.4, richieste nella gestione del ciclo di vita di una NF.

Per essere in grado di eseguire una particolare NNF, tutti i moduli richiesti, o *dependencies*, devono essere presenti sul nodo. Quindi, oltre a tutte le informazioni richieste per l'esecuzione di una generica NF (ad esempio, il numero di porte), il template associato alla NNF include anche una lista di dipendenze, che può comprendere pacchetti software (ad esempio, eseguibili, librerie) che sono già istallati e che sono richiesti dalla NNF per poter funzionare. Il VNF resolver prende in considerazione queste dipendenze quando seleziona la migliore implementazione per la funzione richiesta.

Differentemente da tecnologie di virtualizzazione quali VM e Docker, che nativamente supportano un modello di isolamento per le NF istanziate, le NNF sono semplici script eseguiti nel sistema operativo dell'host. Quindi, il driver delle funzioni native necessita esplicitamente dell'implementazione di un livello in grado di fornire una forma di isolamento delle NNF dal resto del sistema. In particolare, il driver crea un *network namespace* prima di eseguire la NNF, aggiunge al namespace le porte virtuali richieste per connettere la NNF al LSI (tramite l'utilizzo delle interfacce virtuali Ethernet (`veth`)) ed infine esegue la NNF all'interno del namespace.

Il lancio di una NNF, quindi di uno script che gira sul crudo hardware, offre meno garanzie di protezione rispetto ad un software che gira in una VM o in un container Docker, in quanto le NNF non godono dell'aggiuntivo scudo protettivo fornito dall'hypervisor o dal motore di esecuzione Docker. Anche se, tuttavia, esiste una piccola protezione che limita le risorse usate dalle NNF, in termini di consumo di CPU/memoria o numero di CPU core occupati. Nonostante l'impatto dei problemi sopra citati possa essere limitato mediante l'impiego di meccanismi che Linux mette a disposizione, come `cgroup`, la complessità sarebbe tale da portare a

preferire il rimpiazzamento della NNF con un'implementazione basata su Docker.

In ogni caso non esiste nessuna protezione che impedisca ad una NF, di cui ci si aspetta un determinato servizio (ad esempio, firewall), di comportarsi differentemente, ad esempio, lanciando un attacco verso un host remoto. La soluzione corrente semplicemente fa fede sul creatore dell'applicazione o della entità (un possibile marketplace di applicazioni) che mette in commercio le applicazioni. Quindi, pur essendo a conoscenza del problema della determinazione di una NF benigna o maligna, nel caso di NNF, a causa del grado inferiore di isolamento, il sentimento comune è che questo sia un problema così generale da richiede una soluzione altrettanto generica in grado di garantire la bontà delle VNF.

4.2.6 VNF resolver

Il *VNF resolver* è il componente dello universal-node che interagisce con il VNF repository al fine di selezionare le immagini delle VNF da istanziare.

Il VNF resolver è usato nei casi in cui il NF-FG indichi solo alcune caratteristiche delle funzioni richieste (ad esempio, *firewall con tre porte*). In questo caso, esso seleziona la migliore immagine passando per i punti di seguito esposti. Innanzitutto, il VNF resolver richiede al VNF repository i template di tutte le VNF che implementano la funzione cercata. In seguito, seleziona la migliore VNF che, in accordo con il template, soddisfa i vincoli e gli attributi associati alla funzione nel NF-FG (ad esempio, 3 porte e il supporto per 1Gb/s), che è compatibile con gli ambienti di esecuzione supportati dallo universal-node e che richiede un numero disponibile di risorse del sistema (RAM,CPU). In particolare, la VNF selezionata può essere composta da una singola immagine o può essere un nuovo service graph composto da un certo numero di altre funzioni arbitrariamente connesse. In altre parole, il template può descrivere la NF o un altro NF-FG. In questo caso, il VNF resolver ripeterà ricorsivamente le stesse operazioni scritte sopra per ogni giunzione che è parte del nuovo *sub*-NF-FG, fino a quando tutte le VNF richieste siano state selezionate.

VNF repository

Il VNF repository contiene i template ed un certo numero di immagini per ogni funzione di rete. Il VNF template descrive una specifica immagine di NF in termini di funzionalità implementata (ad esempio, firewall, NAT), quantità di risorse fisiche richieste sul nodo al fine di permettere l'esecuzione dell'immagine (ad esempio, CPU, memoria), ambiente di esecuzione richiesto (ad esempio, KVM hypervisor, motore Docker, etc.), numero di interfacce virtuali e tecnologia usata ed altro ancora. L'immagine della VNF cambia, invece, a seconda della tecnologia usata per implementare la NF. Ad esempio, si tratta di un disco VM nel caso di macchine virtuali, di un insieme di script nel caso di NNF, e così via.

4.2.7 Bus interno

Come mostrato in figura 4.3, lo universal-node include un bus dei messaggi interno implementato tramite DoubleDecker (DD). Anche se nell'immagine è mostrato solamente lo universal-node, il monitoring manager (vedi sezione 4.2.8) e le funzioni di monitoring sono connesse al bus. Anche le NF possono essere connesse a DD, ad esempio per ricevere allarmi dal monitoring manager o parametri di configurazione.

4.2.8 Monitoring Manager

Il *monitoring manager* ha l'incarico di gestire i moduli che:

1. misurano alcune metriche del servizio istanziato (ad esempio, CPU/memoria consumata dalle VNF);
2. generano allarmi quando si verifica uno specifico evento o quando viene superata una soglia.

Il monitoring manager può essere configurato così da misurare determinate metriche tramite una stringa, in grado di istruirlo, nello specifico NF-FG.

Dopo l'istanziamento del NF-FG, il monitoring manager istanzia e configura le corrette *monitoring functions (MFs)* che monitorano le metriche richieste e generano allarmi sul bus interno, come *Google cAdvisor* [20] e *Ramon* [21].

In aggiunta, il monitor manager oltre ad eseguire e configurare le corrette MF, riceve gli allarmi tramite DD, aggrega le informazioni ricevute come richiesto dall'istruzione presente nel NF-FG e propaga nuovamente gli eventi sul bus DD. In questo modo, le informazioni aggregate possono raggiungere le NF interessate. Ad esempio, una NF può sfruttare i risultati del monitoring per richiedere un aggiornamento del NF-FG, così da reagire all'evento.

4.2.9 GUI

La GUI facilita la vita agli utenti interessati ad usare lo universal-node senza conoscerne gli aspetti implementativi, fornendo un'interfaccia grafica *user-friendly* che evita l'utilizzo diretto delle REST API per comunicare con lo universal-node.

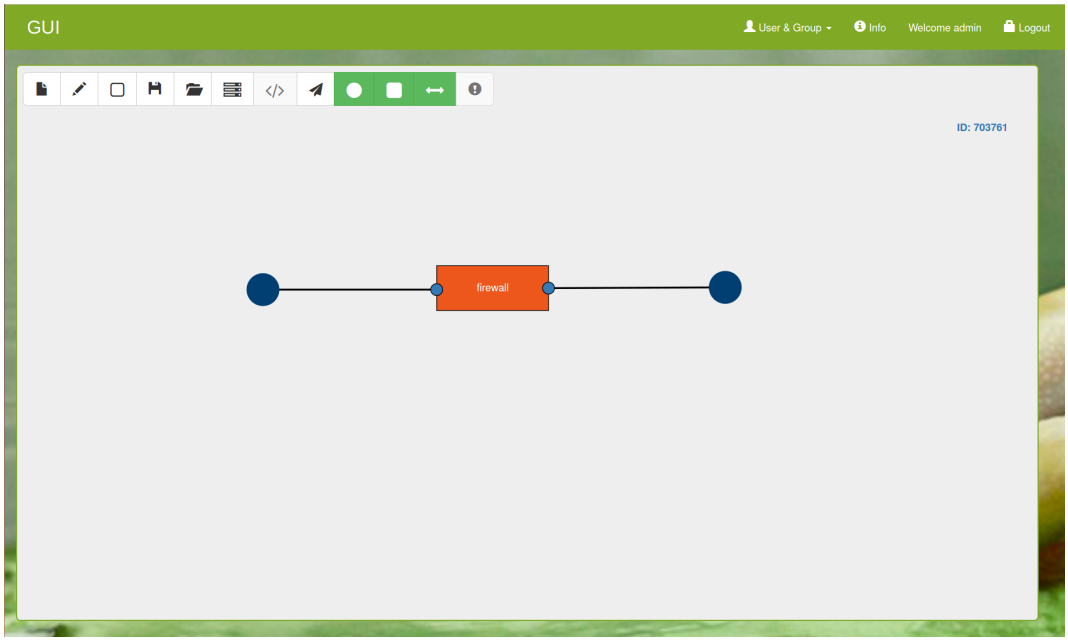

Figura 4.6. visualizzazione di un grafo tramite GUI.

Tramite GUI è possibile eseguire le seguenti operazioni:

- creare un nuovo documento: partendo da un grafo vuoto, opportuni pulsanti permettono la creazione e la caratterizzazione dei Service Access Point,

l'aggiunta di funzioni di rete e la creazione delle flow rule, che collegano gli endpoint e le VNF attraverso match e azioni specificate dall'utente;

- visualizzare i grafi istanziati: la GUI è in grado di interrogare lo universal-node e farsi ritornare la lista dei grafi presenti sul nodo. tramite una *select box* è possibile decidere quale grafo visualizzare;

- salvare/caricare un grafo: una volta costruito un grafo si può decidere di salvarlo su un file in formato json (con il formalismo NF-FG), per un futuro utilizzo o una possibile condivisione. Allo stesso modo è possibile caricare un grafo dall'esterno, per poterlo utilizzarne o apportarvi delle modifiche;

- istanziare un grafo: tramite un semplice click l'utente può decidere di istanziare il grafo costruito sullo universal-node o di modificarne uno preesistente, così da poter usufruire del nuovo servizio.

Un' immagine della GUI è mostrata in figura 4.6. Il grafo mostrato al centro, composto da due punti di accesso (in blu) collegati alle due porte della VNF "firewall" tramite due flow rule bidirezionali, può essere visto come un nuovo grafo pronto per essere istanziato o uno già esistente sul nodo.

A scelta dell'utente il grafo può essere visualizzato in una modalità speciale, che prende il nome di Big-Switch. Questa modalità dispone gli endpoint e le VNF al centro di un grosso switch, formando una *Big Switch Network*, in cui vengono espanse le regole che governano il grafo, così da rendere più chiaro il funzionamento generale.

Capitolo 5

Contributo della tesi corrente all'architettura

La grande maggioranza degli operatori di rete si appoggiano spesso su un collegamento PPPoE per il collegamento di utenti xDSL verso la centrale Telecom per trasportare il traffico da casa dell'utente. Questo permette al fornitore di mantenere un collegamento diretto punto-punto tra l'utente finale e il *BRAS (Broadband Remote Access Server)* che sta all'edge della rete dell'ISP e sul quale quest'ultimo mantiene *policy* e regole di *QoS*.

Questo capitolo vuole presentare e descrivere il contributo dato dalla presente tesi all'architettura dello universal-node per far in modo che sia in grado di implementare un home-gateway connesso al provider tramite una connessione PPPoE. In particolare si presentano le diverse soluzioni pensate, il perché della soluzione scelta e come è stata implementa.

5.1 Estensione per il supporto al PPPoE

Per poter fare in modo che lo universal-node sia utilizzabile come un home-gateway evoluto, il quale permette agli utenti di personalizzare i servizi di rete attivi sulla propria connessione, la sua architettura e i suoi moduli sono stati estesi in modo da supportare un collegamento alla rete del provider che utilizzi il protocollo PPPoE.

Il punto di partenza su cui si è concentrato il lavoro è stato quello di capire come poter creare un collegamento tra le strutture già esistenti ed utilizzate dallo universal-node e la nuova porta PPP necessaria per il funzionamento del protocollo che si vuole supportare, la quale è implementata come porta di livello L3 da un sistema Linux.

Come descritto nel capitolo precedente, lo universal-node utilizza software switch interni di tipo Open vSwitch, il quale lavora solamente con trame Ethernet e non permette quindi un collegamento diretto con una porta di livello L3 come quella PPP. Fin ad ora questo non aveva portato con se alcun problema in quanto il traffico verso Internet utilizzava porte di livello L2 che venivano direttamente agganciate allo switch OvS e grazie alle regole installate all'interno dello switch stesso, era possibile indirizzare correttamente i pacchetti e raggiungere Internet. Questo non è possibile utilizzando una connessione PPPoE, poiché utilizza invece delle porte di tipo PPP che Linux implementa come porte di livello L3 e attraverso le quali transitano pacchetti IP non dotati di intestazione Ethernet. Al fine di raggiungere l'obiettivo finale, è stato quindi necessario utilizzare una struttura di routing di supporto che permettesse di far si che i pacchetti dotati di header L2 in uscita da uno switch OvS venissero processati e inviati alla porta di tipo PPP. È stata dunque necessaria l'aggiunta di un modulo di routing per la gestione di tutto il traffico che deve essere rediretto verso tale tipo di porta.

Un semplice modulo che facesse il forwarding dei pacchetti in uscita dallo switch OvS verso la porta PPP non era sufficiente alla risoluzione del problema. Questo perché, come si può vedere dalla figura 5.1, il routing L3 richiedeva la gestione di due rotte di default: una per redirigere il traffico verso la porta PPP e quindi verso il nodo che termina il grafo di uscita necessario per raggiungere Internet e una per indirizzare il traffico verso il nodo di tipo Host-Stack per permettere di raggiungere lo stack TCP/IP della macchina che lo ospita e quindi i servizi applicativi in esecuzione sulla macchina stessa.

La prima soluzione pensata per poter risolvere il problema prevedeva l'uso del *Policy Based Routing (PBR)*, ovvero l'uso di più tabelle di routing configurate con opportune regole in modo da indirizzare il flusso dei pacchetti sul giusto percorso.

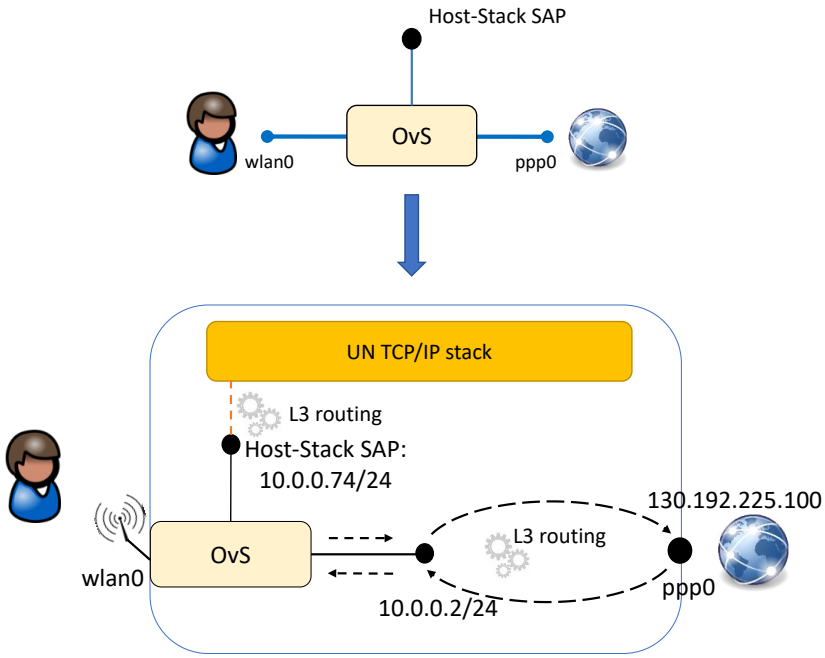

Figura 5.1. Creazione di collegamenti verso una porta PPP di livello L3.

Tuttavia, tale soluzione (dettagliata nell'appendice A) è stata scartata in quanto risultava particolarmente complessa e articolata. Inoltre, nonostante l'uso di quattro tabelle di routing, si presentavano ancora problemi per cui i pacchetti non seguivano il percorso desiderato.

La soluzione finale prevede l'uso di un network namespace *helper* di supporto. Esso permette di gestire, in maniera più semplice rispetto la precedente soluzione scartata, le due rotte di default necessarie all'interno della macchina che ospita lo universal-node. Tale namespace viene creato all'avvio dello universal-node se la configurazione di boot specifica la presenza di una porta di livello L3. Inizialmente, questa soluzione non era stata presa in considerazione in quanto richiede che la porta PPP venga inserita all'interno del namespace, con conseguente perdita della configurazione di un endpoint del tunnel PPP. Infatti, in questo modo, è necessario riconfigurare la porta in modo tale che il tunnel PPP, precedentemente creato durante la negoziazione tra la macchina che ospita lo universal-node e il server ISP, rimanga attivo affinché la comunicazione tramite protocollo PPPoE sia ancora in

grado di funzionare.

Per la gestione di questo nuovo componente, è stato esteso il *Network Manager* in modo tale da renderlo idoneo alla gestione delle porte di livello L3.

5.1.1 Componente di routing per la gestione di porte di livello L3

Il componente aggiuntivo che il lavoro di questa tesi ha implementato per la gestione di collegamenti che interessano porte di livello L3, è il *L3 Port Manager*. Esso estende il modulo del *Network Manager* il quale, come descritto nei capitoli precedenti, si occupa della gestione della parte di rete del grafo che viene istanziato all'interno dello universal-node.

Il *L3 Port Manager* gestisce le funzionalità di routing che determinati servizi richiedono, supportando l'uso di porte di livello L3 che lo switch di tipo OvS non permette di gestire direttamente. Esso viene usato dal *Graph Manager* per istanziare la struttura del namespace *helper* e gestire il collegamento con le altre istanze di rete che compongono il servizio richiesto.

Al boot dello universal-node (1) in figura 5.2) vengono istanziati LSI-0 e, se viene richiesto un collegamento verso Internet che prevede l'uso di una tale porta, viene creato anche il namespace *helper*. Una porta di tipo L2 è direttamente connessa a LSI-0, mentre la porta PPP di tipo L3, viene incapsulata dal *L3 Port Manager* all'interno del namespace subito dopo essere stato creato; fatto ciò il *L3 Port Manager* si occupa anche di riconfigurarla in maniera opportuna per poter mantenere attivo il tunnel PPP già esistente.

Quando viene istanziato un grafo, il namespace è poi collegato tramite una porta al LSI-0 (2) in figura 5.2) e non ad uno specifico Graph-LSI, in quanto si tratta di una terminazione necessaria per tutti i grafi che intendono avere un collegamento verso la rete Internet. Successivamente vengono aggiunte le regole OpenFlow in accordo con quelle specificate all'interno del grafo di servizio.

In particolare ogni volta che viene istanziato un nuovo grafo che contiene un collegamento verso Internet tramite la porta di tipo PPP, viene creato un collegamento tra LSI-0 e namespace *helper* (3) in figura 5.2) seguito dalla configurazione

delle nuove regole per il flusso dei pacchetti.

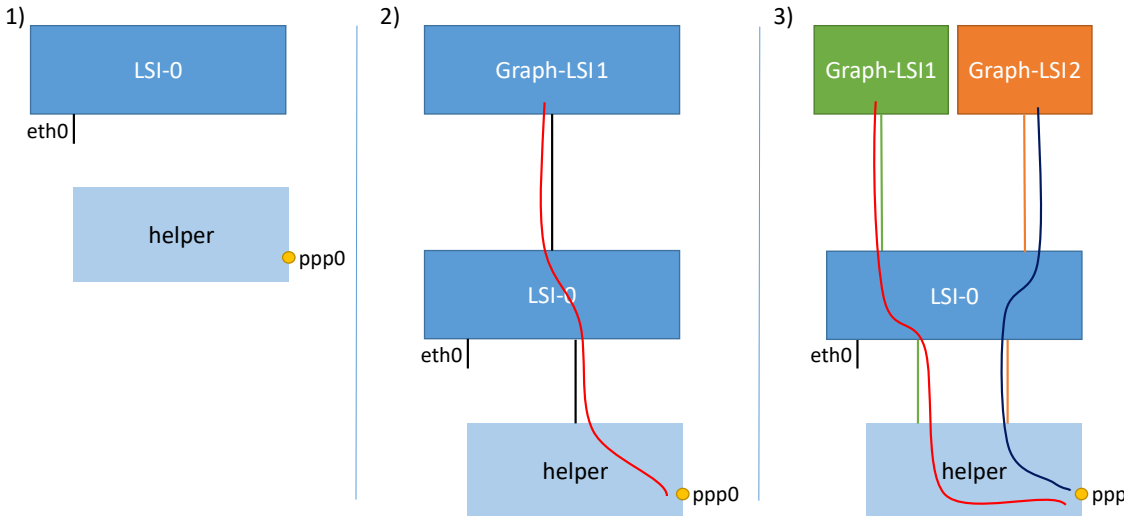

Figura 5.2. Creazione di collegamenti verso una porta PPP di livello L3.

Il formalismo NF-FG non è stato interessato da alcuna modifica in quanto la porta L3 può essere trattata come un nodo di tipo endpoint già supportato dallo universal-node. Tuttavia rimane comunque la necessità di distinguere se l'endpoint mappa una porta di livello L2 o una di livello L3. Di questo si occupa il software dello universal-node, sfruttando i paramenti di input specificati nel file di configurazione che lo universal-node legge al boot. Tale file è stato modificato aggiungendo un parametro opzionale che specifica il nome della porta di livello L3 che sarà sotto il controllo dello universal-node. Quindi, quando il sistema riceve la richiesta di un servizio tramite un grafo espresso con il formalismo NF-FG, lo universal-node sarà in grado di distinguere se l'endpoint specificato è di tipo L2 o L3 e comportarsi di conseguenza.

Il namespace, una volta istanziato dal *L3 Port Manager*, una funzione di router o router+NAT (a seconda del servizio richiesto). Mantiene infatti al suo interno delle regole di routing verso Internet (rotta di default) e verso i grafi, in particolare una regola per collegamento verso LSI-0, che equivale a dire una regola per ogni grafo che richiede un collegamento con la porta di livello L3. Inoltre è possibile configurare al suo interno anche la *chain netfilter* in modo ad esempio da manipolare i pacchetti

per rendere il modulo di routing un modulo che sia anche in grado di avere delle funzioni di NAT (caso comune di un home-gateway in cui il provider fornisce un solo indirizzo pubblico all'utente finale). Per far questo basta digitare il seguente comando:

```
sudo ip netns exec helper iptables -t nat -A POSTROUTING
   -o ppp0 -j MASQUERADE
```

La figura 5.3 mostra in dettaglio come un grafo che richiede una connessione attraverso una porta di livello L3 viene implementato dallo universal-node. In base a quello precedentemente descritto, quando viene avviato lo universal-node e letto il file di configurazione, la situazione è quella mostrata nella "fase 1" della figura 5.3: vengono istanziati LSI-0 e namespace *helper*, agganciate le porte di livello L2 (`wlan0`) a LSI-0 e quella di livello L3 (`ppp0`) viene inserita dentro il namespace. Viene inoltre configurata la routing table interna al namespace con la regola di default che sarà poi necessaria per l'indirizzamento dei pacchetti dal namespace dovranno essere diretti verso Internet. Successivamente ("fase2" in figura 5.3), quando viene istanziato il grafo richiesto, viene aggiunto un Graph-LSI relativo al nuovo grafo, al quale verranno collegati relativi endpoint e VFN richieste. Poiché è richiesto un collegamento verso Internet tramite una porta di tipo L3, allora lo universal-node crea anche un link che collega LSI-0 al namespace. Vengono inoltre aggiunge anche tutte le regole di traffic steering necessarie. A questo punto la nuova interfaccia interna al namespace dovrà essere configurata assegnandole in indirizzo IP; questo farà sì che sarà aggiunta una nuova regola all'interno della routing table del namespace stesso relativa al percorso verso il grafo istanziato.

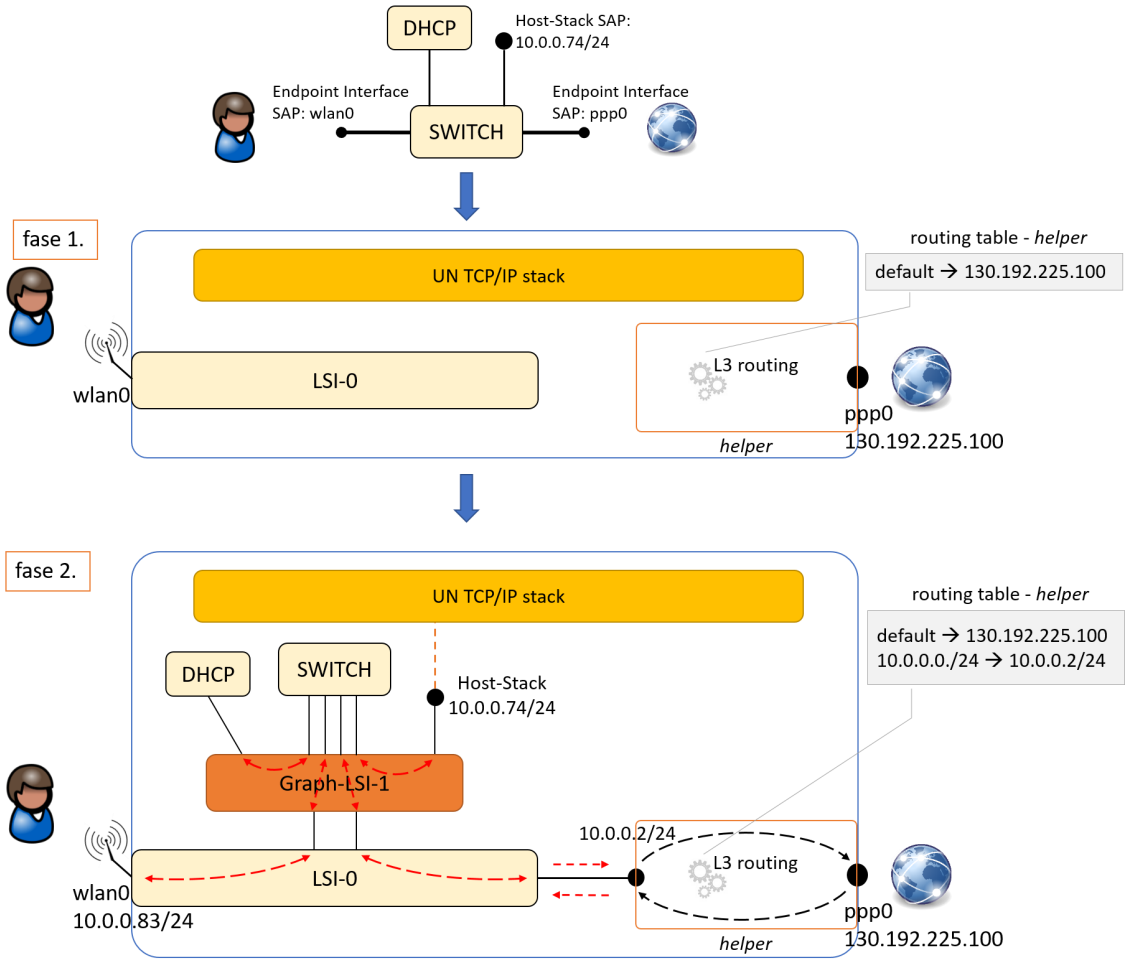

Figura 5.3. Implementazione da parte dello universal-node d un grafo che richiede un collegamento con una porta di livello L3.

Capitolo 6

Servizio di un home-gateway implementato tramite grafi

Le modifiche apportate sono giustificate dall'obiettivo finale del lavoro, cioè il voler ottenere un prototipo di home-gateway che utilizzi il software dello universal-node e fornisca dei servizi all'utente tramite l'utilizzo di grafi che si attestano sulla connessione PPPoE, largamente utilizzata sulle tecnologie xDSL fornite ad un utente.

Lo scenario di utilizzo che si è preso in considerazione è quello mostrato in figura 6.1 che mostra un CPE domestico installato a casa dell'utente e collegato alla rete del provider.

In tale scenario si suppone che sul CPE, e quindi sulla macchina che ospita il software dello universal-node, sia già configurato il PPPoE e dunque sia già attiva una tale connessione e il tunnel PPP verso la rete dell'operatore. Si vuole inoltre riprodurre lo scenario più comune in un'abitazione, ovvero più utenti (in questo caso due) connessi al CPE tramite un access point e quindi una connessione Wireless.

6.1 Bootstrapping

All'accensione del dispositivo vengono avviate alcune applicazioni necessarie al funzionamento, lo universal-node e la GUI.

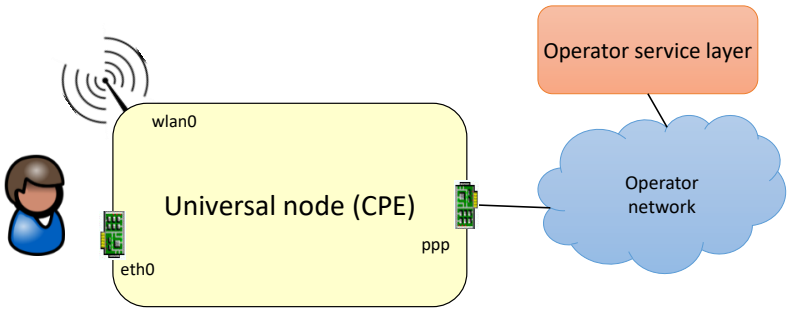

Figura 6.1. Home-gateway nella rete di un operatore

Al boot dello universal-node, vengono istanziati 5 grafi letti dal file di configurazione. Tali grafi inizializzano l'home-gateway permettendone la configurazione e l'utilizzo da parte degli utenti che voglio accedere alla rete. Nella figura 6.2 viene dettagliato lo scenario interno allo universal-node una volta completata la fase di boot e che i grafi vengano istanziati. Come si può vedere, inizialmente il servizio istanziato prevede che ogni utente abbia la possibilità di raggiungere Internet, ma anche l'endpoint di tipo Host-Stack che gli permette di accedere ai servizi applicativi e tramite questi poter personalizzare i grafi richiedendo i servizi di rete di cui vuole usufruire.

I grafi istanziati per comporre il servizio vengono analizzati in dettaglio nelle sezioni successive.

6.1.1 Grafo LAN

Ogni utente ha a sua disposizione un grafo di default sull'estremo LAN della rete che non può modificare e che viene mostrato in dettaglio dalla figura 6.3. Si tratta di un grafo semplice che collega l'interfaccia fisica `wlan0` del dispositivo ad un endpoint di tipo Internal tramite uno switch rappresentato dalla LAN in figura. Il suo scopo è quello di permettere l'interconnessione tra l'utente e il punto di accesso Internal che rappresenta il punto di contatto verso tutti i servizi che ha intenzione di richiedere al provider. Ogni volta che l'utente istanzierà un grafo questo verrà connesso ai punti di accesso Internal del grafo LAN e WAN (descritto dopo).

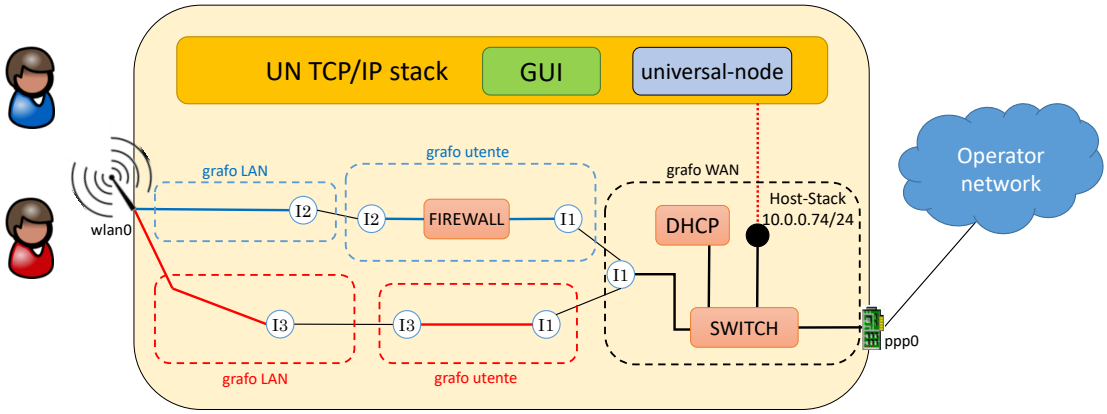

Figura 6.2. Architettura home-gateway gestito con i grafi

Il listing 6.1 descrive il grafo rappresentato in figura 6.3 utilizzando il formalismo NF-FG; come si può vedere è presente una sola regola bidirezionale che lega i due nodi di rete specificando però nella regola di match il MAC address sorgente del dispositivo per cui è stato istanziato quel graf, in quanto, come specificato in precedenza in tale scenario, ogni utente ha un proprio grafo di ingresso sul CPE così da poter distinguere il suo traffico da quello di un altro utente già all'ingresso del nodo.

Listing 6.1. Home-Gateway: Grafo lato LAN

```
{
  "forwarding-graph": {
    "name": "default-lan-graph1",
    "end-points": [
      {
        "id": "internal_lan",
        "name": "port_int",
        "type": "internal",
        "internal": {
          "internal-group": "lan1"
        }
      },
      {
        "id": "wlan-if",
        "name": "wlan-if",
        "type": "interface",
        "interface": {
          "if-name": "wlan0"
        }
      },
      {
        "id": "ethlan-if",
        "name": "ethlan-if",
        "type": "interface",
        "interface": {
          "if-name": "eth0"
        }
      }
    ],
    "VNFs": [
      {
        "id": "switch",
        "name": "switch",
        "functional-capability": "",
        "vnf_template": "8FI5AA",
        "ports": [
          {
            "id": "L2Port:0",
            "name": "data-port"
          },
          {
```

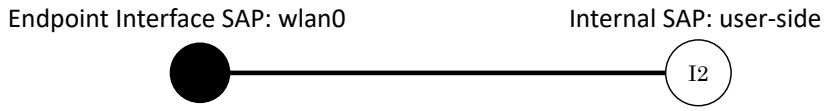

Figura 6.3. Home-gateway - grafo LAN

```
            "id": "L2Port:1",
            "name": "data-port"
          },
          {
            "id": "L2Port:2",
            "name": "data-port"
          }
        ]
      }
    ],
    "big-switch": {
      "flow-rules": [
        {
          "id": "1",
          "priority": 1,
          "match": {
            "port_in": "endpoint:
                internal_lan"
          },
          "actions": [
            {
              "output_to_port": "vnf:
                  switch:L2Port:0"
            }
          ]
        },
        {
          "id": "2",
          "priority": 1,
          "match": {
            "port_in": "vnf:switch:L2
                Port:0"
          },
          "actions": [
            {
              "output_to_port": "
                  endpoint:
                  internal_lan"
            }
          ]
        },
        {
          "id": "3",
          "priority": 1,
          "match": {
            "port_in": "endpoint:
                ethlan-if"
          },
          "actions": [
            {
              "output_to_port": "vnf:
                  switch:L2Port:1"
            }
          ]
        },
        {
          "id": "4",
          "priority": 1,
          "match": {
            "port_in": "vnf:switch:L2
                Port:1"
          },
          "actions": [
            {
              "output_to_port": "
                  endpoint:ethlan-if"
            }
          ]
        },
        {
          "id": "5",
          "priority": 1,
          "match": {
            "port_in": "endpoint:wlan
                -if"
          },
          "actions": [
            {
              "output_to_port": "vnf:
                  switch:L2Port:2"
            }
          ]
        },
        {
          "id": "6",
          "priority": 1,
```

```
117             "match": {
118               "port_in": "vnf:switch:L2
                      Port:2"
119             },
120             "actions": [
121               {
122                 "output_to_port": "
                        endpoint:wlan-if"
123               }
124             ]
125           }
126         ]
127       }
128     }
129   }
```

6.1.2 Grafo WAN

Il grafo WAN, mostrato in figura 6.4, è unico in quanto contiene i servizi comuni messi a disposizione per tutti gli utenti ed è quello che mette effettivamente in collegamento il CPE con la rete dell'operatore. Anche questo grafo, come quello LAN, non può essere modificato dall'utente.

Esso presenta quelle caratteristiche e quei servizi che devono essere comuni a tutti gli utenti. È caratterizzato da un punto di accesso di tipo internal che rappresenta il terminatore di tutti i grafi utente, e da un collegamento con il DHCP server necessario per permettere agli utenti di ottenere un indirizzo IP. È presente anche un accesso allo stack TCP/IP e quindi ai servizi applicativi che ospita il dispositivo tramite un endpoint di tipo Host-Stack. Quest'ultimo viene configurato in maniera statica e permette all'utente di configurare il proprio home-gateway, cambiare password di accesso all'interfaccia wireless, visualizzare i grafi presenti all'interno del CPE, modificarli e crearne di nuovi utilizzando la GUI. Infine si ha anche il collegamento verso il endpoint di uscita, che in questo scenario è rappresentato dalla porta di tipo PPP. Lo scenario che si sta analizzando prevede anche una funzione di NAT che però non è visibile in figura 6.2. La funzione è richiesta in quanto essendo un home-gateway, gli utenti avranno una propria rete privata e dovranno poi interfacciarsi verso Internet con un unico indirizzi IP pubblico assegnato dal fornitore di servizio. Poiché è presente un collegamento verso Internet tramite una porta di livello L3, il collegamento verso la porta `ppp0` viene tradotto dal software dello lo universal-node in un modulo che funge da router o route+NAT. In questo caso, essendo lo scenario in questione un comune esempio di accesso ad un home-gateway e che necessita quindi delle funzioni di NAT per far si che l'utente possa accedere alla rete Internet, la struttura che gestisce tale nodo dovrà contenere anche le regole necessarie per effettuare il natting.

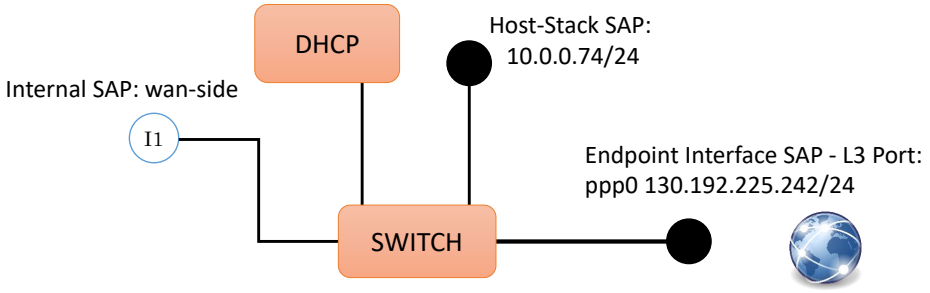

Figura 6.4. Home-gateway - grafo WAN

Il listing 6.2 descrive il grafo WAN con il formalismo NF-FG. Anche in questo caso le regole di traffic steering specificate sono delle regole bidirezionali necessarie per permettere a tutti i componenti del grafo di essere interconnessi.

Listing 6.2. Home-Gateway: Grafo lato WAN

```
{
  "forwarding-graph": {
    "name": "default-wan-graph",
    "end-points": [
      {
        "id": "wan-if",
        "name": "wan-if",
        "type": "interface",
        "interface": {
          "if-name": "ppp0"
        }
      },
      {
        "id": "internal_out",
        "name": "egress",
        "type": "internal",
        "internal": {
          "internal-group": "egress"
        }
      },
      {
        "id": "1",
        "name": "host-stack",
        "type": "host-stack",
        "host-stack": {
          "configuration": "STATIC",
          "IPv4": "10.0.0.74/24"
        }
      }
    ],
    "VNFs": [
      {
        "id": "dhcp",
        "name": "dhcp",
        "functional-capability": "",
        "vnf_template": "23C8AT",
        "ports": [
          {
            "id": "L2Port:1",
            "name": "data-port"
          }
        ]
      },
      {
        "id": "switch",
        "name": "switch",
        "functional-capability": "",
        "vnf_template": "8FI5AA",
        "ports": [
          {
            "id": "L2Port:0",
            "name": "data-port"
          },
          {
            "id": "L2Port:1",
            "name": "data-port"
          },
          {
            "id": "L2Port:2",
            "name": "data-port"
          },
          {
            "id": "L2Port:3",
            "name": "data-port"
          }
        ]
      }
    ],
    "big-switch": {
      "flow-rules": [
        {
          "id": "1",
          "priority": 1,
          "match": {
            "port_in": "vnf:dhcp:L2
                Port:1"
          },
```

```
            "actions": [
              {
                "output_to_port": "vnf:switch:L2Port:0"
              }
            ]
          },
          {
            "priority": 1,
            "match": {
              "port_in": "vnf:switch:L2Port:0"
            },
            "id": "2",
            "actions": [
              {
                "output_to_port": "vnf:dhcp:L2Port:1"
              }
            ]
          },
          {
            "priority": 1,
            "match": {
              "port_in": "vnf:switch:L2Port:1"
            },
            "id": "3",
            "actions": [
              {
                "output_to_port": "endpoint:internal_out"
              }
            ]
          },
          {
            "priority": 1,
            "match": {
              "port_in": "endpoint:internal_out"
            },
            "id": "4",
            "actions": [
              {
                "output_to_port": "vnf:switch:L2Port:1"
              }
            ]
          },
          {
            "priority": 1,
            "match": {
              "port_in": "vnf:switch:L2Port:2"
            },
            "id": "5",
            "actions": [
              {
                "output_to_port": "endpoint:wan-if"
              }
            ]
          },
          {
            "priority": 1,
            "match": {
              "port_in": "endpoint:wan-if"
            },
            "id": "6",
            "actions": [
              {
                "output_to_port": "vnf:switch:L2Port:2"
              }
            ]
          },
          {
            "id": "7",
            "priority": 1,
            "match": {
              "port_in": "vnf:switch:L2Port:3"
            },
            "actions": [
              {
                "output_to_port": "endpoint:000001"
              }
            ]
          },
          {
            "id": "000008",
            "priority": 1,
            "match": {
              "port_in": "endpoint:000001"
            },
            "actions": [
              {
                "output_to_port": "vnf:000002:L2Port:3"
              }
            ]
          }
        ]
      }
    }
}
```

6.1.3 Grafo utente

Il servizio prevede inoltre, per ogni utente, un grafo intermedio tra quello lato LAN e quello lato WAN. Nello scenario mostrato in 6.2 i grafi utente rappresentati sono diversi tra un utente e l'altro in modo da evidenziare il fatto che ogni utente ha la possibilità di istaziare il servizio che preferisce indipendentemente dagli altri

utenti. Il grafo del primo utente, mostrato in figura, 6.5, è costituito da una VNF che implementa un firewall collegata tramite due endpoint Internal al rispettivo grafo LAN e a quello lato WAN. Il grafo del secondo utente, come si vede in figura 6.6, è costituito solamente da due endpoint connessi tra di loro che a loro volta saranno collegati rispettivamente uno lato LAN e uno lato WAN. Come si può vedere in entrambi i grafi, l'endpoint Internal lato LAN ha lo stesso *internal-group* dell'endpoint Internal presente nel grafo LAN; allo stesso modo, l'endpoint Internal lato WAN ha lo stesso *internal-group* dell'endpoint Internal presente sul grafo WAN.

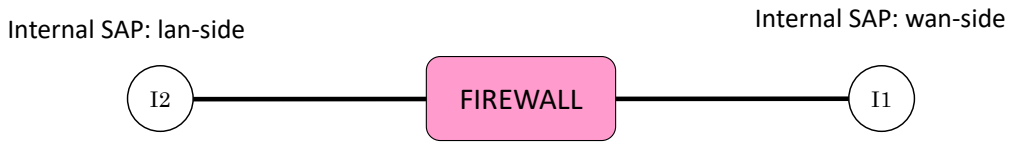

Figura 6.5. Home-gateway - grafo utente con VNF firewall

Figura 6.6. Home-gateway - grafo utente senza VNF

La scelta di interporre un ulteriore grafo intermedio tra i due grafi LAN e WAN descritti sopra è giustificata dal voler far si che ogni utente abbia due grafi di default, uno che permette di introdurre il suo traffico all'interno del CPE non modificabile

e legato al MAC address che l'utente sta usando e l'altro per farlo uscire verso Internet, e infine un ulteriore grafo messo a sua disposizione che può manipolare e modificare per poter aggiungere i servizi che meglio soddisfano le sue esigenze.

Grazie all'interazione con la GUI, alla quale è possibile accedere tramite il endpoint di tipo Host-Stack, l'utente è in grado di modificare e personalizzare il proprio grafo utente con la configurazione che preferisce.

I listing 6.3 e 6.4 descrivono i grafi utente mostrati in figura 6.2 secondo il formalismo NF-FG.

Listing 6.3. Home-Gateway: Grafo utente con VFN firewall

```
{
  "forwarding-graph": {
    "name": "default-user-graph1",
    "end-points": [
      {
        "id": "internal_lan",
        "name": "port_int",
        "type": "internal",
        "internal": {
          "internal-group": "lan1"
        }
      },
      {
        "id": "internal_wan",
        "name": "port_int",
        "type": "internal",
        "internal": {
          "internal-group": "wan"
        }
      }
    ],
    "VNFs": [
      {
        "id": "firewall",
        "name": "firewall",
        "functional-capability": "",
        "vnf_template": "90ZGH0",
        "ports": [
          {
            "id": "User:0",
            "name": "data-port"
          },
          {
            "id": "WAN:1",
            "name": "data-port"
          }
        ]
      }
    ],
    "big-switch": {
      "flow-rules": [
        {
          "id": "1",
          "priority": 1,
          "match": {
            "port_in": "endpoint:
                internal_lan"
          },
          "actions": [
            {
              "output_to_port": "vnf:
                  firewall:User:0"
            }
          ]
        },
        {
          "id": "2",
          "priority": 1,
          "match": {
            "port_in": "vnf:firewall:
                User:0"
          },
          "actions": [
            {
              "output_to_port": "
                  endpoint:
                  internal_lan"
            }
          ]
        },
        {
          "id": "3",
          "priority": 1,
          "match": {
            "port_in": "endpoint:
                internal_wan"
          },
          "actions": [
            {
              "output_to_port": "vnf:
                  firewall:WAN:1"
            }
          ]
        },
        {
          "id": "4",
          "priority": 1,
          "match": {
            "port_in": "vnf:firewall:
                WAN:1"
          },
          "actions": [
            {
              "output_to_port": "
                  endpoint:
                  internal_wan"
            }
          ]
        }
      ]
```

```
      }
    }
```

Listing 6.4. Home-Gateway: Grafo utente con nessuna VNF

```
{
  "forwarding-graph": {
    "name": "default-user-graph1",
    "end-points": [
      {
        "id": "internal_lan",
        "name": "port_int",
        "type": "internal",
        "internal": {
          "internal-group": "lan2"
        }
      },
      {
        "id": "internal_wan",
        "name": "port_int",
        "type": "internal",
        "internal": {
          "internal-group": "wan"
        }
      }
    ],
    "big-switch": {
      "flow-rules": [
        {
          "id": "1",
          "priority": 1,
          "match": {
            "port_in": "endpoint:
                internal_lan"
          },
          "actions": [
            {
              "output_to_port": "
                  endpoint:
                  internal_wan"
            }
          ]
        },
        {
          "id": "2",
          "priority": 1,
          "match": {
            "port_in": "endpoint:
                internal_wan"
          },
          "actions": [
            {
              "output_to_port": "
                  endpoint:
                  internal_lan"
            }
          ]
        }
      ]
    }
  }
}
```

Capitolo 7

Validazione

Questo capitolo riporta alcuni test effettuati per misurare thoughput, latenze e tempi di boot dello universal-node e dei grafi di default scelti per il caso d'uso dell'home-gateway come descritto sopra.

In particolare vengono analizzati sia il caso in cui siano istanziati i grafi per un singolo utente, sia quello in cui vengano istanziati per due utenti. Questi scenari sono stati analizzati sia collegando gli utenti tramite Wi-Fi, sia nel caso in cui gli utenti siano connessi all'home-gateway tramite una connessione via cavo.

7.1 Misurazione di throughput e latenza

La figura 7.1 mostra il banco di prova utilizzato; le macchine coinvolte solo le seguenti:

- macchina Intel x86 i7 6700 dotato di un processore di 3,40 GHz sulla quale viene eseguito il software dello universal-node;
- macchina Intel x86 i5 34500 con processore a 2.80 GHz che ospita il server che fornisce il servizio PPPoE e sul quale si ha dunque la terminazione del tunnel PPP;
- due macchine Intel x86 i7 6500U dotate di un processore a 2.50 GHz che sono state utilizzate come client.

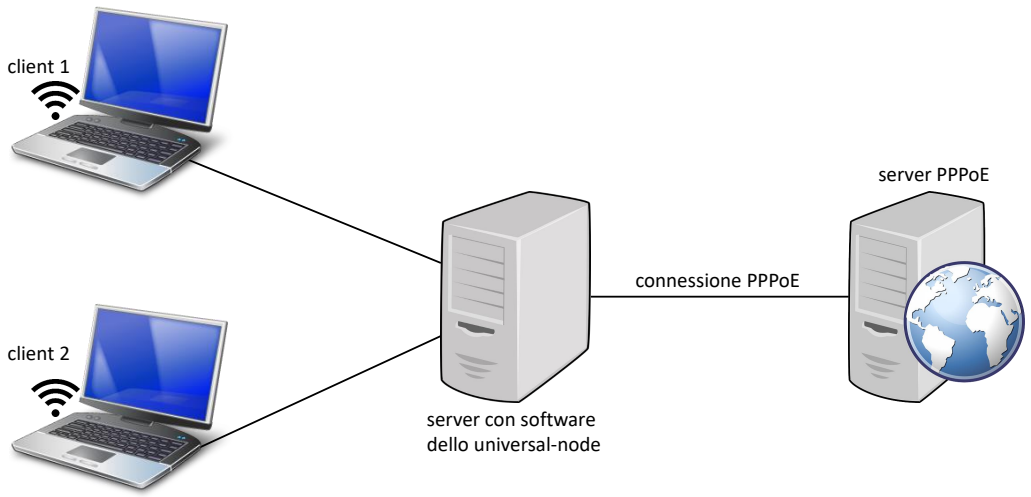

Figura 7.1. Rappresentazione del banco di prova.

Nel caso di connessione tramite Wi-Fi, i due client sono collegati alla macchina che ospita lo universal-node utilizzando un adattatore wireless, che supporta una tecnologia IEEE 802.11b/g, configurato come access point ai fini dei test.

La latenza è stata misurata analizzando i tempi di risposta in seguito all'invio di pacchetti ICMP dai client verso il server usando il comando `ping`, mentre per le misure del throughtput è stato effettuato un test sfruttando il comando `iperf`, il quale permette di misurare la banda utilizzata da uno stream di dati scambiato tra un client ed un server.

La tabella 7.1 mostra i risultati ottenuti durante la misurazione del throughput. I valori ottenuti nel caso di un collegamento wireless non sono molto performanti in quanto il dispositivo utilizzato come access point supporta una banda massima che si aggira intorno i 24 Mb/s. Per quanto riguarda invece la connessione via cavo, si hanno dei risultati superiori a quelli tradizionalmente offerti dai provider, sia nello scenario in cui è presente un solo utente sia nel caso in cui sono presenti due utenti. I risultati riportati nel caso in cui siano collegati due client, sono una media dei valori ottenuti per ogni singolo client.

La tabella 7.2 mostra invece i risultati ottenuti durante la misurazione della

	Throughput [Mbits/s]	
	1 client	**2 client**
Wi-Fi [802.11g]	16.29	5.69
Ethernet	436	425

Tabella 7.1. Misurazione del throughput

latenza. Anche in questo caso i valori ottenuti nel caso di una connessione wireless sono dovuti alle prestazioni non decisamente performanti del dispositivo utilizzato come access point, infatti nel caso di una connessione Ethernet si ottengono dei risultati decisamente più performanti. In particolare si può notare come nel caso di una connessione via cavo, a differenza di quanto ci si potrebbe aspettare, la latenza risulta minore nel caso in cui siano connessi due client piuttosto che uno solo. Questo probabilmente è dovuto al funzionamento interno del vSwitch che risulta più efficiente nel caso in cui sia attraversato da più traffico.

	Latenza [ms]	
	1 client	**2 client**
Wi-Fi [802.11g]	2.54	4.78
Ethernet	0.838	0.684

Tabella 7.2. Misurazione della latenza

7.2 Tempo di boot

Questo test intende misurare i tempi necessari al software dello universal-node per attivare il servizio di un home-gateway. Nello specifico sono stati misurati i tempi necessari per istanziare la nuova struttura utile per la comunicazione tramite protocollo PPPoE, quanto tempo richiedono i cinque grafi di default per essere istanziati e il tempo complessivo di boot dello universal-node. Un ultimo test è stato effettuato per verificare in quanto tempo lo universal-node apporta la modifica di un grafo utente in seguito alla richiesta, tramite l'interazione con la GUI, dell'aggiunta di un servizio di firewall all'interno del proprio grafo.

La figura 7.2 riassume i tempi misurati durante i test. Come si può notare il nuovo modulo di routing aggiunto all'architettura dello universal-node ha un

impatto praticamente trascurabile rispetto ai tempi di avvio dello universal-node. Anche i tempi di boot dei grafi di servizio dello scenario proposto sono soddisfacenti e rientrano nei tempi necessari ad un CPE domestico per avviarsi e fornire servizio all'utente.

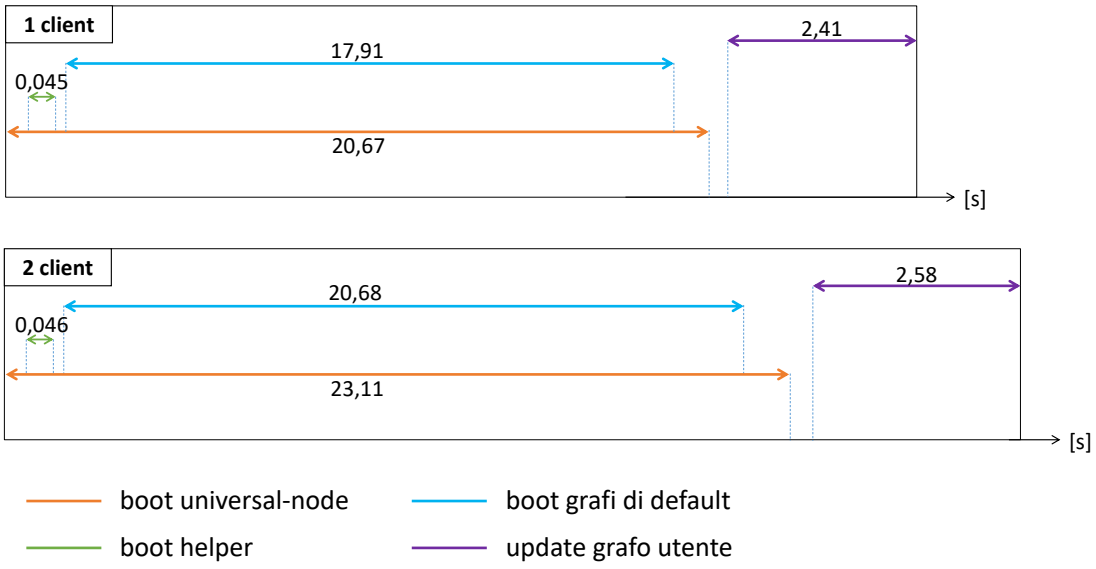

Figura 7.2. Misurazione dei tempi di boot dello universal-node configurato come home-gateway connesso al provider tramite protocollo PPPoE.

Capitolo 8

Conclusioni

Questa tesi ha utilizzato lo UN come un home-gateway il cui comportamento e' definito mediante grafi di servizio che si connettano alla rete del provider tramite una connessione PPPoE.

Estendendo il software dello universal-node e definendo opportuni grafi è stato possibile portare a termine l'obiettivo della tesi, cioè creare dei servizi virtualizzati all'interno di un prototipo di home-gateway, utilizzando NFV, SDN e una connessione PPPoE.

A partire dal lavoro svolto, un importate punto da cui partire per gli sviluppi futuri può essere quello di far si che il software dello universal-node con le modifiche apportate sia in grado di essere eseguito all'interno di hardware utilizzato per i CPE.

Altro spunto per lavori futuri potrebbe essere quello di implementare in maniera robusta la sicurezza e la gestione dei permessi che gli utenti hanno sui grafi istanziati. Ad esempio il grafo LAN e quello WAN non dovrebbero poter essere modificati dall'utente, il quale invece dovrebbe essere autorizzato esclusivamente alla modifica del proprio grafo utente sul quale può aggiungere e modificare i servizi.

A tal proposito, il servizio istanziato con i grafi scelti durante questa tesi ha un unico problema nel caso in cui un utente intenda istanziare nel suo grafo dei servizi di livello L3. Questo infatti implicherebbe una divisione della rete dell'utente e potrebbe quindi ad esempio essere necessario un ulteriore DHCP server anche sul grafo LAN o su quello dell'utente. Un'idea per poter andare incontro a questa necessità potrebbe essere quella di avere un modulo che sia in grado di capire se il

servizio richiesto sia una funzione di livello L3 e arricchire quindi il grafo con altri componenti che risultano necessari per mantenere il corretto funzionamento.

Appendice A

Soluzione con Policy Based Routing

Questa sezione presenta la soluzione che utilizza il *Policy Based Routing (PBR)* e che era stata pensata per implementare il modulo di routing che gestisse il traffico verso una porta di livello L3 all'interno dello universal-node. Questa soluzione è stata tuttavia scartata in quanto troppo articolata e poiché porta con se dei problemi legati al traffico di pacchetti applicativi.

A.1 Routing in Linux e *Policy Based Routing*

Il kernel Linux supporta la presenza di più tabelle di routing [22]. Oltre la tabella comunemente conosciuta e manipolata per gestire le rotte (chiamata `main` table), il kernel mantiene automaticamente altre due tabelle, la tabella `local` e la tabella `default`. La `local` è una tabella speciale e che non dovrebbe essere modificata in condizioni normali. Essa ha priorità 0, ovvero maggiore rispetto tutte le altre tabelle di routing. Tale tabella viene e deve essere consultata prima delle altre in quanto contiene informazioni sulle rotte presenti sull'host e per cui non è necessario inoltrare i pacchetti al di fuori della macchina. La `default` è invece una tabella vuota e riservata per il processamento di pacchetti che non hanno trovato la rotta neanche all'interno della tabella `main`, che ha priorità intermedia tra la `local` e la `default`. Oltre queste tabelle, il kernel Linux supporta fino a 252 tabelle di routing

e quindi la possibilità di gestire il routing utilizzando delle policy tramite il Routing Policy Database (RPDB) [23]. Quest'ultimo permette di dare delle regole in modo da istruire il kernel sul modo in cui deve ispezionare le tabelle di routing. Tramite il comando `ip rule` si possono infatti aggiungere delle regole per indicare le regole che bisogna matchare per ispezionare una determinata tabella, in particolare si posso mettere delle regole in base alla input o alla output interface oppure in base all'indirizzo sorgente o destinatario del pacchetto che deve essere processato; si può anche aggiungere, tramite l'opzione `pref`, la priorità che si vuole dare alla tabella. Oltre le regole che indicano al kernel di ispezionare una determinata tabella nel caso ci sia un match di una regola `ip rule`, potrebbero anche trovarsi regole di NAT, regole per cui un match deve provocare la generazione da parte del kernel di una ICMP unreachable o una ICMP prohibited oppure regole che indicano al kernel che un determinato pacchetto deve essere scartato.

Quando un nuovo pacchetto arriva, il kernel inizia a controllare la regola 0 (sarà sempre l'ispezione della tabella `local` essendo quella a priorità più alta) e itera su tutte le regole fin quando il pacchetto che necessita una regola di routing non matcha una regola. Fatto ciò, il kernel segue le istruzioni di tale regola, se viene trovato un match sulle regole di routing, il kernel usa tale regola, altrimenti ritorna nuovamente a ispezionare le regole date dal RPDB fino a quando ce ne sono disponibili.

A.2 Configurazione del PBR

Il PBR si presentava come la soluzione alla necessità di gestire due rotte di default (figura 5.1), una verso l'endpoint di tipo Host-Stack necessario per raggiungere lo stack TCP/IP della macchina che ospita lo universal-node e una verso la porta PPP e quindi verso il nodo che termina il grafo di uscita necessario per raggiungere Internet. Tuttavia, è stato necessario isolare l'endpoint (quello collegato a OvS in figura 5.1) dal quale i pacchetti devono essere inviati verso la porta `ppp0`. Questo è dovuto al fatto che tale endpoint appartiene alla stessa rete IP dell'endopoint Host-Stack quindi si creano disambiguità quando un pacchetto deve essere inviato dalla porta `ppp0` alla rete `10.0.0.0/24`. Il nuovo scenario diventa quindi quello mostrato

dalla figura A.1 in cui viene aggiunto un network namespace (*helper*) attraverso il quale dovranno fluire i pacchetti che dallo switch OvS devono giungere alla porta ppp0.

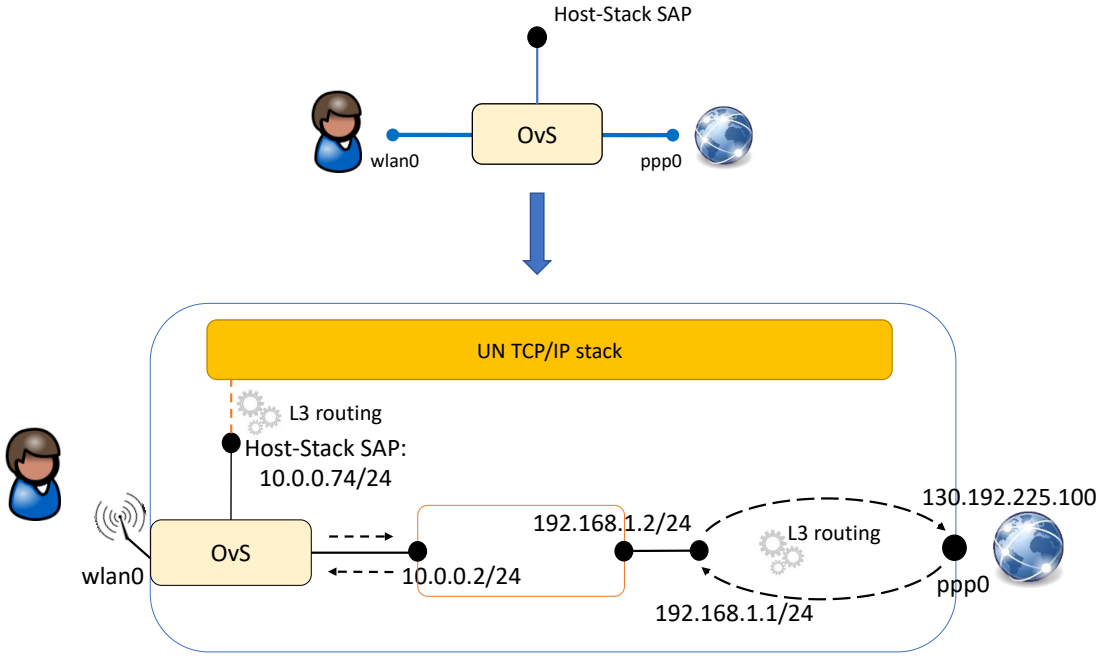

Figura A.1. Creazione di collegamenti verso una porta PPP di livello L3 con PBR.

Per emulare lo scenario in figura A.1 sono state utilizzate due macchine virtuali collegate tramite una connessione PPPoE, una che ospita il server e l'altra che funge da client ed emula la macchina sulla quale eseguire lo universal-node.

Di seguito sono riportati i passi e i comandi necessari per la configurazione della macchina client come descrive la figura A.2. In particolare per emulare il client collegato allo universal-node è stato utilizzato un namespace "client" collegato a OvS.

La prima cosa da fare è abilitare il forwarding per fare in modo che i pacchetti riescano a fluire tra veth7 e ppp0:

```
sudo sysctl -w net.ipv4.ip_forward=1
```

Creazione, configurazione dei due namespace e assegnazione degli indirizzi IP:

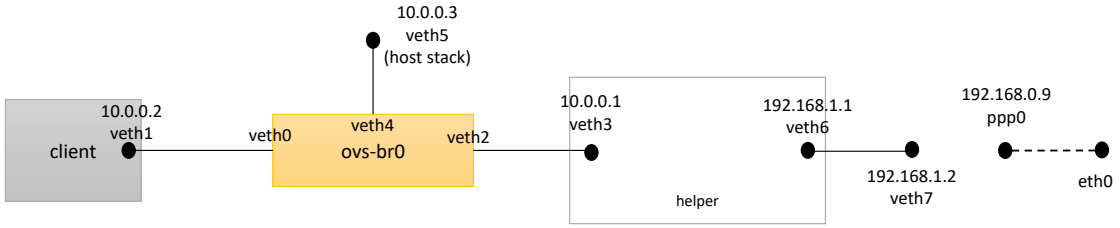

Figura A.2. Creazione di collegamenti verso una porta PPP di livello L3 con PBR.

```
sudo ip netns add client
sudo ip netns add helper
sudo ip link add veth0 type veth peer name veth1
sudo ip link add veth2 type veth peer name veth3
sudo ip link add veth6 type veth peer name veth7
sudo ip link set veth1 netns client
sudo ip link set veth3 netns helper
sudo ip link set veth6 netns helper
sudo ip netns exec client ifconfig veth1 up
sudo ip netns exec client ifconfig veth1 10.0.0.2/24
sudo ifconfig veth0 up
sudo ip netns exec helper ifconfig veth3 up
sudo ip netns exec helper ifconfig veth3 10.0.0.1/24
sudo ip netns exec helper ifconfig veth6 up
sudo ip netns exec helper ifconfig veth6 192.168.1.1/24
sudo ifconfig veth2 up
sudo ifconfig veth7 up
sudo ifconfig veth7 192.168.1.2/24
sudo ip netns exec client ip route add default via 10.0.0
   .1
sudo ip netns exec helper ip route add default via
   192.168.1.2
```

Creazione e configurazione dell'endpoint hoststack e dello switch OvS:

```
sudo ip link add veth4 type veth peer name veth5
sudo ifconfig veth4 up
sudo ifconfig veth5 up
sudo ifconfig veth5 10.0.0.3/24
sudo ovs-vsctl add-br ovs-br0
sudo ifconfig ovs-br0 up
sudo ovs-vsctl add-port ovs-br0 veth0
sudo ovs-vsctl add-port ovs-br0 veth2
sudo ovs-vsctl add-port ovs-br0 veth4
```

Creazione e configurazione delle tabelle di routing:

```
sudo echo 2 T1 >> /etc/iproute2/rt\_tables
sudo echo 2 T1 >> /etc/iproute2/rt\_tables
sudo echo 4 T3 >> /etc/iproute2/rt\_tables
sudo echo 5 T4 >> /etc/iproute2/rt\_tables
sudo ip route flush table main
sudo ip route add 192.168.1.0/24 dev veth7 table T1
sudo ip route add default via 192.168.0.9 dev ppp0 table
   T1
sudo ip route add 192.168.0.254 dev ppp0 table T1
sudo ip route add 10.0.0.0/24 via 192.168.1.1 dev veth7
   table T1
sudo ip route add 192.168.1.0/24 dev veth7 table T2
sudo ip route add default via 192.168.0.9 dev ppp0 table
   T2
sudo ip route add 192.168.0.254 dev ppp0 table T2
sudo ip route add 10.0.0.0/24 via 192.168.1.1 dev veth7
   table T2
sudo ip route add 10.0.0.0/24 dev veth5 table T3
sudo ip route add default via 10.0.0.1 dev veth5 table T3
```

```
sudo ip route del broadcast 10.0.0.0 dev veth5 table
   local
sudo ip route del broadcast 10.0.0.255 dev veth5 table
   local
sudo ip route del local 10.0.0.3 dev veth5 table local
sudo ip route add local 10.0.0.3 dev veth5 table T4
sudo ip route add broadcast 10.0.0.255 dev veth5 table T4
sudo ip route add broadcast 10.0.0.0 dev veth5 table T4
```

Configurazione delle regole necessarie al kernel per la scelta della tabella di routing da consultare:

```
sudo ip rule add from 192.168.1.2 table T2 pref 2997
sudo ip rule add from 192.168.0.9 table T2 pref 2998
sudo ip rule add iif lo table T3 pref 2999
sudo ip rule add iif ppp0 table T1 pref 3000
sudo ip rule add iif veth7 table T1 pref 3001
sudo ip rule table T4 pref 4000
```

A questo punto lo stato delle tabelle e quello delle regole necessarie al kernel per sapere in base a quali criteri deve ispezionare una routing table piuttosto che un'altra, è quello mostrato rispettivamente nelle figure A.3 e A.4.

Per far si che il server riesca a rispondere ai pacchetti che arrivano dal client e hanno un indirizzo IP sorgente appartenente alla rete privata interna alla macchina (`10.0.0.0/24`) bisogna aggiungere una regola all'interno della tabella `main` del server; questo è necessario in quanto non è stato utilizzato nessun modulo che implementa funzioni di NAT.

```
sudo ip route add 10.0.0.0/24 dev ppp0
```

```
pppoe@pppoe:~$ ip route show table local
broadcast 127.0.0.0 dev lo  proto kernel  scope link  src 127.0.0.1
local 127.0.0.0/8 dev lo  proto kernel  scope host  src 127.0.0.1
local 127.0.0.1 dev lo  proto kernel  scope host  src 127.0.0.1
broadcast 127.255.255.255 dev lo  proto kernel  scope link  src 127.0.0.1
local 192.168.0.9 dev ppp0  proto kernel  scope host  src 192.168.0.9
broadcast 192.168.1.0 dev veth7  proto kernel  scope link  src 192.168.1.2
local 192.168.1.2 dev veth7  proto kernel  scope host  src 192.168.1.2
broadcast 192.168.1.255 dev veth7  proto kernel  scope link  src 192.168.1.2
pppoe@pppoe:~$ ip route show table T1
default via 192.168.0.9 dev ppp0
10.0.0.0/24 via 192.168.1.1 dev veth7
192.168.0.254 dev ppp0  scope link
192.168.1.0/24 dev veth7  scope link
pppoe@pppoe:~$ ip route show table T2
default via 192.168.0.9 dev ppp0
10.0.0.0/24 via 192.168.1.1 dev veth7
192.168.0.254 dev ppp0  scope link
192.168.1.0/24 dev veth7  scope link
pppoe@pppoe:~$ ip route show table T3
default via 10.0.0.1 dev veth5
10.0.0.0/24 dev veth5  scope link
pppoe@pppoe:~$ ip route show table T4
broadcast 10.0.0.0 dev veth5  proto kernel  scope link  src 10.0.0.3
local 10.0.0.3 dev veth5  proto kernel  scope host  src 10.0.0.3
broadcast 10.0.0.255 dev veth5  proto kernel  scope link  src 10.0.0.3
```

Figura A.3. Routing table.

```
pppoe@pppoe: $ ip rule list
0:      from all lookup local
2997:   from 192.168.0.9 lookup T2
2998:   from 192.168.1.2 lookup T2
2999:   from all iif lo lookup T3
3000:   from all iif ppp0 lookup T1
3001:   from all iif veth7 lookup T1
4000:   from all lookup T4
32766:  from all lookup main
32767:  from all lookup default
```

Figura A.4. Regole seguite dal kernel per scoprire quale routing table utilizzare.

In base a tali regole i pacchetti riescono a raggiungere la porta `ppp0` e il server, seguendo il flusso mostrato nella figura A.5.

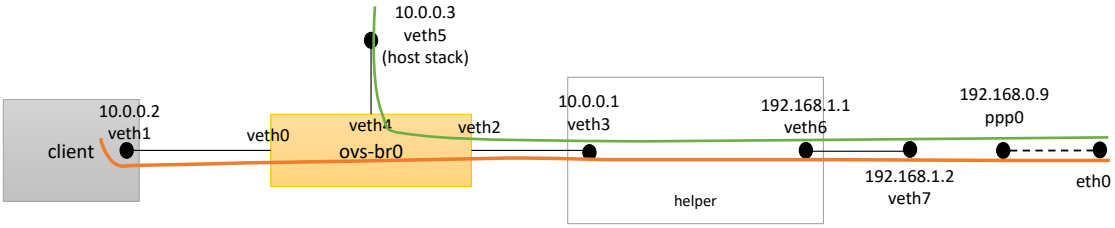

Figura A.5. Flusso dei pacchetti verso la porta PPP utilizzando le tabelle configurate grazie al PBR.

Tuttavia questa soluzione presenta delle problematiche legate al flusso di pacchetti applicativi, ovvero a differenza del traffico ICMP, il traffico TCP e UDP sembra non seguire il corretto percorso configurato tramite le tabelle quando giunto sulla porta `ppp0` deve essere inviato verso `veth7`. A causa delle lunghe tempistiche richieste per risolvere tale problema si è dunque optato per la soluzione presentata da questa tesi.

Bibliografia

[1] Brian Lavallée. “What is Distributed NFV and why do you need it?” In: (2016). URL: http://www.ciena.com/insights/articles/What-is-D-NFV-and-why-do-you-need-it.html.

[2] Z. Bronstein e E. Shraga. “NFV virtualisation of the home environment”. In: *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*. 2014, pp. 899–904.

[3] T. Cruz et al. “An architecture for virtualized home gateways”. In: *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. 2013, pp. 520 –526.

[4] N. Herbaut et al. “Migrating to a NFV-based Home Gateway: Introducing a Surrogate vNF approach”. In: *Network of the Future (NOF), 2015 6th International Conference on the*. 2015, pp. 1–7. DOI: 10.1109/NOF.2015.7333284.

[5] F. Sánchez e D. Brazewell. “Tethered Linux CPE for IP service delivery”. In: *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*. 2015, pp. 1–9.

[6] G. Faraci e G. Schembra. “An Analytical Model to Design and Manage a Green SDN/NFV CPE Node”. In: *IEEE Transactions on Network and Service Management* 12.3 (2015), pp. 435–450.

[7] Open Networking Foundation. “OpenFlow Switch Specification”. In: (2014).

[8] *Open vSwitch*. URL: http://openvswitch.org/.

[9] *Docker*. URL: https://docs.docker.com/engine/understanding-docker.

[10] *KVM*. URL: http://www.linux-kvm.org.

[11] Avi Kivity et al. "kvm: the Linux Virtual Machine Monitor". In: (2007), pp. 225–230.

[12] L. Mamakos et. al. *A Method for Transmitting PPP Over Ethernet (PPPoE)*. RFC 2516. The Internet Society, feb. 1999, pp. 1–17. URL: https://tools.ietf.org/html/rfc2516.

[13] *Universal Node*. URL: https://github.com/netgroup-polito/un-orchestrator.

[14] *DoubleDecker*. URL: https://github.com/Acreo/DoubleDecker.

[15] Pieter Hintjens. *ZeroMQ: Messaging for Many Applications*. " O'Reilly Media, Inc.", 2013.

[16] *xDPd*. URL: http://www.xdpd.org.

[17] László Molnár et al. "Dataplane Specialization for High-performance OpenFlow Software Switching". In: *Proceedings of the 2016 ACM conference on SIGCOMM*. Florianopolis,Brazil, 2016.

[18] Pat Bosshart et al. "P4: Programming protocol-independent packet processors". In: *ACM SIGCOMM Computer Communication Review* 44.3 (2014), pp. 87–95.

[19] *DPDK*. en. 2015. URL: http://dpdk.org/.

[20] *Google cAdvisor*. URL: https://github.com/google/cadvisor.

[21] P. Kreuger e R. Steinert. "Scalable in-network rate monitoring". In: *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 2015, pp. 866–869. DOI: 10.1109/INM.2015.7140396.

[22] *Routing Tables*. URL: http://linux-ip.net/html/routing-tables.html.

[23] *Routing Policy Database (RPDB)*. URL: http://linux-ip.net/html/routing-rpdb.html.