SAPIENZA
UNIVERSITÀ DI ROMA

FACOLTÁ DI INGEGNERIA DELL'INFORMAZIONE,
INFORMATICA E STATISTICA

Corso di Laurea Magistrale in:
INGEGNERIA DELLE COMUNICAZIONI

Implementazione Open vSwitch delle funzionalità di forwarding del protocollo LISP in uno scenario Software Defined Network

Laureando:

David Lo Bascio

Relatore:
Prof. Marco Listanti

Correlatore:
Prof. Antonio Cianfrani

Anno Accademico 2013 – 2014

"L'unica gioia al mondo è cominciare.
È bello vivere perché vivere è cominciare, sempre, ad ogni istante."

Cesare Pavese

Ringraziamenti

L'attività di ricerca e sperimentazione che è alla base di questa tesi è stata per me occasione di estrema realizzazione, pur nella fatica del lavoro quotidiano. Non volendo sminuire l'edificazione che essa - come tutto il mio percorso accademico - ha rappresentato in termini di conoscenze e competenze nel campo delle telecomunicazioni e in particolare del networking, considero la crescita personale come la dimensione più significativa di questa esperienza. Desidero pertanto esprimere alcuni ringraziamenti.

Innanzitutto la mia riconoscenza va al prof. Marco Listanti, per l'esempio professionale e umano che ha saputo dare in questi anni e in questi mesi. Un corposo ringraziamento al prof. Antonio Cianfrani, che nello sviluppo di questa tesi ha dimostrato smisurata disponibilità, oltre a rappresentare una luminosa fonte di incoraggiamento e consigli.

Una particolare menzione per alcuni colleghi, che in maniera non prevedibile all'inizio di questo percorso, hanno contribuito con i loro suggerimenti preziosi al raggiungimento della meta: David Rossi (CNR) - con il quale auspico di condividere un futuro di lunghe collaborazioni professionali -, Calogero Lo Leggio (Teleunit SPA), Lorand Jakab (CISCO Systems), Diego Montero (UPC), Brent Salisbury (Red Hat) - persone che non ho mai incontrato fisicamente ma solo "by network" e che pure si sono dimostrate di cortesia infinita.

Non dimentico i compagni di viaggio che hanno reso più leggera la strada, condividendo soddisfazioni e delusioni del percorso universitario: Daniele, Gianmarco e Pierpaolo, con cui ho mosso i primi passi del cammino; Mirko, Lorenzo e Marco, con cui ho ingranato la marcia per il rush decisivo.

Ai miei genitori, una gratitudine che non trova parole per esprimersi: la soddisfazione di questo lavoro possa in parte ripagare l'affetto e la dedizione che non smettono di avere nei miei confronti.

Grazie infine a Martina, senza il cui amore non avrei trovato il coraggio di arrivare fin qui.

Indice

Elenco delle figure

Elenco delle tabelle

Acronimi

ACL	**A**ccess **C**ontrol **L**ist
ALT	**A**lternative **L**ogical **T**opology
API	**A**pplication **P**rogramming **I**nterface
ARP	**A**ddress **R**esolution **P**rotocol
BGP	**B**order **G**ateway **P**rotocol
CIDR	**C**lassless **I**nter-**D**omain **R**outing
COTS	**C**ommercial **O**ff-**T**he-**S**helf
CN	**C**OTS **N**etworking
DNS	**D**omain **N**ame **S**ystem
EID	**E**ndpoint **ID**entifier
ETR	**E**gress **T**unnel **R**outer
GRE	**G**eneric **R**outing **E**ncapsulation
IANA	**I**nternet **A**ssigned **N**umbers **A**uthority
IEEE	**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers
IH	**I**nner **H**eader
IP	**I**nternet **P**rotocol
IPSec	**I**nternet **P**rotocol **Sec**urity
IT	**I**nformation **T**ecnology
ITR	**I**ngress **T**unnel **R**outer
LACP	**L**ink / **A**ggregation **C**ontrol **P**rotocol
LAN	**L**ocal **A**rea **N**etwork
LISP	**L**ocator / **I**dentifier **S**eparation **P**rotocol
MAC	**M**edia **A**ccess **C**ontrol
MPLS	**M**ulti **P**rotocol **L**abel **S**witching
NIC	**N**etwork **I**nterface **C**ard

OF	**O**pen**F**low
OVS	**O**pen **V**irtual **S**witch
OVSDB	**O**pen **vS**witch **D**ata**B**ase
OSI	**O**pen **S**ystem **I**nterconnection
PITR	**P**roxy **ITR**
PETR	**P**roxy **ETR**
PBB	**P**rovider **B**ackbone **B**ridges
QoS	**Q**uality **o**f **S**ervice
RLOC	**R**outing **LOC**ator
RPC	**R**emote **P**rocedure **C**all
SDN	**S**oftware **D**eefined **N**etworking
SIP	**S**ession **I**nitiation **P**rotocol
TCP	**T**rasmission **C**ontrol **P**rotocol
TE	**T**raffic **E**ngineering
TLS	**T**ransport **L**ayer **S**ecurity
TTL	**T**ime **T**o **L**ive
UDP	**U**ser **D**atagram **P**rotocol
VLAN	**V**irtual **LAN**
VRF	**V**irtual **R**outing and **F**orwarding
VXLAN	**V**irtual e**X**tensible **LAN**
VM	**V**irtual **M**achine
xTR	**T**unnel **R**outer di tipo Ingress o Egress

A ***mamma*** *e* ***papà***,

che mi hanno insegnato
a pedalare senza rotelle

Capitolo 1

Introduzione

Le reti sono tipicamente costruite da un gran numero di device come router, switch e numerosi tipi di *middleboxes* nei quali sono implementate decine di protocolli complessi. Gli operatori di rete sono responsabili della configurazione di politiche che rispondano ad un vasto range di eventi e applicazioni della rete: essi devono trasformare manualmente queste politiche di alto livello in comandi di configurazione a basso livello che si adattino alle mutevoli condizioni della rete; spesso dispongono di strumenti limitati per eseguire operazioni di questo tipo, molto complesse. Come risultato, la gestione delle reti e l'ottimizzazione delle prestazioni sono piuttosto impegnative e soggette a errori. Il fatto che i device di rete siano di solito "scatole nere" integrate verticalmente, aggrava la sfida che devono affrontare operatori e amministratori di rete. Un ulteriore ostacolo - quasi insormontabile - che i professionisti di rete e i ricercatori devono fronteggiare, riguarda l'"ossificazione di Internet". A causa della sua gigantesca base di distribuzione e del fatto che è considerato parte dell'infrastruttura critica della nostra società (come i trasporti e l'energia elettrica), Internet è divenuto estremamente difficile da far evolvere sia in termini di infrastruttura fisica che di protocolli e prestazioni. Tuttavia, poiché le correnti ed emergenti applicazioni e servizi di Internet divengono più complessi ed esigenti, è necessario che Internet riesca a evolversi per affrontare tali nuove sfide.

Alle problematiche di rete emergenti negli anni, finora si è tentato di rispondere con soluzioni ad-hoc che, tuttavia, spesso non sono riuscite a imporsi per una naturale resistenza al cambiamento degli operatori: il rapporto costo/benefici deve pendere pesantemente verso i vantaggi per far sì che si scelga di intervenire sulla propria infrastruttura per implementare nuove funzionalità. Un esempio tipico di questa *ossificazione* è rappresentato dal protocollo IPv6: esso cerca di rispondere in maniera definitiva alla carenza di indirizzi IPv4 - un problema reale e significativo - ma richiede la sostituzione di tutti i router di Internet. Questo l'ha portato, dopo centinaia di test e "IPv6-day", ad un

deployment decisamente limitato ad alcune aree della rete; di conseguenza altre soluzioni hanno parallelamente preso piede e aggirato il problema originario, rendendolo *de facto* inesistente e comportando un sostanziale immobilismo del protocollo IPv6. Questo esempio ci insegna come sia necessario affrontare le attuali sfide del networking mediante un diverso approccio: appare cioè ineludibile un cambio di paradigma della rete, una rivoluzione sistemica che ponga definite condizioni per la possibilità di evoluzione di Internet; al tempo stesso tale trasformazione della rete deve potersi realizzare senza "*flag day*", per i motivi illustrati precedentemente.

Il **Software Defined Networking** (SDN) appare oggi la prospettiva capace di interpretare questi obiettivi e il mio lavoro di tesi s'innesta in questo percorso. Nel Capitolo 2 presenterò le opzioni di fondo del paradigma e come esse cercano di rispondere alle limitazioni della tecnologia attuale; approfondirò dunque l'architettura SDN così come definita dal protocollo OpenFlow, per definire infine modelli ibridi di SDN, i quali costituiscono vie operative per la transizione della rete verso la nuova filosofia.

Nel capitolo 3 illustrerò il protocollo implementato in modalità SDN, vale a dire il **Locator/Identifier Separation Protocol** (LISP). Non si tratta di una scelta casuale: LISP affronta per via protocollare la questione della scalabilità della rete, garantendo inoltre interoperabilità con gli apparati e i protocolli preesistenti; questo favorisce una graduale transizione.

Il Capitolo 4 mostrerà infine lo sviluppo - tramite il software **Open vSwitch** - di un nodo di rete SDN capace di supportare le funzionalità di forwarding di LISP. Tale lavoro costituisce il mio contributo all'attività di ricerca e sperimentazione in campo SDN. Gli obiettivi proposti dalla mia soluzione riguardano infatti:

- nel breve termine, la possibilità per uno switch SDN di implementare il piano dati di un protocollo legacy (LISP).
- nel lungo termine, l'integrazione con una strategia complessiva per reti ibride SDN.

Le conclusioni di questa tesi saranno discusse nel Capitolo 5; è infine inclusa un'appendice circa la terminologia tecnica dei protocolli utilizzati.

Capitolo 2

Software Defined Networking

L'idea delle "reti programmabili" è stata proposta come un modo per facilitare l'evoluzione della rete. In particolare, il **Software Defined Networking (SDN)** è un nuovo paradigma di networking nel quale l'hardware di *forwarding* è disaccoppiato dalle decisioni di controllo. Esso promette di semplificare drasticamente la gestione delle reti e consentire innovazione ed evoluzione. L'idea principale è permettere agli sviluppatori software di fare affidamento sulle risorse di rete nella stessa facile maniera con cui essi utilizzano risorse di storage e di calcolo. In SDN l'intelligenza della rete è dal punto di vista logico centralizzata in controllori software (nel piano di controllo), mentre i device di rete divengono semplici device di inoltro pacchetti (piano dati) che possono essere programmati mediante un'interfaccia aperta come *OpenFlow*.

2.1 Limitazioni delle attuali tecnologie di rete ed esigenza di un nuovo paradigma

L'esplosione dei device e dei contenuti mobili, la virtualizzazione dei server e l'avvento dei servizi cloud sono fra i trend che stanno conducendo l'industria del networking a riesaminare le tradizionali architetture di rete. Molte reti convenzionali sono gerarchiche, costruite con switch Ethernet disposti in strutture ad albero. Questa topologia aveva senso quando era dominante il paradigma client-server, ma un'architettura così statica mal si adatta alle necessità di calcolo e storage dinamici degli attuali data center aziendali. Alcune tendenze chiave dell'informatica conducono verso il bisogno di un nuovo paradigma di rete che includa la possibilità di variare i modelli di traffico: all'interno dei data center infatti, i modelli di traffico sono cambiati in maniera significativa. Piuttosto che applicazioni client-server in cui la maggior parte della comunicazione avviene

tra un client e un server, le applicazioni odierne accedono a differenti database e server creando una raffica di traffico "orizzontale" fra macchine prima di restituire i dati all'utente finale nel classico modello di traffico "verticale". Allo stesso tempo gli utenti stanno cambiando modello di traffico di rete poiché premono per accedere ai contenuti aziendali da qualsiasi tipo di device, connettendosi sempre e dovunque. Infine, molti amministratori di data center aziendali si stanno orientando su un modello di servizio che preveda private cloud, public cloud o una qualche forma mista di entrambi, con la conseguenza di traffico aggiuntivo attraverso la wide area network. Soddisfare le attuali esigenze di mercato è potenzialmente impossibile con le tradizionali architetture di rete. Di fronte a budget decisamente ridotti, le sezioni IT delle aziende stanno tentando di trarre il massimo dalle loro reti, utilizzando strumenti di gestione dei device e processi manuali. Gli operatori affrontano sfide importanti come la domanda esplosa di mobilità e di larghezza di banda; i profitti si vanno erodendo per la crescita dei costi in conto capitale dell'infrastruttura assieme al declino dei ricavi. Le esistenti architetture di rete non furono progettate per soddisfare le richieste attuali degli utenti, delle imprese e degli operatori; piuttosto, i progettisti di rete sono vincolati dai limiti della rete corrente, che includono:

Complessità che porta alla stasi: la tecnologia di rete è fino ad oggi consistita di una notevole serie di protocolli progettati per connettere in maniera affidabile host su distanze arbitrarie, velocità di collegamento e topologie. Per soddisfare esigenze commerciali e tecniche, negli ultimi decenni sono stati sviluppati protocolli di rete in grado di fornire prestazioni e affidabilità superiori, più ampia connettività e sicurezza più rigorosa. I protocolli tendono a essere definiti in maniera isolata o comunque ciascuno a risolvere un problema specifico e senza il beneficio di alcuna astrazione fondamentale. Questo ha portato ad una delle principali limitazioni della rete di oggi: la complessità. Per esempio, per aggiungere o rimuovere qualsiasi device, è necessario intervenire su diversi switch, router, firewall, portali di autenticazione web, ecc. oltre ad aggiornare ACL, VLAN, QoS e altri meccanismi basati su protocollo, attraverso strumenti di management dei device. Inoltre bisogna considerare la topologia di rete, i modelli degli switch proprietari e la versione del software. A causa di questa complessità, le reti di oggi sono relativamente statiche dal momento che l'IT cerca di minimizzare il rischio di interruzione del servizio. La natura statica delle reti è in forte contrasto con la natura dinamica dell'ambiente server di oggi, in cui la virtualizzazione dei server ha aumentato decisamente il numero di host che necessitano di connettività di rete e ha sostanzialmente alterato le assunzioni relative alla locazione fisica degli host. Prima dell'avvento della virtualizzazione, le applicazioni risiedevano su un singolo server e scambiavano traffico principalmente con client selezionati. Oggi le applicazioni sono distribuite fra macchine virtuali (VM)

multiple, che scambiano fra loro flussi di traffico. Le VM vengono migrate per ottimizzare e bilanciare il carico dei server, causando nel tempo la variazione (talvolta rapida) dei punti terminali dei flussi esistenti. La migrazione delle VM sfida molti aspetti del tradizionale networking, dagli schemi di indirizzamento alle nozioni basilari di un piano di instradamento hop-by-hop. Infine, per l'adozione di tecnologie di virtualizzazione, molte aziende oggi implementano una rete IP-convergente per voce, dati e traffico video. Le reti esistenti possono fornire livelli differenziati di qualità del servizio per applicazioni diverse, ma la fornitura di tali risorse è prevalentemente manuale: l'IT deve configurare separatamente ciascun apparato proprietario e regolare parametri come la larghezza di banda e QoS su base sessione o applicazione. A causa di questa sua statica natura, la rete non può dinamicamente adattarsi alla variazione del traffico, delle applicazioni e delle richieste dell'utente.

Politiche inconsistenti: per implementare una *network policy* su una rete vasta, l'IT dovrebbe configurare migliaia di device e meccanismi. Ad esempio, ogni volta che una nuova macchina virtuale è "tirata su", possono essere necessarie ore, in alcuni casi giorni, per riconfigurare le ACL nell'intera rete. La complessità della rete di oggi rende molto difficile applicare una serie di accessi, sicurezza, QoS e altre politiche per utenti sempre più mobili, rendendo l'impresa vulnerabile a violazioni di sicurezza, non conforme ai regolamenti, e ad altre conseguenze negative.

Incapacità di scalare: A un rapido incremento delle richieste nel data center deve corrispondere una crescita della rete; allo stesso tempo, la rete diviene molto più complessa con l'aggiunta di centinaia di migliaia di device che devono essere configurati e gestiti. L'IT è spesso ricorsa all'*oversubscription* dei link per scalare la rete, basandosi su modelli di traffico prevedibili; tuttavia nei data center virtualizzati di oggi, i modelli di traffico sono estremamente dinamici e pertanto imprevedibili. I grandi operatori, come *Google*, *Yahoo!* e *Facebook*, affrontano sfide di scalabilità sempre più scoraggianti. Questi provider implementano algoritmi di elaborazione paralleli su vasta scala attraverso il loro intero pool di calcolo.

Dipendenza dai vendor: gli operatori e le imprese cercano di sviluppare nuove abilità e servizi in rapida risposta alle mutevoli esigenze di business o alle richieste degli utenti. Ad ogni modo, la loro capacità di risposta è ostacolata dai cicli produttivi degli apparati proprietari, che possono durare fino a tre anni o più. La mancanza di standard e interfacce *open* limita la possibilità degli operatori di rete di adattare la rete ai propri ambienti. Questa mancata corrispondenza tra le richieste di mercato e le capacità di rete ha portato l'industria a un punto di non ritorno, che ha portato alla creazione dell'architettura Software Defined Networking e allo sviluppo degli standard ad essa associati.

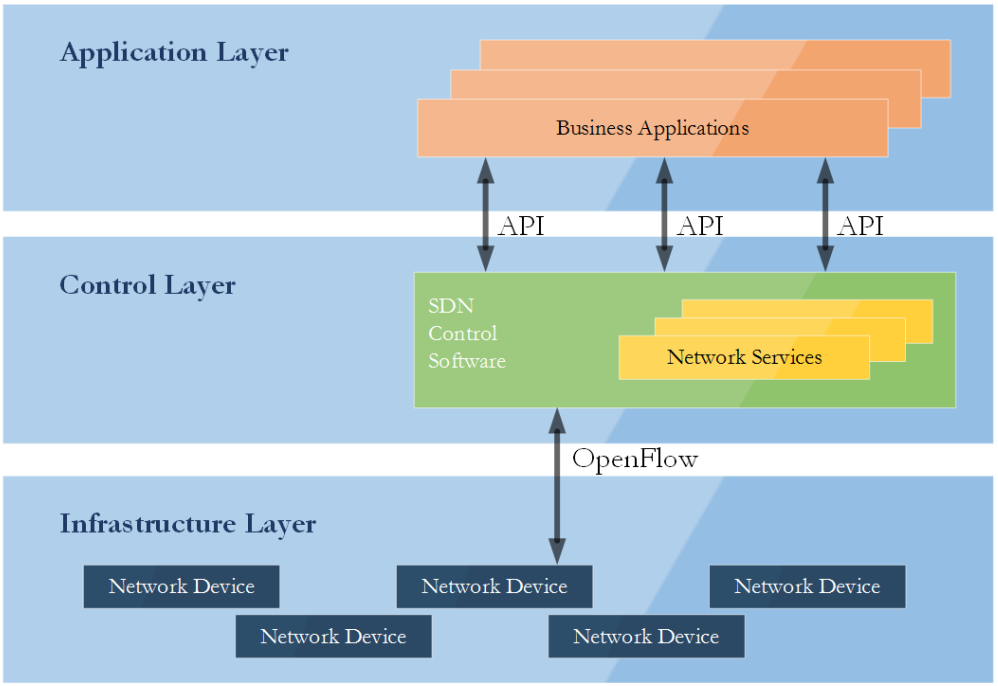

FIGURA 2.1: Architettura Software-Defined Network.

2.2 Architettura SDN

Software Defined Networking (SDN) è un'emergente architettura di rete dove il piano di controllo è disaccoppiato dalle funzioni di *forwarding* ed è direttamente programmabile. Questa migrazione del controllo, finora strettamente legato ai singoli dispositivi di rete, verso device di calcolo accessibili, consente all'infrastruttura sottostante di essere astratta per applicazioni e servizi di rete, che possono trattare la rete come un'entità logica o virtuale.

La figura 2.1) illustra una vista logica dell'architettura SDN. L'intelligenza di rete è logicamente centralizzata in dei controller SDN di tipo software, che conservano una visione globale della rete. Come risultato, la rete appare alle applicazioni - e la policy si comporta - come di fronte ad un unico switch logico. Attraverso SDN, le aziende e gli amministratori guadagnano autonomia dai vendor nel controllo sull'intera rete da un unico punto logico, semplificando enormente la progettazione e la gestione della rete. SDN semplifica in maniera significativa anche i device di rete, dal momento che essi non necessitano più di comprendere e processare migliaia di protocolli standard, ma accettare meramente le istruzioni dettate dai controller SDN. Forse ancora più importante, gli operatori e gli amministratori di rete possono configurare in maniera programmatica questa semplificata astrazione di rete piuttosto che avere da gestire manualmente decine di migliaia di linee di configurazione sparse fra migliaia di device. Inoltre, sfruttando

l'intelligenza centralizzata del controller, l'IT può alterare in modalità real-time il comportamento della rete e implementare nuove applicazioni e servizi in una questione di ore o di giorni, piuttosto che in settimane o mesi così come richiesto oggi. Centralizzando lo stato della rete nel control layer, SDN dà ai gestori di rete la flessibilità per configurare, gestire, mettere al sicuro e ottimizzare le risorse di rete mediante programmi SDN dinamici e automatizzati; essi stessi possono scrivere tali programmi, senza attendere l'aggiunta di nuove *features* agli ambienti software proprietari nel mezzo della rete. Oltre all'astrazione della rete, l'architettura SDN supporta una serie di API che rendono possibile l'implementazione di comuni servizi di rete, inclusi routing, multicast, sicurezza, controllo degli accessi, gestione della banda, ingegneria del traffico, qualità del servizio, ottimizzazione del *processing* e dello *storage*, risparmio energetico e tutte le forme di *policy management*, personalizzate sui clienti, per raggiungere gli obiettivi del business. Ad esempio, un'architettura SDN rende facile definire e rafforzare politiche coerenti per l'accesso cablato e wireless alla rete di un'ateneo. Analogamente, SDN rende possibile la gestione dell'intera rete attraverso sistemi intelligenti di *provisioning* e *orchestration*. In questo modo, attraverso API tra i layer SDN di controllo e applicativo, le applicazioni business possono operare su un'astrazione della rete, sfruttando i servizi e le capacità della stessa, senza essere legati ai dettagli della loro implementazione. SDN rende la rete non più "*application-aware*" ma piuttosto "*application-customized*" e di conseguenza le applicazioni non "*network-aware*" ma "*network-capability-aware*". La conseguenza è un'ottimizzazione delle risorse di calcolo, storage e rete.

2.3 Il protocollo *OpenFlow*

Il Software Defined Networking è stato sviluppato per facilitare l'innovazione e abilitare un semplice controllo programmatico del percorso dati nella rete; ci soffermeremo sul modello immaginato da *OpenFlow*, la specifica che su questo fronte ha riscontrato maggiore successo, divenendo lo standard *de facto* di SDN.

2.3.1 Switch

Uno Switch *OpenFlow* consiste di una o più *flow table* e di una *group table*, che eseguono *lookup* e *forwarding* dei pacchetti; un *OpenFlow Channel* interagisce con un controller esterno (Figura 2.2). Attraverso il protocollo *OpenFlow*, lo switch comunica con il controller e il controller gestisce lo switch stesso: utilizzando il protocollo *OpenFlow*, il controller può aggiungere, aggiornare e cancellare *flow entries* nelle *flow table*, sia in modalità reattiva (in risposta al pacchetto) che proattiva.

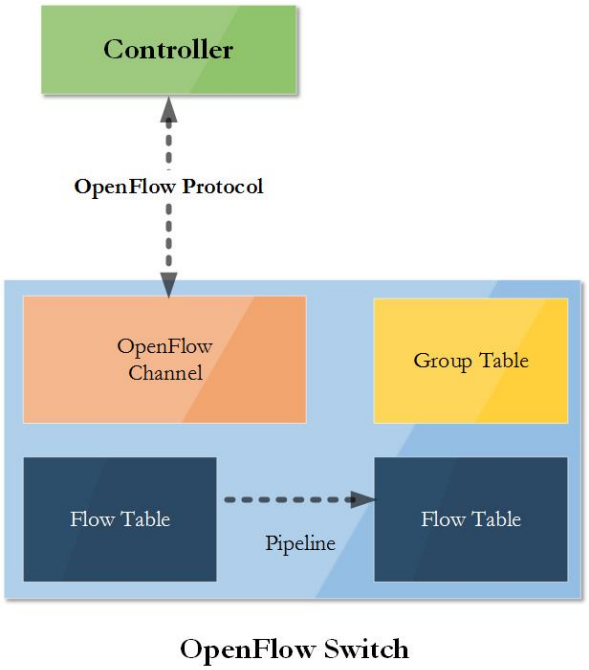

FIGURA 2.2: Principali componenti di uno switch OpenFlow.

Ciascuna ***flow table*** nello switch contiene una serie di *flow entries*; ogni flusso consiste di *match fields*, *counters* e una serie di *instructions* da applicare ai pacchetti positivi al *matching*. Il *matching* ha inizio alla prima *flow table* e può continuare nelle successive *flow table*; le *flow entries matchano* i pacchetti secondo un ordine di priorità, utilizzando la *matching entry* rinvenuta per prima in ogni tabella. Per ogni *matching entry*, sono eseguite le istruzioni associate alla relativa *flow entry*; in caso di nessun *matching*, le operazioni da eseguire dipendono dalla configurazione prevista in caso di *flow entry* mancante: ad esempio, il pacchetto può essere inoltrato al controller sul canale *OpenFlow*, scartato, oppure inoltrato alla successiva *flow table*. Le istruzioni associate con ciascuna *flow entry* possono sia contenere azioni da intraprendere sia modifiche alla *pipeline processing*. Le azioni incluse nelle istruzioni descrivono il *forwarding* o la modifica dei pacchetti e l'elaborazione da parte di una *group table*. Le istruzioni della pipeline *processing* consentono ai pacchetti di essere inviati alle successive tabelle per ulteriore elaborazione e permettono all'informazione, nella veste di metadati, di essere comunicata fra tabelle. Il *processing* termina quando la serie di istruzioni associate ad un *matching flow entry* non specifica un ulteriore tabella; a questo punto il pacchetto è generalmente modificato e *forwardato*.

Le *flow entries* possono essere inoltrate verso una porta; si tratta generalmente di una porta fisica, ma potrebbe anche essere una porta logica definita dallo switch oppure una porta riservata definita dal protocollo OF stesso (cfr. paragrafo 2.3.1.1). Una porta può specificare generiche azioni di *forwarding* - come invio al controller o *flooding* - oppure

forwarding di tipo non-*OpenFlow*, come il *processing* di un "normale" switch, mentre le porte logiche possono specificare *link aggregation*, tunnel o interfacce di loopback. Le azioni associate con le *flow entries* possono anche indirizzare pacchetti a un gruppo che specifica del *processing* aggiuntivo. I gruppi contengono una serie di azioni per il *flooding* ma anche per una più complessa semantica di *forwarding* (multipath, fast reroute, link *aggregation*); i gruppi consentono anche *flow entries* multiple sulla base di un singolo identificatore (es.: IP *forwarding* ad un comune next hop). Questa astrazione permette che azioni di output, comuni a diverse *flow entries*, possano essere modificate in maniera efficiente.

La ***group table*** contiene group entries, a ciascuna delle quali corrisponde una lista di *action bucket* con una semantica specifica per ciascun tipo di gruppo; le azioni in uno o più bucket sono applicate ai pacchetti inviati al gruppo.
I progettisti di switch sono liberi di realizzare le componenti interne in qualsiasi modo conveniente, purché siano preservate la corretta semantica del *matching* e delle *instructions*. Ad esempio, la pipeline prevista nello switch OF può essere fisicamente implementata con un diverso numero di tabelle hardware.

2.3.1.1 Porte *OpenFlow*

Le porte OF sono le interfacce di rete di passaggio per i pacchetti fra il *processing* *OpenFlow* e il resto della rete; gli switch OF sono interconnessi logicamente mediante le porte OF. Il numero di porte OF non coincide necessariamente con le interfacce fisiche dello switch: alcune interfacce hardware potrebbero essere disabilitate per *OpenFlow* e allo stesso tempo OF potrebbe definire porte logiche in aggiunta alle interfacce hardware.
I pacchetti OF sono ricevuti su una **ingress port** e processate dalla pipeline *OpenFlow*, la quale può inoltrare loro verso una **output port**. Uno switch *OpenFlow* supporta tre tipi di porte OF:

- **Porte fisiche**: si tratta di porte dello switch corrispondenti alle sue interfacce hardware; ad esempio, in uno switch Ethernet, le porte fisiche sono mappate una ad una con le interfacce Ethernet.
- **Porte logiche**: sono porte dello switch non corrispondenti direttamente a interfacce hardware dello switch. Le porte logiche sono astrazioni di più alto livello definibili anche in switch non-OF; esse possono includere l'incapsulamento dei pacchetti ed essere mappate con diverse porte fisiche. Le porte logiche differiscono da quelle fisiche unicamente per la possibilità di avere associato con esse un campo metadati extra chiamato *Tunnel-ID*.

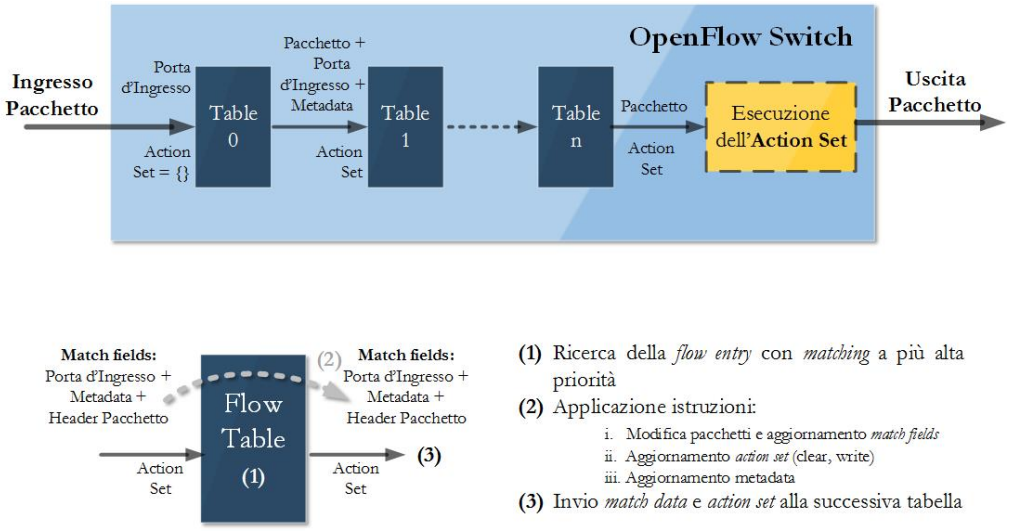

FIGURA 2.3: Nella pipeline i pacchetti sono *matchati* su più tabelle.

- **Porte riservate**: le porte riservate sono quelle definite dal protocollo *OpenFlow*, destinate a generiche funzioni di inoltro, quali l'invio al controller, il *flooding*, o il *forwarding* non-OF, come il "normale" *processing* dello switch.

2.3.1.2 Tabelle *OpenFlow*

Gli switch OF compatibili possono essere di due tipi: *OpenFlow-only* e *OpenFlow-hybrid*; nel primo caso sono supportate solo le operazioni di tipo *OpenFlow*, per cui tutti i pacchetti sono processate dalla pipeline *OpenFlow* e non possono essere elaborati diversamente. Gli switch OF **ibridi** invece supportano sia le operazioni *OpenFlow* che il normale *switching* Ethernet; si rende pertanto necessario un meccanismo di classificazione - esterno a *OpenFlow* - che instradi il traffico verso la pipeline normale oppure quella di tipo *OpenFlow*. Ad esempio, uno switch potrebbe utilizzare il tag VLAN o la input port del pacchetto per decidere su quale pipeline processare il pacchetto.

La **pipeline** *OpenFlow* di ogni switch OF contiene più *flow table*, ciascuna contenente diverse *flow entries*. La pipeline OF stabilisce come i pacchetti interagiscono con le *flow table* (Figura 2.3). Uno switch OF deve avere almeno una *flow table* e può averne più di una; chiaramente nel caso di un'unica *flow table* il pipeline *processing* è enormemente semplificato. Nel caso di *flow table* multiple, ciascuna di esse è numerata in maniera sequenziale da 0 a N; un pacchetto processato in una *flow table* può essere inoltrato esclusivamente verso una *flow table* di numero superiore.

Una ***flow table*** (tabella 2.1) consiste di *flow entries*, contenenti i seguenti campi:

Match Fields	Priority	Counters	*instructions*	Timeouts	Cookie

TABELLA 2.1: Principali componenti di una *flow entry* in una *flow table*.

- **match fields**: campi da utilizzare nel *matching* del pacchetto; si tratta della porta d'ingresso e degli header del pacchetto, opzionalmente metadati specificati da una tabella precedente.
- **priority**: ordine di precedenza della *flow entry* nell'operazione di *matching*.
- **counters**: viene aggiornato quando i pacchetti sono *matchati*.
- **instructions**: modificano le azioni da intraprendere sul pacchetto o modificano la pipeline del *processing*.
- **timeouts**: tempo massimo oltre il quale il flusso è eliminato dallo switch.
- **cookie**: valore scelto dal controller, utilizzato da esso per filtrare statistiche, modifiche e cancellazioni di flussi; non interviene nel *processing* dei pacchetti.

Una *flow table* entry è identificata univocamente dai campi *match fields* e *priority*; la *flow entry* che non omette tutti i campi e ha priorità pari a 0 corrisponde alla *table-miss flow entry*. Il processo di match-action è ben descritto in figura 2.4 Alla ricezione di un pacchetto, lo switch OF avvia l'esecuzione del *table lookup* nella prima *flow table* e, secondo la pipeline *processing*, eventualmente procede con il table lookup nelle successive *flow table*. Un pacchetto matcha una *flow table* entry se i valori nei match fields del pacchetto utilizzati per il lookup combaciano con quelli definiti nella *flow table* entry; i match fields rappresentano il pacchetto nello stato corrente, ovvero subiscono modifiche se le *actions* intraprese nelle precedenti *flow table* hanno modificato l'header del pacchetto. La *flow entry* a più alta priorità che matcha il pacchetto viene selezionata: i counters associati sono aggiornati e vengono eseguite le azioni corrispondenti. Ogni *flow table* deve supportare una ***table-miss flow entry*** per processare i pacchetti che non hanno riscontrato alcun *matching* nelle entries della *flow table*; tali pacchetti possono - ad esempio - essere scartati, inviati al controller o ad una *flow table* successiva. La table-miss *flow entry* si comporta sostanzialmente come qualsiasi altra *flow entry*; qualora non sia stata definita, di default i pacchetti *not-matching* con nessuna *flow entry*, sono *droppati*.
Sono inoltre previsti meccanismi aggiuntivi - se vogliamo più raffinati - per l'esecuzione di una varietà di *actions* corrispondenti ad uno o più flussi, oppure relative alle performance (servizi QoS), quali le ***group table*** e le ***meter table***; tuttavia non saranno approfonditi in questa tesi, non essendo rilevanti ai fini del lavoro svolto.

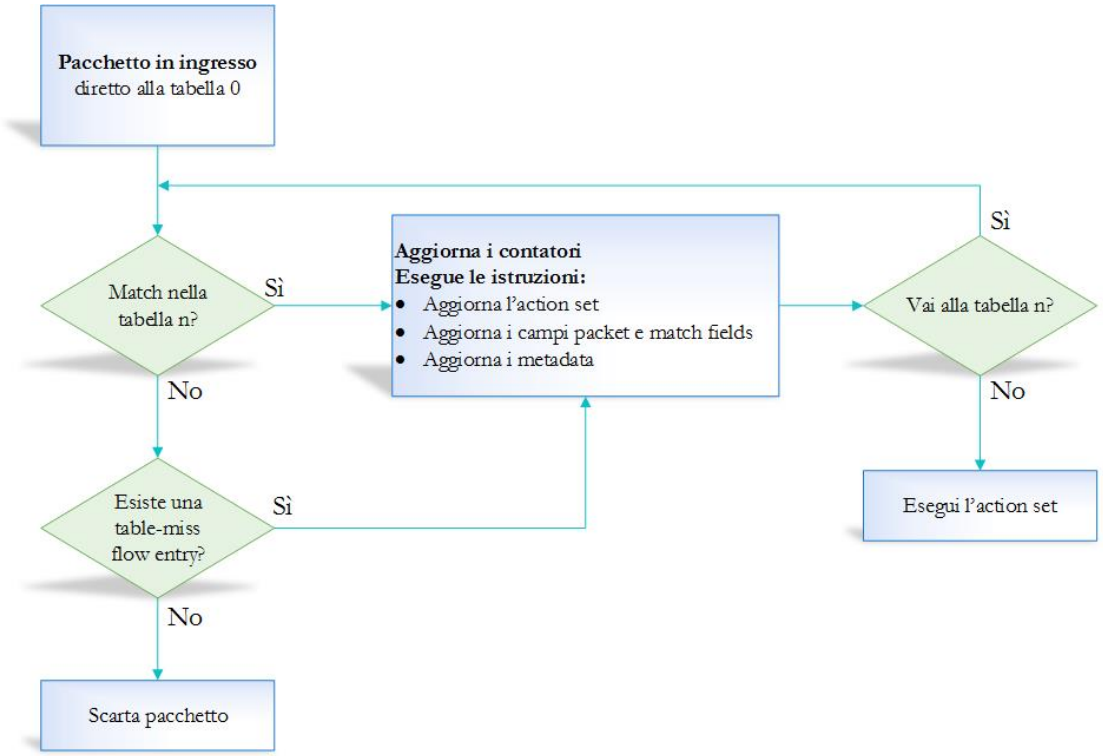

FIGURA 2.4: Operazioni compiute su un pacchetto attraverso uno switch *OpenFlow*.

2.3.1.3 Istructions e Actions

Ciascuna *flow entry* contiene una serie d'istruzioni che sono eseguite quando un pacchetto matcha una *flow entry*; uno switch OF supporta obbligatoriamente alcune istruzioni, altre sono invece opzionali.

La serie d'istruzioni associate ad ogni *flow entry* contiene al massimo un'istruzione per tipo, le quali sono eseguite secondo l'ordine indicato in tabella 2.2. In pratica gli unici vincoli sono che l'istruzione *Meter* sia eseguita prima della *Apply-Actions*, che la *Clear-Actions* sia eseguita prima della *Write-Actions* e che la *Goto-Table* sia eseguita per ultima.
Ad ogni pacchetto è dunque associato un ***Action Set*** - vuoto di default - che attraversando la pipeline *processing* si costituisce tramite le istruzioni Write-Action e Clear-Action associate con un particolare match per ogni *flow table*; quando l'istruzione *Goto-Table* non è presente in una *flow table*, s'interrompe la pipeline e l'action set del pacchetto è eseguito. All'interno di ogni Action Set, le azioni sono eseguite nell'ordine di cui sotto, sebbene possano essere state aggiunte al set in un diverso ordine:

1. **copy TTL inwards**: applica sul pacchetto le azioni di copia del TTL interno.
2. **pop**: applica al pacchetto tutte le azioni di *tagging* pop.

Meter *meter_id* (*opzionale*)	Indirizza i pacchetti verso una specifica metrica.
Apply-Actions *actions* (*opzionale*)	Applica le specifiche azioni immediatamente, senza alcuna modifica all'Action Set.
Clear-Actions (*opzionale*)	Elimina istantaneamente tutte le azioni previste nell'Action Set.
Write-Actions actions (*richiesta*)	Aggiunge le azioni specificate nell'action set corrente, sovrascrivendo le azioni ridondanti.
Write-Metadata *metadata/mask* (*opzionale*)	Scrive i metadati nei *metadata fields*.
Goto-Table *next-table-id* (*richiesta*)	Indica la tabella successiva nella pipeline *processing*. Il table-id specificato deve essere maggiore di quello corrente. L'ultima *flow table* non deve includere questa istruzione; per gli switch OF con una singola *flow table* chiaramente si tratta di un'istruzione non necessaria.

Tabella 2.2: Istructions supportate dagli switch *OpenFlow*.

3. **push-MPLS**: applica al pacchetto l'azione di *tagging* push MPLS.
4. **push-PBB**: applica al pacchetto l'azione di *tagging* push PBB.
5. **push-VLAN**: applica al pacchetto l'azione di *tagging* push VLAN.
6. **copy TTL outwards**: applica sul pacchetto le azioni di copia del TTL esterno.
7. **decrement TTL**: applica sul pacchetto l'azione di decremento del TTL.
8. **set**: applica sul pacchetto le azioni di *set-field*.
9. **qos**: applica sul pacchetto le azioni di QoS, come *set-queue*.
10. **group**: se è specificata una *group action*, sono applicate le azioni relative, secondo lo stesso ordine di questo elenco.
11. **output**: se non è specificata una *group action*, inoltra il pacchetto sulla porta *OpenFlow* (fisica, logica o riservata) specificata dall'azione output.

Drop: Non c'è un'azione specifica per rappresentare lo scarto dei pacchetti; piuttosto, i pacchetti i cui action set non prevedono azioni di output dovrebbero essere scartati.

L'azione di output è eseguita per ultima; se in un action set sono specificate le azioni sia output sia group, quest'ultima ha precedenza e l'output è ignorato. Qualora non

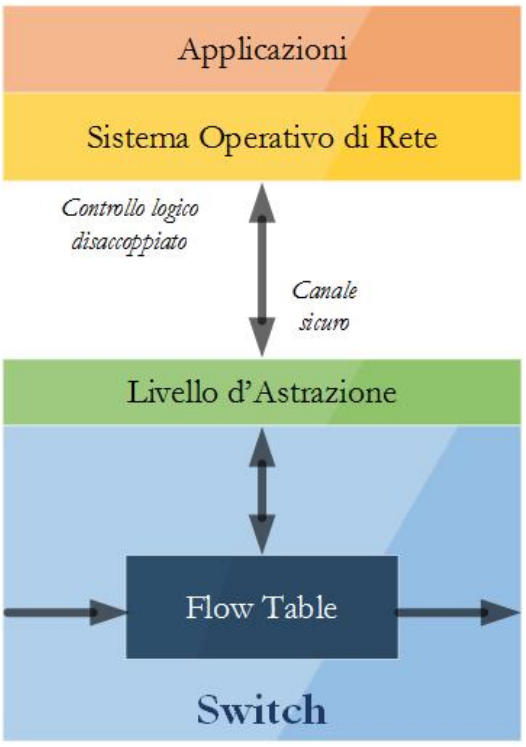

FIGURA 2.5: Il controllo logico separato può essere visto come un sistema operativo di rete, sul quale costruire i "programmi" di rete.

siano state definite né le azioni group né action, il pacchetto è scartato. L'esecuzione delle group è ricorsiva fin quanto supportata dallo switch.

2.3.2 Controller

Il sistema disaccoppiato è comparato ad un sistema operativo nel quale il controller fornisce un'interfaccia programmatica alla rete, e dove le applicazioni possono essere scritte per migliorare la gestione dei compiti e offrire nuove funzionalità. Una visione stratificata di questo modello è illustrata in figura 2.5. Questa vista assume che il controllo è centralizzato e le applicazioni sono scritte come se la rete fosse un singolo sistema. Mentre questo semplifica il rafforzamento della politica e le attività di gestione, devono essere mantenuti legami stretti fra il controller e gli elementi di *forwarding* della rete. La prima preoccupazione che si riscontra quando si trasferisce il controllo dall'hardware di commutazione riguarda la scalabilità e le performance del controllore di rete. Ciò nonostante possono essere usati controller multipli per ridurre la latenza o aumentare la tolleranza ai guasti. Una questione correlata è il problema della collocazione del controllore, che consiste nel determinare sia il numero ottimale dei controllori che la loro locazione all'interno della topologia di rete, spesso scegliendo tra l'ottimazione della latenza tra caso medio e caso peggiore. Inoltre, la modellizzazione del controllo impatta

enormemente sulla scalabilità della rete. Nel seguito, discutiamo differenti metodi di controllare una SDN, molti dei quali sono interrelati.

2.3.2.1 Modelli di controllo: Centralizzato contro Distribuito

Nonostante protocolli come *OpenFlow* specifichino come uno switch sia controllato da un unico controller e dunque sembrino implementare la centralizzazione, SDN può avere un piano di controllo centralizzato o distribuito. Sebbene la comunicazione fra controller non sia definita da *OpenFlow*, essa si rende necessaria per qualsiasi tipo di distribuzione o ridondanza nel piano di controllo. Un controller fisicamente centralizzato rappresenta infatti un *single point of failure* per l'intera rete; per questo, *OpenFlow* consente la connessione di più controller allo switch, permettendo di *backuppare* i controller per assumerne le funzionalità in caso di guasto. Permane anche l'idea di tentare di mantenere un piano di controllo logicamente centralizzato ma fisicamente distribuito; questo diminuirebbe l'*overhead* di ricerca, sfruttando la comunicazione con controller locali, ma conservando ancora la possibilità per le applicazioni di essere scritte con una visione centrale e semplificata della rete. Il potenziale rovescio della medaglia è costituito da quei trade-off che possono essere realizzati con consistenza e pesantezza quando la distribuzione dello stato avviene per tutto il piano di controllo, con la potenziale conseguenza di provocare il comportamento scorretto di quelle applicazioni che credono di avere una visione precisa. Un approccio ibrido può utilizzare i controller locali per le applicazioni locali e rimandare ad un controller globale le decisioni che richiedono uno stato centralizzato della rete. Questo riduce il carico sul controllore globale filtrando il numero di nuove richieste di flusso, mentre fornisce anche il path con risposte più veloci, le quali possono essere gestite da un'applicazione locale di controllo. Una rete SDN può avere anche un certo livello di decentralizzazione logica, con diversi controller logici.

2.3.2.2 Granularità del Controllo

Tradizionalmente l'unità base del networking è stata il pacchetto. Ciascun pacchetto contiene le informazioni d'indirizzo necessarie per uno switch di rete per prendere decisioni di routing individuali. Ad ogni modo, la maggior parte delle applicazioni invia dati come un flusso di diversi pacchetti individuali. Una rete che desidera offrire QoS o servizi garantiti a certe applicazioni potrebbe giovare dal controllo basato su singoli flussi piuttosto che su singoli pacchetti. Il controllo può essere ulteriormente astratto ad un match di flussi aggregati piuttosto che flussi individuali. L'aggregazione dei flussi può essere basata sulla sorgente, sulla destinazione, sull'applicazione o una qualsiasi combinazione delle precedenti. In una rete SDN con controllo remoto dell'hardware di rete,

l'*overhead* è provocato dal traffico fra il piano dati e il piano di controllo. Come tale, utilizzare la granularità basata su pacchetto esporrebbe ad un ritardo aggiuntivo poiché il controller dovrebbe prendere una decisione per ogni pacchetto in arrivo. Controllando flussi individuali la decisione presa per il primo pacchetto del flusso può essere applicata a tutti i successivi pacchetti del flusso incluso nel piano dati. L'*overhead* potrebbe essere ulteriormente ridotto raggruppando insieme i flussi, come il traffico tra due host, ed eseguendo le decisioni di controllo sui flussi aggregati.

2.3.2.3 Politiche Reattive contro Politiche Proattive

Con un modello a controllo **reattivo**, gli elementi di *forwarding* devono consultare un controller ogni volta che deve essere presa una decisione, come ad esempio quando raggiunge lo switch un nuovo flusso di pacchetti. Nel caso di granularità di controllo basata su flusso, ci sarà un piccolo ritardo prestazionale dovendo inoltrare al controller per la decisione (es.: inoltro o scarto) il primo pacchetto di ogni nuovo flusso, dopo il quale, i futuri pacchetti di traffico inclusi in quel flusso viaggeranno al tasso di linea all'interno dell'hardware di *forwarding*. Mentre il ritardo del primo pacchetto è trascurabile in molti casi, esso potrebbe divenire un problema se il controller è geograficamente remoto (nonostante questo possa essere mitigato mediante la distribuzione fisica del controller) oppure se la maggior parte dei flussi è di breve vita, come i flussi a singolo pacchetto. Ci sono anche alcuni problemi nelle reti più grandi, come il *throughput* del controller che deve essere in grado di gestire il volume delle nuove richieste dei flussi. Un approccio alternativo, di tipo **proattivo**, consiste nel trasferire le regole di politica dal controller agli switch: il controller raramente richiede di essere consultato circa i nuovi flussi e il traffico è tenuto all'interno del piano dati.

2.3.3 Comunicazione *Southband*: Controller-Switch

Un importante aspetto delle SDN è il collegamento tra il piano dati e il piano di controllo; siccome gli elementi di *forwarding* sono controllati da un'interfaccia aperta, è importante che questo collegamento rimanga disponibile e sicuro. Il protocollo *OpenFlow* può essere visto come una possibile implementazione delle interazioni controller-switch, poiché definisce la comunicazione tra l'hardware di commutazione e il controllore di rete in quello che è chiamato ***OpenFlow Channel***.
OpenFlow Channel è l'interfaccia che connette ogni switch OF al controller; attraverso questa interfaccia, il controller configura e gestisce gli switch, dai quali riceve eventi e ai quali inoltra pacchetti. L'interfaccia fra il datapath e l'OF *Channel* è specifica dell'implementazione dello switch, ma i messaggi OF *Channel* hanno il formato previsto

dalla specifica *OpenFlow*. L'*OpenFlow Channel* è solitamente criptato con TSL, ma può essere eseguito direttamente anche su TCP. Il protocollo OF supporta tre tipi di messaggio:

1. **controller-to-switch**: sono messaggi iniziati dal controller e usati per gestire direttamente o ispezionare lo stato degli switch.
2. **asynchronous**: sono messaggi iniziati dallo switch e utilizzati per aggiornare il controller degli eventi di rete e le modifiche dello stato dello switch.
3. **symmetric**: sono messaggi iniziati sia dallo switch sia dal controller e inviati senza sollecito.

2.3.4 Comunicazione *Northband*: Controller-Applicazioni

I sistemi di gestione esterna o i servizi di rete potrebbero auspicare di estrarre informazioni circa la rete sottostante o di controllare un aspetto del comportamento o della politica della rete. In aggiunta i controller potrebbero trovare necessario comunicare fra di loro per una varietà di motivi: un'applicazione interna di controllo potrebbe avere bisogno di riservare risorse fra diversi domini di controllo, un controller primario potrebbe avere bisogno di condividere le informazioni di policy con un backup, ecc. Diversamente dalla comunicazione controller-switch, ad oggi non esiste uno standard accettato per le interazioni *northbound* ed essi sono più facili da implementare su base *ad hoc* per particolari applicazioni.

2.4 SDN ibride

Tramite la centralizzazione e personalizzazione del piano di controllo, SDN promette di facilitare la progettazione e la gestione della rete; tuttavia il dibattito fra operatori è aperto su come favorire una larga diffusione di SDN. Infatti, se da un lato la transizione dal paradigma CN[1] a quello SDN è incoraggiata, dall'altro l'impiego di SDN nelle reti esistenti pone sfide non trascurabili sul piano economico, organizzativo e tecnico. Innanzitutto SDN ha dei costi iniziali di *deployment* non trascurabili, in termini di rinnovo degli apparati e mancanza di competenze; chiaramente gli operatori sono solitamente riluttanti nel dismettere costosi nodi CN per abilitare un completo impiego di una

[1]Con il termine *COTS Networking* (CN) si indicano reti progettate, realizzate e gestite tramite apparati legacy che non subiscono alcuna modifica dagli acquirenti; questi ultimi sono responsabili dell'interazione fra dispositivi eterogenei, realizzati da vendor differenti. I nodi della rete eseguono pertanto le funzioni di controllo utilizzando software proprietari e non possono essere in alcun modo modificati.

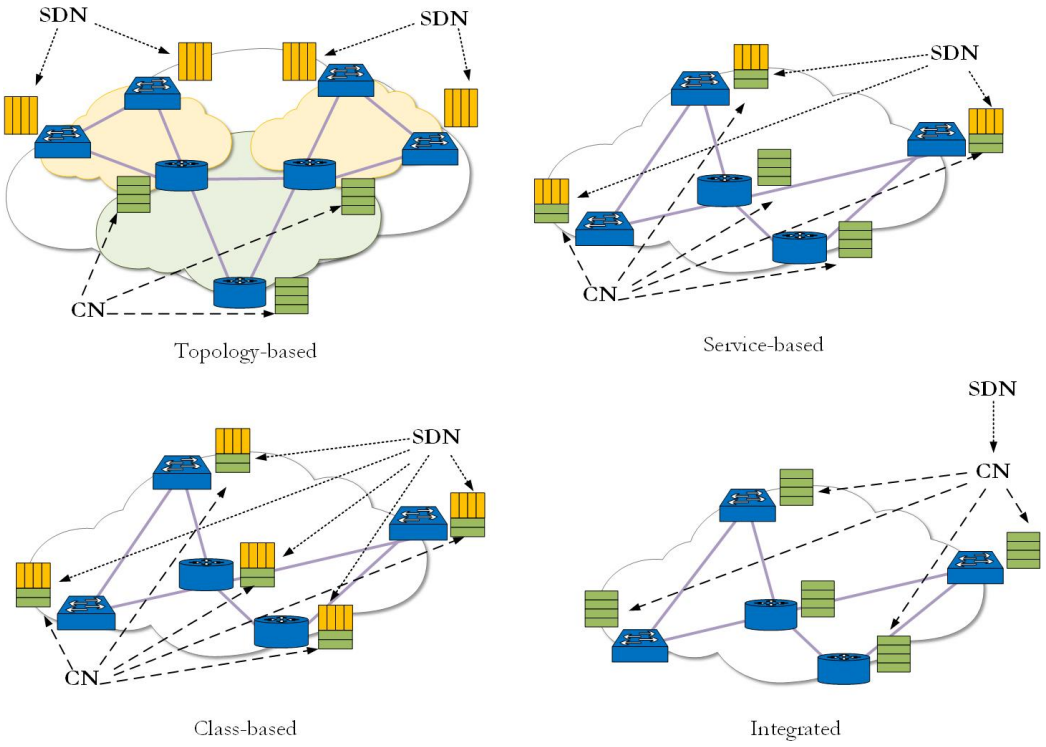

FIGURA 2.6: Esempi di modelli ibridi SDN.

SDN. Inoltre, dal momento in cui SDN richiede un cambio radicale nella mentalità degli operatori, essi necessitano di qualificarsi nella progettazione e gestione delle reti SDN, a meno di essere rimpiazzati da una nuova generazione di programmatori di rete. In secondo luogo, i controller SDN riscontrano ancora difficoltà di realizzazione a livello di produzione; nonostante i recenti sforzi nel fornire linguaggi di programmazione di alto livello e potenti astrazioni, probabilmente ancora non sono disponibili metodologie e strumenti effettivi per costruire controller SDN affidabili nel gestire complesse politiche di rete. Oltre a tali questioni operative, che franano gli operatori nel perseguire una rapida transizione a SDN, ci sono alcuni aspetti critici del paradigma stesso che vanno considerati. Innanzitutto, per garantire affidabilità nell'aggiornamento dei flussi, si rendono necessarie vaste aree di rete *out-of-band* fra controller e nodi, che comportano una duplicazione della progettazione della gestione della rete; attualmente infatti, si fa un uso dell'*out-of-band* solo per operazioni non critiche e comunque sempre in sottoreti di dimensioni ridotte. La centralizzazione logica può anche complicare il management del piano di controllo, come il suo stesso aggiornamento (deploy di una nuova applicazione di rete, installazione di una nuova versione del software del controller); durante un upgrade di questo tipo, non sarebbe disponibile un piano di controllo SDN, provocando un'ampia indisponibilità della rete. Infine, nelle reti grandi o altamente dinamiche, i controller possono dover prendere veloci decisioni per tutti i nodi di rete dovendo scegliere fra

un'enorme varietà e frequenza di eventi, quali guasti, cambi della domanda di traffico, arrivo di nuovi flussi. Combinare SDN e le architetture tradizionali in dei modelli **SDN ibridi** ha il potenziale di sommare i loro vantaggi mitigando i rispettivi impedimenti. Infatti, i protocolli e le tecniche sviluppate nell'approccio tradizionale possono fornire soluzioni operative per alcune difficoltà riscontrate da SDN; modelli ibridi possono al tempo stesso favorire una graduale transizione da un CN a un paradigma SDN completo. Sono definibili quattro modelli SDN ibridi, ciascuno dei quali risulta particolarmente indicato per una specifica strategia di transizione e per determinate esigenze architetturali di certi operatori di rete. Ogni modello ibrido prevede una diversa interazione fra il controller SDN e il piano di controllo distribuito nei device non-SDN:

- **Topology-based:** questo modello si fonda su una separazione topologica dei nodi controllati da ciascun paradigma; la rete è partizionata in zone, ciascuna composta da una serie di nodi controllati dallo stesso paradigma.
- **Service-based:** in questo modello, CN e SDN forniscono diversi servizi, per cui i due paradigmi controllano una differente porzione delle interfacce di rete di ciascun nodo.
- **Class-based:** questo modello è basato sulla partizione del traffico in classi, ciascuna delle quali controllata da CN o SDN; i due paradigmi coinvolgono tutti i nodi della rete (contrariamente al *topology-based*), fornendo tutti i servizi di rete (contrariamente al *service-based*) alle loro classi di competenza.
- **Integrated:** rispetto ai modelli precedenti, in cui CN e SDN si complementano controllando porzioni delle interfacce di rete dei nodi, l'Integrated Hybrid SDN usa i protocolli CN come interfaccia verso i nodi, mentre SDN è responsabile di tutti i servizi di rete.

La soluzione proposta nel mio lavoro (illustrata al Cap. 4) è classificabile secondo il modello ibrido SDN *Service-Based*: il nodo sviluppato tramite Open vSwitch (cfr. 4.1) implementa le tradizionali funzioni di rete in modalità CN, mentre il protocollo LISP è realizzato *SDN-mode*, tramite la definizione di flussi OpenFlow. È stata testata l'interazione di tale sistema con un nodo CN - per l'esattezza OpenLISP, un software router che implementa il protocollo LISP. A lungo termine, tramite lo sviluppo di ulteriori protocolli legacy in modalità SDN, il nodo OVS può divenire, a tutti gli effetti, un nodo SDN; lo scenario convergerebbe dunque verso un *Topology-Based*, integrandosi come soluzione strategica complessiva per reti SDN ibride.

Capitolo 3

Locator/Identifier Separation Protocol

Questo capitolo illustra i principali elementi costitutivi del *Locator / Identifier Separation Protocol* (LISP), che per le sue caratteristiche è stato scelto come oggetto del nostro lavoro. Si tratta di un protocollo che opera a livello di rete, prevedendo una separazione dell'indirizzamento IP in due nuovi spazi di numerazione: gli Endpoint Identifier (EID) e i Routing Locator (RLOC); senza alcuna modifica ai protocolli preesistenti, tanto negli host quanto nel *core* dell'infrastruttura di Internet - dunque con la possibilità di essere implementato con estrema facilità - offre diverse soluzioni di *traffic engineering*, multihoming e mobilità, affrontando in maniera efficace il problema della scalabilità della rete. Per vie diverse dunque, l'approccio SDN e il protocollo LISP perseguono il medesimo obiettivo della semplificazione dell'instradamento; infatti, per i motivi illustrati in precedenza (Capitolo 2), con la possibilità esplosa di accedere a basso costo a capacità di rilevanti di processing e storage, la rete rischia ormai di divenire il collo di bottiglia dell'IT.

3.1 La questione della scalabilità della rete

"*La routing scalability costituisce oggi la questione più rilevante di Internet e deve essere risolta*". [1]
Il sistema di indirizzamento e instradamento di Internet non sta scalando sufficientemente rispetto alla crescita esplosiva di nuovi siti; una ragione va cercata nell'aumento

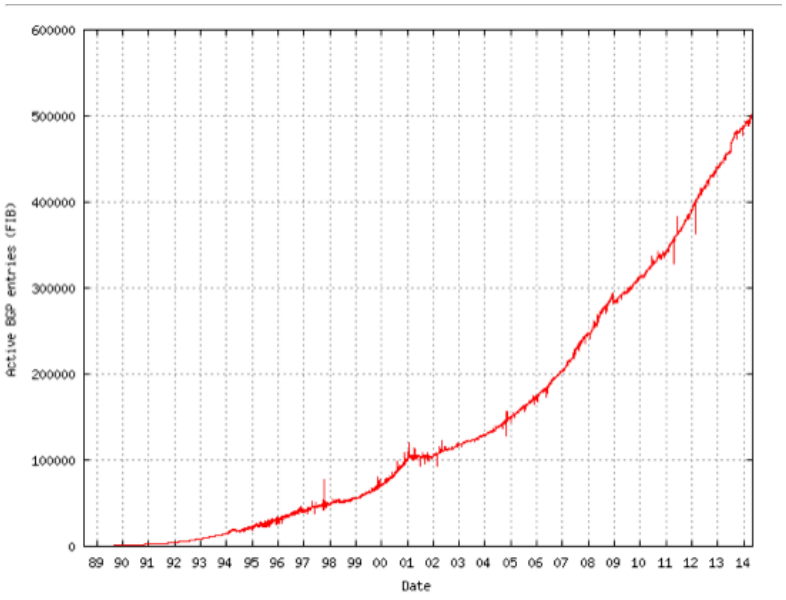

FIGURA 3.1: Incremento delle entries BGP.

dei siti *multihomed*[1] e di altri siti che non possono essere indirizzati come parte di prefissi aggregati in base alla topologia di rete o in base ai servizi forniti. Infatti, utilizzando un singolo indirizzo sia per identificare un device che per determinare dove sia topologicamente localizzato nella rete, si richiede un'ottimizzazione lungo due assi conflittuali:

- affinché il routing sia efficiente, gli indirizzi devono essere assegnati in funzione della topologia di rete;
- per far sì che una serie di device sia gestita in maniera semplice ed efficace, senza la necessità di essere re-indirizzati in risposta alle modifiche topologiche (come causato dall'aggiunta o dalla rimozione di punti di connessioni alla rete o da eventi di mobilità), l'indirizzo deve essere esplicitamente non collegato alla topologia.

L'approccio seguito da LISP per risolvere la questione della scalabilità del routing è quello di sostituire gli indirizzi IP con due nuovi tipi di numeri: gli RLOC, che sono topologicamente assegnati ai punti di allaccio alla rete (e così favorevoli a meccanismi di aggregazione), utilizzati per il routing e il forwarding dei pacchetti attraverso la rete; e gli EID, che vengono assegnati indipendentemente dalla topologia di rete per numerare i device e possono essere aggregati all'interno dei confini amministrativi. LISP

[1]Il Multihoming è una tecnica utilizzata per aumentare l'affidabilità di una rete IP; consiste generalmente nell'assegnare lo stesso spazio di indirizzamento assegnato a collegamenti multipli. Utilizzando un protocollo di routing - nella maggior parte dei casi Border Gateway Protocol (BGP) - il sito annuncia ai propri collegamenti lo spazio IP in uso; quando uno dei link viene meno, il protocollo notifica l'evento su entrambi i fronti, così il traffico non è più instradato sul link guasto.

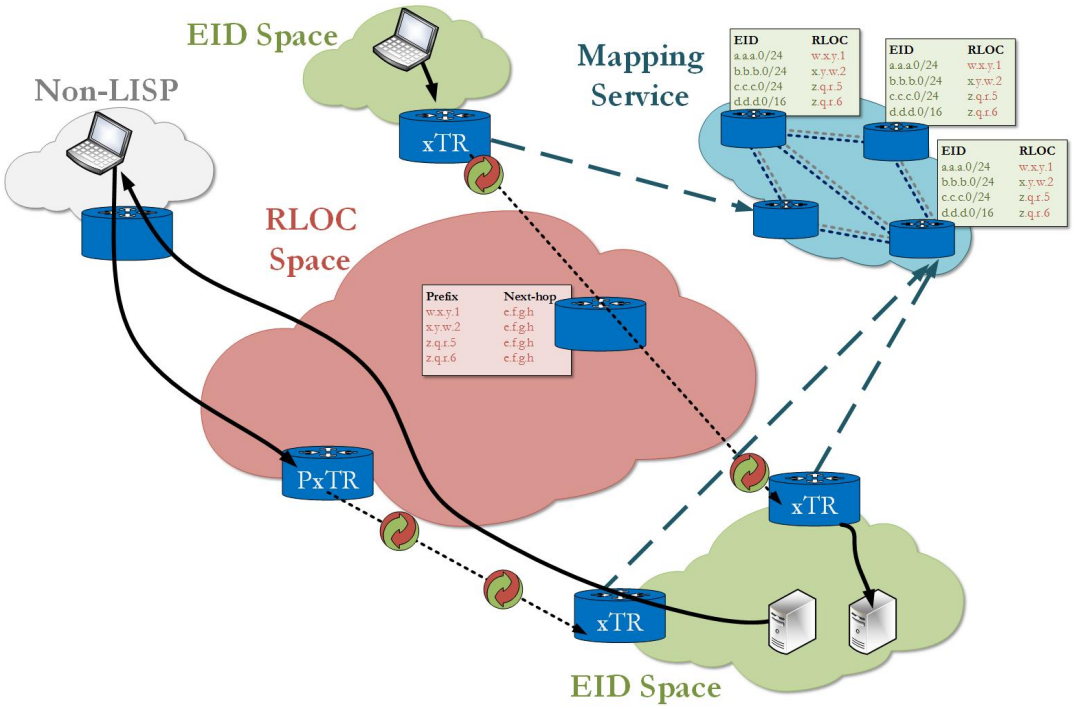

FIGURA 3.2: La rete LISP.

definisce allora funzioni di mapping fra i due spazi di indirizzamento e meccanismi per incapsulare il traffico, originato da quei device che utilizzano EID, non instradabile attraverso un'infrastruttura di rete che instrada e inoltra pacchetti utilizzando gli RLOC. Entrambi EID e RLOC sono sintatticamente identici agli indirizzi IP; differisce invece la semantica con cui essi vengono utilizzati.

3.2 Visione generale del protocollo [2]

Un concetto chiave di LISP è che gli end-system operano allo stesso modo di oggi: gli indirizzi IP utilizzati dagli host per tracciare socket e connessioni, per inviare e ricevere pacchetti, non cambiano. Nella terminologia LISP questi indirizzi IP sono chiamati **Endpoint Identifier (EID)**. Parliamo quindi in parole a 23 o 128 bit - a seconda che si usi IPv4 o IPv6 - utilizzate nei campi *source address* e *destination address* dell'header IP più interno. Ogni host ottiene un EID di destinazione allo stesso modo in cui oggi ottiene un indirizzo di destinazione, ad esempio attraverso un lookup DNS o uno scambio SIP; l'EID ottenuto attraverso i meccanismi esistenti è usato per impostare l'indirizzo IP "locale" dell'host. Gli host nemmeno sanno che esista un'associazione EID-RLOC; essi inviano pacchetti a EID di destinazione, assumendo che il pacchetto arrivi effettivamente alla destinazione da essi indicata. Un EID usato nella rete pubblica deve avere le stesse

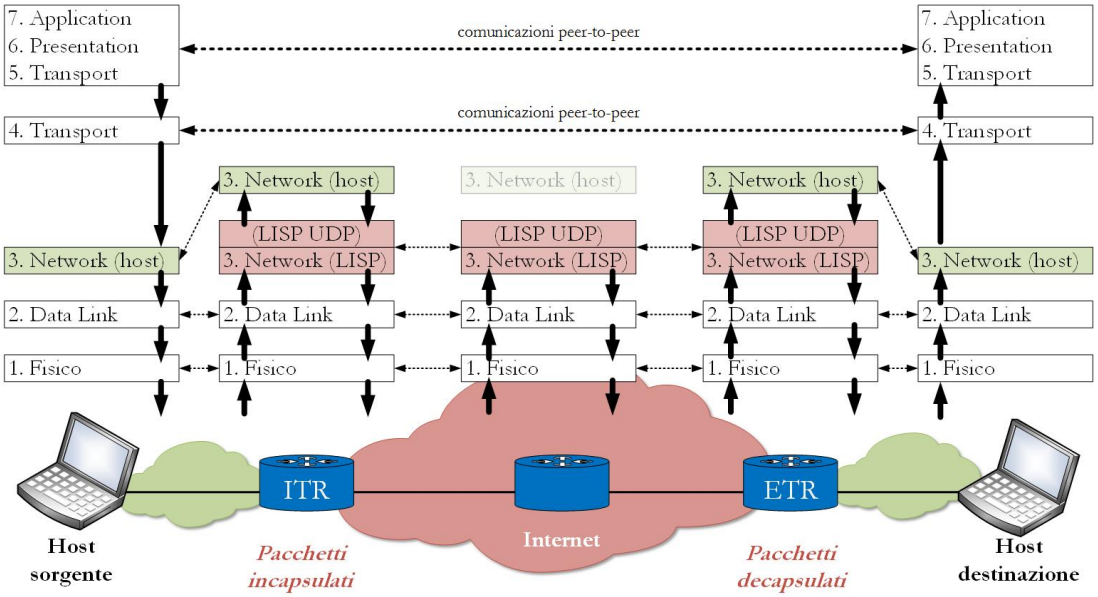

FIGURA 3.3: Visione OSI del protocollo LISP nei vari punti del collegamento fra host.

proprietà di qualsiasi altro indirizzo IP utilizzato alla stessa maniera; per cui, fra le altre cose, deve essere globalmente univoco. Gli EID possono essere assegnati in maniera gerarchica, indipendente dalla topologia di rete, per facilitare la scalabilità all'interno del mapping database; inoltre, un blocco EID assegnato a un sito potrebbe avere una sua strutturazione locale (subnetting) per l'instradamento all'interno del sito stesso.
Dal canto loro, i router continuano a inoltrare pacchetti in base agli indirizzi IP di destinazione; quando un pacchetto è incapsulato LISP, questi indirizzi sono indicati come **Routing Locator (RLOC)**. Gli RLOC sono infatti indirizzi IP assegnati ai router - preferibilmente su base topologica - dal fornitore di blocchi CIDR. La maggior parte dei router lungo il path fra due host non cambierà: essi continueranno a eseguire routing e forwarding ricercando gli indirizzi di destinazione. Modificheranno invece il loro comportamento i router presenti nel LISP site, cioè nel bordo fra la edge network e la core network; essi infatti costituiscono il punto di demarcazione fra la porzione di rete nell'edge che viene chiamata EID Space e il core della rete che diviene l'RLOC Space. Questi Tunnel Router sono di due tipi:

- **Ingress Tunnel Router (ITR)**: ricevendo un pacchetto dall'interno dell'EID space tratta l'indirizzo IP di destinazione come un EID ed esegue una ricerca del mapping EID-RLOC, al fine di determinare il percorso di instradamento. L'ITR allora aggiunge al pacchetto un header IP esterno, contenente nel campo *source address* uno dei suoi RLOC globalmente instradabili e nel campo *destination address* il risultato della ricerca effettuata circa l'associazione EID-RLOC.

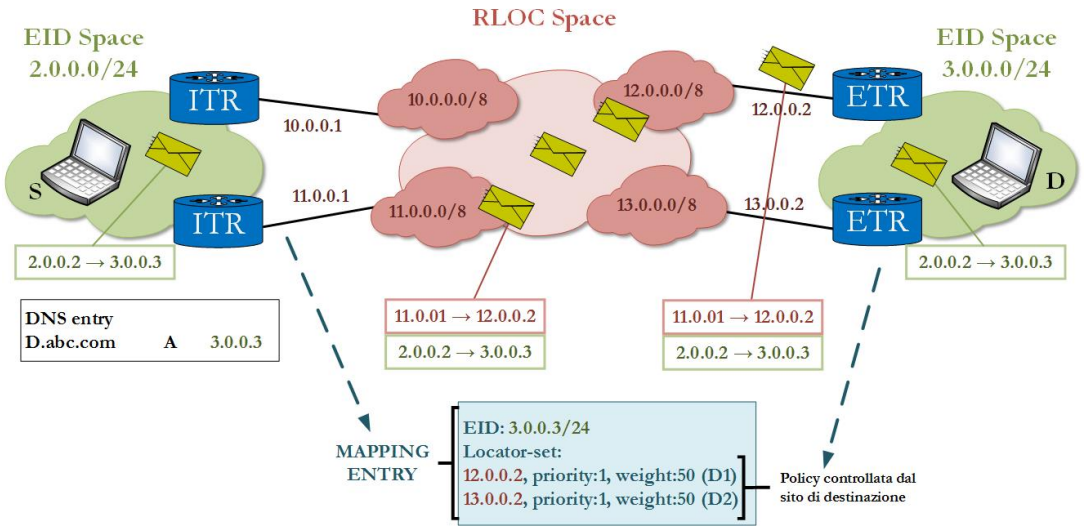

FIGURA 3.4: Forwarding di un pacchetto unicast.

- **Egress Tunnel Router (ETR)**: accetta pacchetti la cui destination address nell'header IP esterno è uno dei suoi RLOC. L'ETR estrae l'header esterno e inoltra i pacchetti basandosi sul successivo indirizzo IP di destinazione trovato (ovvero l'EID contenuto nel campo *destination address* dell'header IP interno). Sostanzialmente l'ETR riceve "lato Internet" pacchetti IP incapsulati LISP che invia decapsulati sull'altro lato, verso gli host.

Dunque, per i router fra l'host sorgente e l'ITR, così come fra ETR e gli host di destinazione, il *destination address* è l'EID; per i router fra ITR e ETR, il destination address è l'RLOC. Tipicamente gli RLOC sono assegnati in funzione di blocchi aggregabili topologicamente, che sono assegnati a un sito per tutti i punti in cui esso si allaccia alla rete globale.

3.2.1 Sequenza di flusso di pacchetti

Forniamo un esempio di flusso di pacchetti unicast, nel seguente contesto:

- L'host sorgente "*host1.abc.esempio.it*" deve inviare un pacchetto a "*host.2.xyz.esempio.it*"; esattamente quello che *host1* farebbe se non stesse usando LISP.
- Ciascun sito è *multihomed*, quindi ogni xTR ha un RLOC assegnato dal service provider.
- Sia ITR che ETR sono direttamente connessi rispettivamente alla sorgente e alla destinazione, ma queste possono essere localizzate ovunque all'interno dell'EID space.

L'*host1* vuole comunicare con l'*host2*:

1. *host1.abc.esempio.it* effettua un DNS lookup su *host2.xyz.esempio.it*, ricevendo un EID come indirizzo di destinazione. *host1.abc.esempio.it*, utilizzando il proprio EID sorgente, genera un pacchetto IP e lo inoltra secondo i normali meccanismi attraverso l'EID space, fino a raggiungere il LISP ITR.

2. L'ITR deve mappare l'EID di destinazione del pacchetto con l'RLOC dell'ETR del sito di destinazione: invia dunque un messaggio di controllo LISP di tipo **Map-Request**, in attesa di ricevere un messaggio **Map-Reply** che comunichi l'associazione EID-RLOC. Nel paragrafo 3.4 sarà analizzato questo aspetto con maggiore dettaglio.

3. ITR riceve il messaggio Map-Reply e memorizza l'informazione nella sua *mapping cache*.

4. I successivi pacchetti inviati da *host1.abc.esempio.it* a *host2.xyz.esempio.it* avranno un header LISP aggiunto da ITR, contenente l'RLOC di destinazione appropriato.

5. L'ETR riceve questi pacchetti direttamente, rimuove l'header LISP e inoltra i pacchetti al relativo host di destinazione, utilizzando l'header IP interno (dunque l'EID di destinazione). Eventualmente ETR può creare una map-cache fra i source EID-RLOC dei pacchetti ricevuti, col fine di agevolare il mapping lookup inverso.

3.3 Incapsulamento LISP [3]

Analizziamo ora i meccanismi di incapsulamento e decapsulamento operati dagli xTR mediante l'aggiunta e la rimozione di un header LISP. Dal momento che EID e RLOC possono essere sia IPv4 che IPv6, l'architettura LISP supporta le quattro possibili combinazioni del formato pacchetti:

1. IPv4 EID con IPv4 RLOC
2. IPv4 EID con IPv6 RLOC
3. IPv6 EID con IPv4 RLOC
4. IPv6 EID con IPv6 RLOC

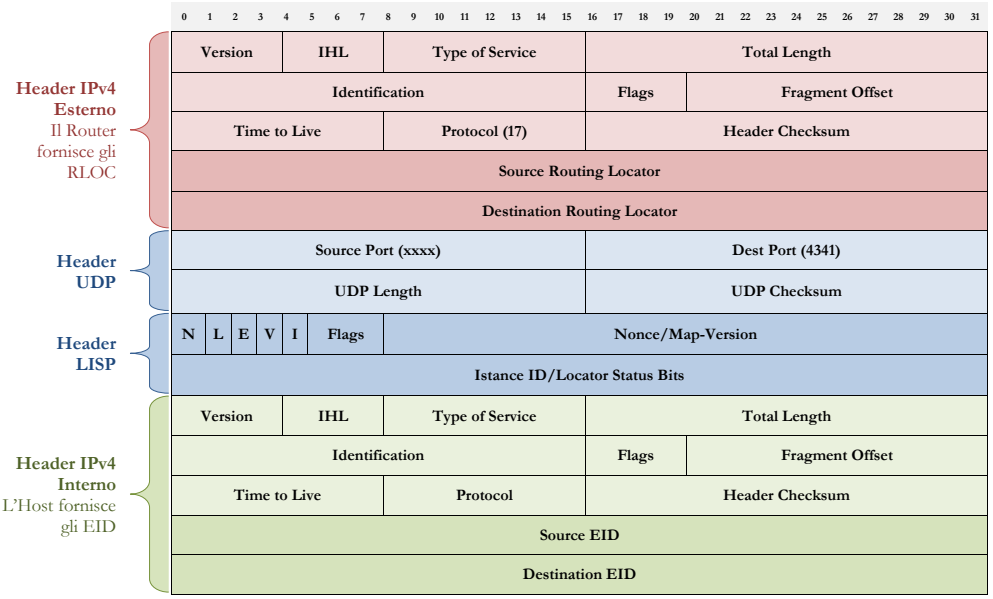

FIGURA 3.5: Formato dell'header LISP nel caso omogeneo IPv4-in-IPv4.

A fini esemplificativi, faremo riferimento esclusivamente al formato 1. dell'header LISP. Consideriamo un pacchetto incapsulato LISP in ricezione all'ETR. L'**inner header** è l'header del datagramma ricevuto dall'host originario; gli indirizzi IP sorgente e destinazione sono EID. A questo viene aggiunto innanzitutto un **header** tipico del protocollo **LISP**, composto dai seguenti campi:

- **N**: si tratta di un bit di presenza del *nonce*; quando è settato a 1, gli ultimi 24 bit dei primi 32 bit dell'header LISP contengono un Nonce.
- **L**: questo bit, se settato a 1, attiva il campo Locator-Status-Bit negli ultimi 32 bit dell'header LISP.
- **E**: quando N=1 e questo bit è settato a 1, un ITR richiede che il valore Nonce sia inviato indietro; il valore di E si ignora quando N=0.
- **V**: quando V=1, gli ultimi 24 bit dei primi 32 bit dell'header LISP sono utilizzati per la Map-Version; di conseguenza N=0 e il Nonce non è utilizzato.
- **I**: se I=1, il campo Locator-Status-Bits è ridotto a 8 bit e i primi 24 bit degli ultimi 32 bit sono usati come Istance ID. Se L=0, gli ultimi 8 bit sono posti a 0 e ignorati in ricezione.

- **flags**: si tratta di un campo a 3 bit riservato per usi futuri (per ora settato a 0).
- **Nonce / Map-Version**:
  - **Nonce**: campo a 24 bit generato randomicamente dall'ITR quando N=1 e reinviato dall'ETR; serve a scopi di TE per utilizzare in direzione opposta lo stesso path utilizzato.
  - **Map-Version**: campo che identifica la versione dell'associazione EID-RLOC, che, in caso di variazione rispetto a uno stato precedente, aggiorna la mapping cache degli xTR.
- **Istance ID / Locator-Status-Bits**:
  - **Istance ID**: campo a 24 bit utilizzato dall'ITR per identificare univocamente lo spazio di indirizzi usato, qualora nel medesimo sito LISP organizzazioni diverse utilizzino indirizzi privati.
  - **Locator-Status-Bits**: questo campo a numero variabile di bit è usato dall'ITR per informare gli ETR circa lo status (up/down) di tutti gli ETR locali.

All'header LISP viene anteposto un **header UDP**, che contiene una *source port* selezionata dall'ITR al momento dell'incapsulamento. La *destination port* invece è impostata al valore 4341 assegnato dall'IANA. Il campo *UDP Length* indica la lunghezza totale dell'header di un pacchetto incapsulato, dunque pari alla somma delle lunghezze degli header IH, UDP e LISP. Il campo *UDP Checksum* dovrebbe essere trasmesso dall'ITR con valore 0; in caso contrario, l'ETR può scegliere di eseguire una verifica sul valore trasmesso: qualora non fosse corretto, il pacchetto sarà scartato e non decapsulato.

Infine viene preposto dall'ITR un **outer header**: è un nuovo header, del tutto simile all'IH, ovvero al tradizionale header IP; i campi *source address* e *destination address* contengono RLOC ottenuti dalla EID-RLOC cache. Il campo *Protocol* è settato a 17, ovvero il valore identificativo di UDP, che, come visto precedentemente, è il protocollo di trasporto utilizzato a supporto di LISP.

3.4 Mapping Service EID-RLOC

Per consentire la separazione degli spazi di indirizzamento, si rende necessaria la definizione di meccanismi protocollari che eseguano il mapping fra EID e RLOC. Il LISP Mapping Service definisce due nuovi tipi di device LISP:

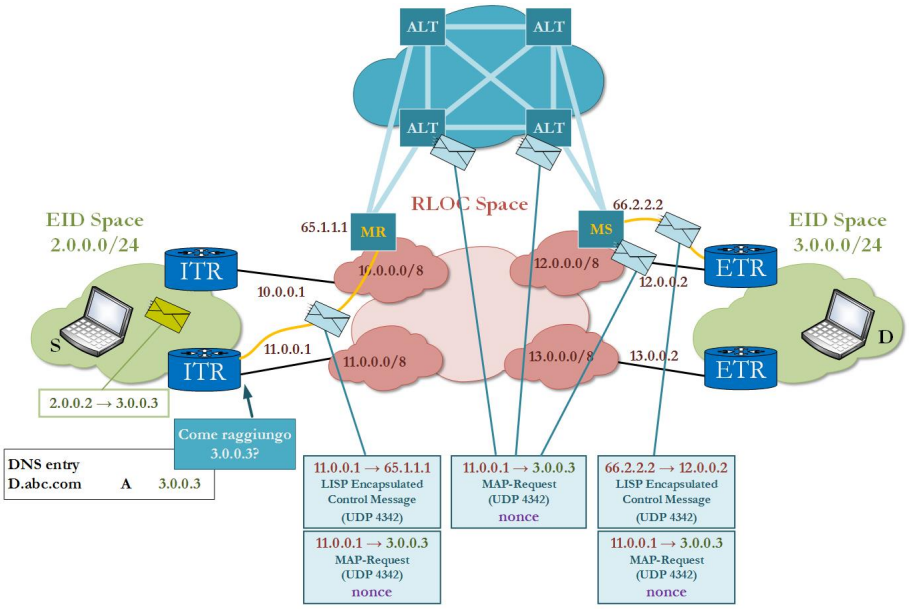

FIGURA 3.6: Esempio di invio di un Map-Request.

- il **Map-Server**, un componente dell'infrastruttura di rete che acquisisce dagli ETR le informazioni di mapping EID-RLOC contenute nel Mapping Database; i LISP Map-Server sono concettualmente simili ai server DNS.
- il **Map-Resolver**, che accetta messaggi Map-Requests dall'ITR e determina - qualora sia presente - l'appropriata associazione EID-RLOC consultando il Mapping Server; analogamente, sono concettualmente simili ai DNS resolver.

Vengono utilizzati 4 tipi di messaggi di controllo LISP per recuperare, risolvere e comunicare le informazioni di mapping EID-RLOC:

- **Map-Request**: messaggio basato su UDP (*source port*=random, *destination port*=4342), inviato dall'ITR al Map Resolver quando necessita di conoscere l'RLOC associato a un determinato EID, vuole testare la raggiungibilità di un RLOC, oppure quando vuole aggiornare un mapping prima della scadenza del TTL. Nel primo caso, l'IP di destinazione del pacchetto Map-Request è il destination EID del pacchetto dati, mentre negli altri due si utilizza uno degli RLOC nel Locator-Set della Map-Cache. Presenta un nonce utile per la replica del Map Resolver.
- **Map-Reply**: messaggio basato su UDP (*source port*=4342, *destination port*=source port del Map-Request) in risposta al Map-Request; contiene gli RLOC associati all'EID richiesto e viene inviato da un ETR all'ITR richiedente.
- **Map-Register**: messaggio inviato dall'ETR al Map-Server per registrare gli EID associati al suo RLOC.

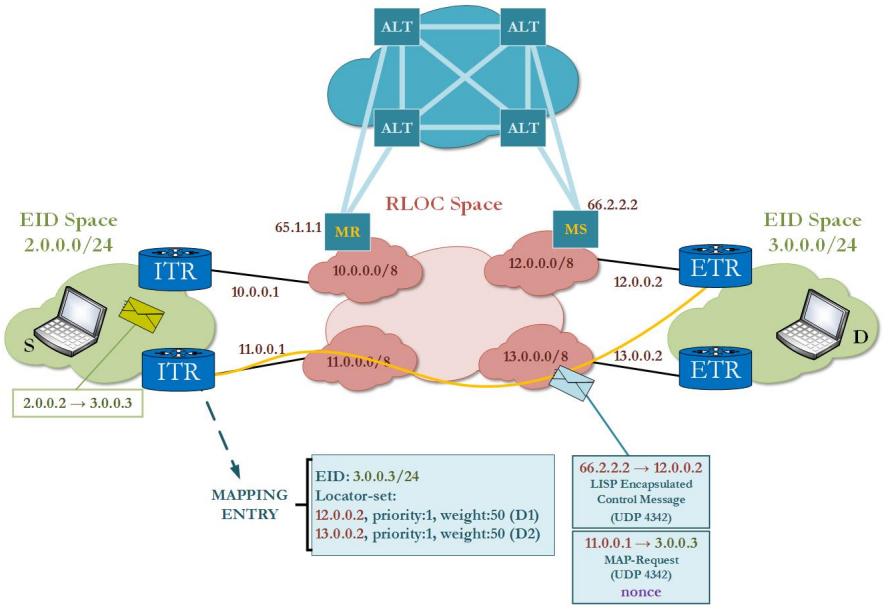

FIGURA 3.7: Esempio di invio di un MAP-Reply.

- **Map-Notify**: messaggio inviato dal Map-Server all'ETR per confermare la ricezione ed elaborazione del messaggio Map-Register.

Il piano di controllo di LISP si esprime allora attraverso due processi, ciascuno dei quali coinvolge un xTR, uno dei device del Mapping Service e lo scambio di 2 dei 4 messaggi di controllo "Map" di cui sopra.

3.4.1 Risoluzione del Mapping EID-RLOC

Un ITR è configurato con uno o più indirizzi (RLOC) del Map-Resolver, raggiungibili sulla rete senza la necessità di essere risolti attraverso il mapping EID-RLOC, che introdurrebbe una dipendenza circolare (il Mapping Service deve quindi trovarsi nel RLOC space). Quando l'ITR necessita di risolvere un'associazione EID-RLOC non presente nella sua map-cache locale, invia un Map-Request incapsulato LISP a un Map-Resolver configurato; tale messaggio è, decapsulato, instradato nella rete alternativa del Mapping Service [2] fino al Map-Server, dal quale è inoltrato nuovamente incapsulato LISP fino all'ETR di destinazione. L'ITR si aspetta in risposta uno dei seguenti messaggi:

[2] Alternative Logical Topology (ALT) è un semplice sistema distribuito di indice, usato dal LISP Map-Resolver per determinare l'ETR che contiene le informazioni di mapping per un particolare EID. L'indice è costruito come una rete sovrapposta all'Internet pubblico, realizzata tramite i protocolli BGP e GRE. [4]

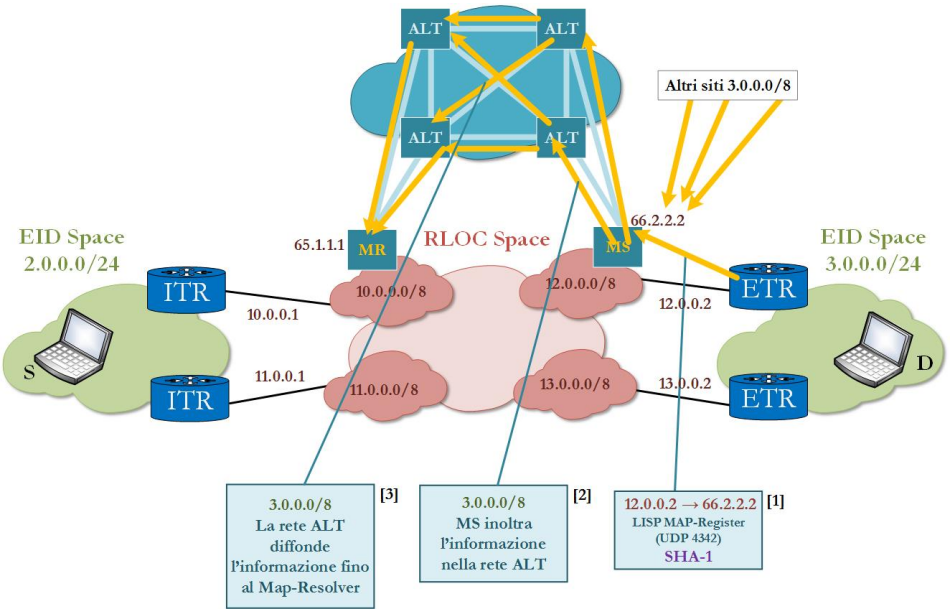

FIGURA 3.8: Esempio di registrazione di un RLOC tramite Map-Register.

- Un immediato Map-Reply negativo dal Map-Resolver, se questi determina che l'EID richiesto non esiste. L'ITR salva l'EID nella sua cache marcandolo come "*non-LISP-capable*", apprendendo di non eseguire incapsulamento LISP per i pacchetti che hanno esso come destinazione.
- Un Map-Reply negativo dal Map-Server, se questi trova l'EID fra le entry del Mapping Database ma a esso non corrisponde un RLOC; in tal caso si attiva una procedura di TTL per l'interrogazione e l'eventuale rimozione della entry.
- Un LISP Map-Reply risolutivo dell'associazione, inviato dall'ETR stesso che conosce il mapping in esame.

3.4.2 Configurazione degli EID e registrazione degli ETR

Un ETR pubblica gli indirizzi del suo EID space sul Map-Server inviando periodicamente messaggi Map-Register (con un intervallo suggerito di un minuto fra i messaggi); dal canto suo il Map-Server dovrebbe rimuovere le associazioni nel Database di cui non riceve aggiornamento ogni 3 minuti. Un ETR potrebbe richiedere al Map-Server - settando un flag nel Map-Request - di inviare una notifica esplicita circa la ricezione e l'elaborazione del Map-Request stesso; in tal caso il Map-Server risponde con l'invio di un messaggio Map-Notify. Il Map-Server diffonde le informazioni nella rete del Mapping Service, fino a raggiungere anche il Map-Resolver.

3.5 Meccanismi di *internetworking* coi siti non-LISP

Per garantire l'operabilità della rete fra i siti LISP e quelli che non supportano il protocollo, sono stati pensati dei meccanismi di **Proxy xTR**:

- **Proxy Ingress Tunnel Router (PITR)**: ricevono il traffico dai siti non-LISP, incapsulandolo verso i siti LISP (Figura 3.9).
- **Proxy Egress Tunnel Router (PETR)**: svolgono la funzione inversa rispetto ai PITR. (Figura 3.10).

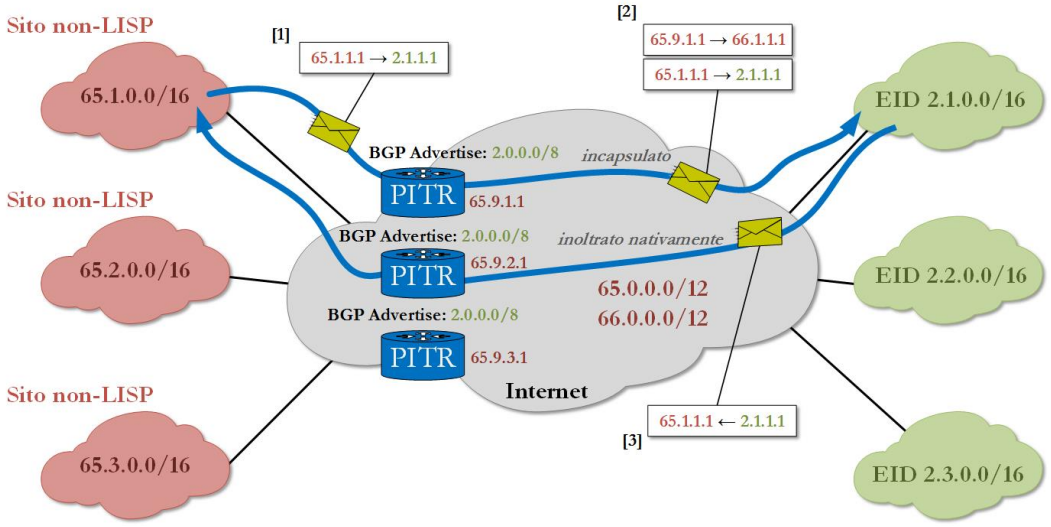

FIGURA 3.9: Traffico IP dai siti non-LISP verso i siti LISP.

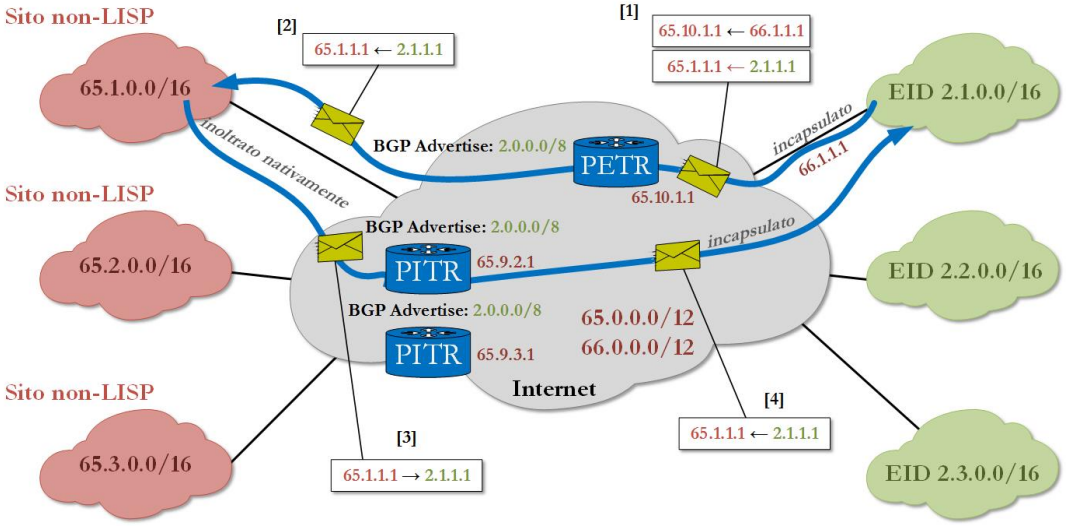

FIGURA 3.10: Traffico IP dai siti non-LISP verso i siti LISP.

3.6 Casi d'uso del protocollo LISP

I due livelli di indirizzamento consentono di tenere fissi gli EID mentre gli RLOC sono soggetti a variabilità; in questo modo gli EID non necessitano di essere rinominati e le connessioni TCP possono essere mantenute durante i trasferimenti delle macchine. Modificare RLOC invece risulta utile nel cambio di service provider, nella mobilità degli host e nella riallocazione di VM. In questo paragrafo analizziamo qualche *use case* che mostra particolarmente i vantaggi portati dal protocollo LISP.

Multi Homing efficiente: La maggior parte dei siti oggi necessita di connettersi a provider multipli al fine di aumentare l'affidabilità delle connessioni, pur mantenendo bassi i costi operativi e in conto capitale. LISP fornisce una soluzione flessibile per la gestione di connettività e politica multi-provider, senza la complessità del BGP. LISP consente la ramificazione di siti dove il multihoming è tipicamente troppo caro. (Figura 3.11)

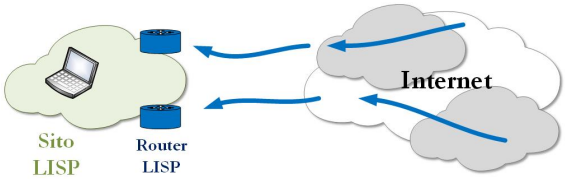

FIGURA 3.11: Scenario Multi Homing per un sito LISP.

Supporto alla transizione IPv6: Il bisogno di un rapido impiego di IPv6, attraverso l'utilizzo di un'infrastruttura quanto più minimale possibile, trova in LISP un valido supporto; infatti l'incapsulamento LISP è agnostico rispetto alla famiglia di indirizzi. Questo chiaramente provoca un'accelerazione dell'adozione di IPv6 mediante l'aggiunta di una configurazione minimale, che può essere utilizzata come soluzione di transizione o permanente. (Figura 3.12)

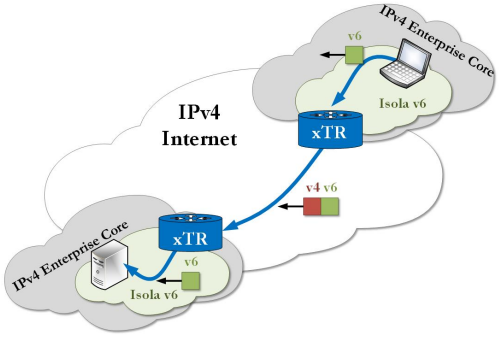

FIGURA 3.12: Interoperabilità IPv4/IPv6 tramite LISP.

Multi-Tenancy: LISP ha un ruolo notevole nella semplificazione delle funzioni di VRF in un'architettura multitenant; tramite l'utilizzo del campo a 24-bit *Istance-ID* dell'header LISP è possibile infatti mantenere univoche le associazioni di mapping nel control-plane, qualora si utilizzino indirizzi EID duplicati, com'è facile in ambiente virtualizzato. La segmentazione degli spazi di indirizzi è dunque integrata - e dunque portata a scala elevata - in una soluzione IP, senza alcun coinvolgimento del piano di trasporto. La *Network Virtualization* è realizzata "Over the Top", in maniera trasparente rispetto al *core* della rete. (Figura 3.13)

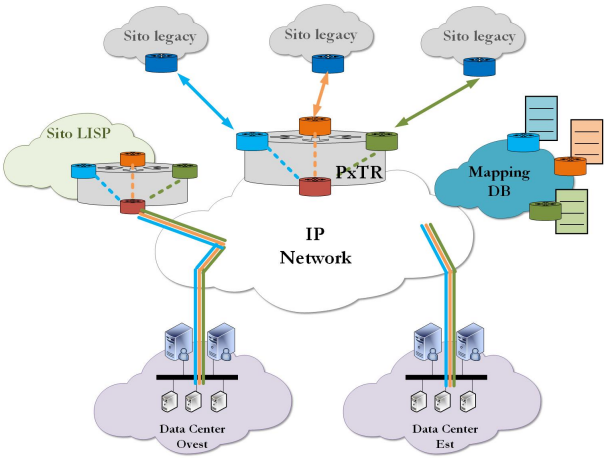

FIGURA 3.13: Scenario Multi-Tenancy per un sito LISP.

Mobilità delle VM: Una delle applicazioni più rilevanti in cui trova effetto il protocollo LISP riguarda la mobilità delle macchine virtuali fra *subnet*. Il concetto di mobilità è integrato nel protocollo, laddove sono distinti gli indirizzi che identificano gli host e quelli che lo localizzano nella rete. In caso di trasferimento di VM non devono essere operate nuovamente configurazioni di routing e DNS: il mapping dinamico EID-RLOC realizza un'automatica ridefinizione dei path, consentendo che le connessioni attive siano mantenute attraverso gli spostamenti delle VM. (Figura 3.14)

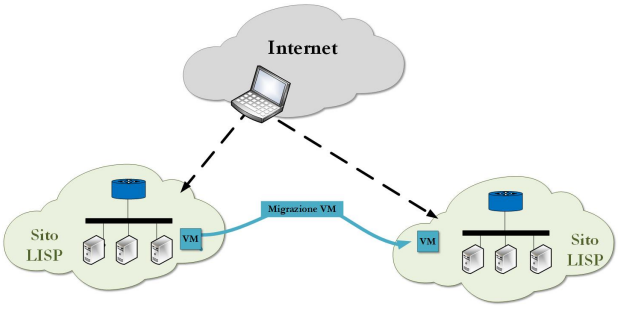

FIGURA 3.14: Mobiilità delle VM semplificata dall'utilizzo di LISP.

LISP Mobile Node: I vantaggi di mobilità apportati da LISP in campo VM possono ripercuotersi in ambiente cellulare. I device mobili infatti necessitano di switchare da una rete all'altra (3G, 4G, Wi-Fi, ecc.) senza reset della connessione: necessitano pertanto di un indirizzo IP permanente - che LISP sa offrire con gli EID - e al tempo stesso di un accesso dinamico alla rete, realizzabile tramite RLOC. (Figura 3.15)

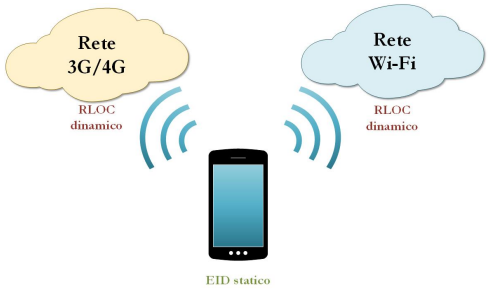

FIGURA 3.15: LISP Mobile Node.

Capitolo 4

Implementazione del piano dati LISP in uno scenario SDN

In questo capitolo analizzeremo il *core* del nostro lavoro, che si è concentrato sull'implementazione delle funzioni di forwarding del protocollo LISP in uno scenario SDN. Proponiamo un sistema convergente, in cui il protocollo LISP trova facile possibilità di impiego grazie al suo sviluppo in ambiente OpenFlow; tuttavia, nessun dispositivo hardware OF è stato utilizzato per la nostra ricerca. La configurazione è stata realizzata tramite l'utilizzo del software Open vSwitch, con l'obiettivo di offrire una soluzione operativa, accessibile in termini di costi, per il deployment di nuove funzionalità di rete.

4.1 Open vSwitch

4.1.1 Caratteristiche e Componenti [5]

Open vSwitch (OVS) è un software switch multilayer soggetto a licenza open source *Apache* 2; ha l'obiettivo di implementare la produzione di una piattaforma switch qualificata, che supporti interfacce standard di management e introduca estensioni e controllo programmatici per le funzioni di forwarding. OVS supporta diverse tecnologie di virtualizzazione basate su Linux, incluse *Xen/XenServer*, *KVM* e *VirualBox*. Il cuore del codice è scritto in *platform-independent* C ed è facilmente portabile in altri ambienti. La versione corrente di OVS - la 2.1.2, utilizzata nella nostra implementazione - supporta le seguenti caratteristiche:

- VLAN (standard 802.1Q) con porte access e trunk
- Bonding delle NIC con o senza LACP

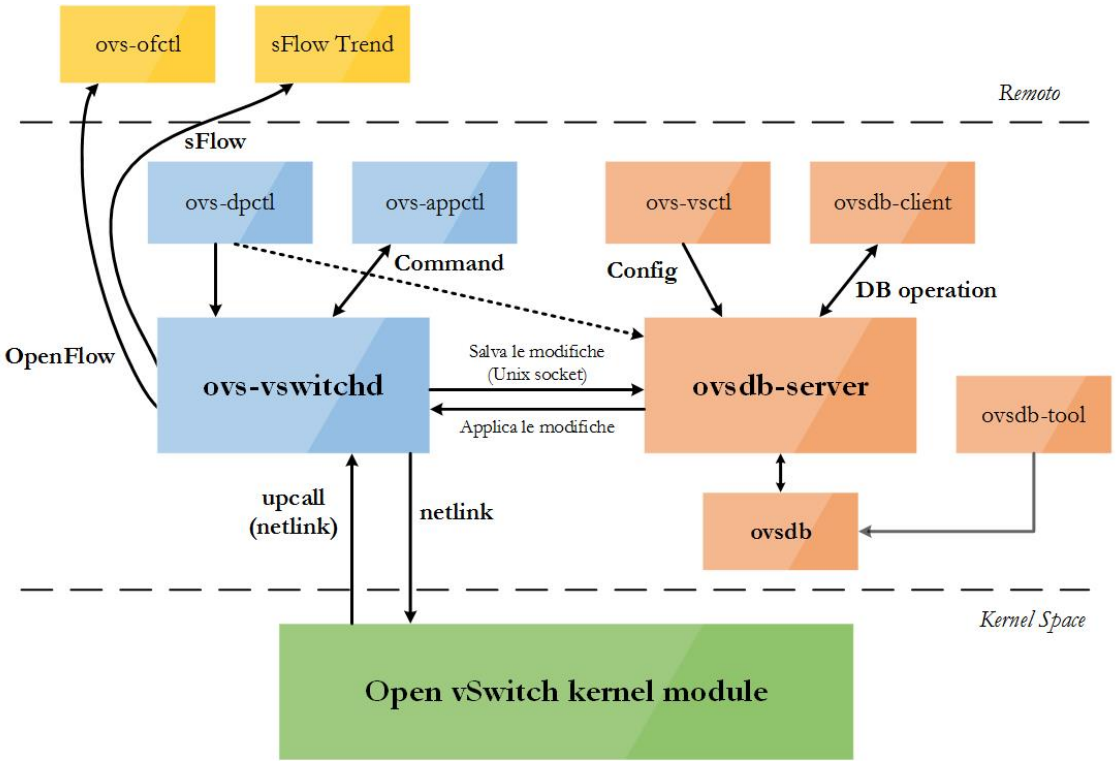

FIGURA 4.1: Interazione dei componenti di Open vSwitch.

- NetFlow, sFlow e mirroring[1] per un'aumentata visibilità
- Configurazioni QoS e policing
- GRE[2], GRE su IPSEC, VXLAN[3] e **LISP** tunneling
- Protocollo 802.1ag per la *connectivity fault management*
- OpenFlof v1.0 e numerose estensioni
- Configurazione transazionale del database, con collegamenti C e Python
- Forwarding a elevate prestazioni tramite l'utilizzo del modulo del kernel Linux

OVS può operare anche, a discapito delle performance, interamente nello userspace, senza l'assistenza di un modulo del kernel; questa implementazione - ancora sperimentale - dovrebbe rendere più semplice la portabilità rispetto alla versione basata su kernel [6]. Le principali componenti di questa distribuzione sono:

[1]NetFlow e sFlow sono strumenti di monitoraggio del traffico di rete, sviluppati da Cisco e altri

[2]Il Generic Routing Encapsulation è un protocollo di rete progettato nel 1994, in grado di incapsulare fino a venti tipi di protocolli

[3]Tecnologia di virtualizzazione che tenta di risolvere problemi di scalabilità associati con vasti ambienti di cloud computing; usa una tecnica di incapsulamento sul modello VLAN per incapsulare frame ethernet basati su MAC (livello 2) all'interno di pacchetti UDP

1. **ovs-vswitchd**, un demone che implementa le funzionalità dello switch, insieme al modulo del kernel Linux per la commutazione basata su flusso **(piano dati o di forwarding)**. Il primo pacchetto di un flusso è processato nello userspace, dal demone vswitchd, per la decisione sul forwarding; questi restituisce i dati del path al modulo del kernel che esegue il forwarding dei successi pacchetti del flusso (per cui il primo pacchetto ha un processing pi lento dei successivi).

2. **ovsdb-server**, un leggero database server interrogato da ovs-vswitchd per ottenere la sua configurazione.

3. **ovs-dpctl**, un tool per la configurazione del modulo del kernel switch.

4. **ovs-vsctl**, un'utility per interrogare e aggiornare la configurazione di ovs-vswitchd.

5. **ovs-appctl**, un'utility che invia comandi ai demoni OVS in esecuzione.

6. **ovs-ofctl**, un tool per interrogare e controllare gli switch e i controller OF.

7. **ovs-pki**, per la creazione e la gestione dell'infrastruttura a chiave pubblica per gli switch OF.

8. Una patch per *tcdump*, che lo abilita al parsing dei messaggi OF.

4.1.2 Architettura logico-funzionale di rete

Il demone **ovs-vswitchd** gestisce e controlla qualsiasi numero di switch OVS sulla macchina locale. La configurazione del vswitch avviene secondo il modello client-server, ovvero per interrogazione di ovsdb-server, al quale si può connettere in modalità attiva o passiva tramite SSL, TCP o Socket di dominio locale Unix (opzione di default). Ovs-vswitchd ricava la sua configurazione dal database all'avvio: imposta i datapath di OVS e opera la commutazione tra ciascun bridge descritto nei suoi file di configurazione; come il database cambia, ovs-vswitchd automaticamente aggiorna la sua configurazione. Lo switch ovs-vswitchd può essere configurato con una delle caratteristiche illustrate precedentemente. Può essere eseguita solo una singola istanza di ovs-vswitchd alla volta. Un singolo processo ovs-vswitchd può gestire qualsiasi numero di istanze switch, fino al numero massimo di datapath Open vSwitch supportati.

ovsdb-server, è un processo che fornisce interfaccia RPC verso uno o più database Open vSwitch (OVSDB) e realizza la configurazione del vswitch. L'OVSDB file deve essere specificato da linea di comando: `ovsdb-server` *`[database]`* Sono supportate connessioni JSON al client attive o passive con SSL, TCP o Socket di dominio locale Unix (opzione di default).

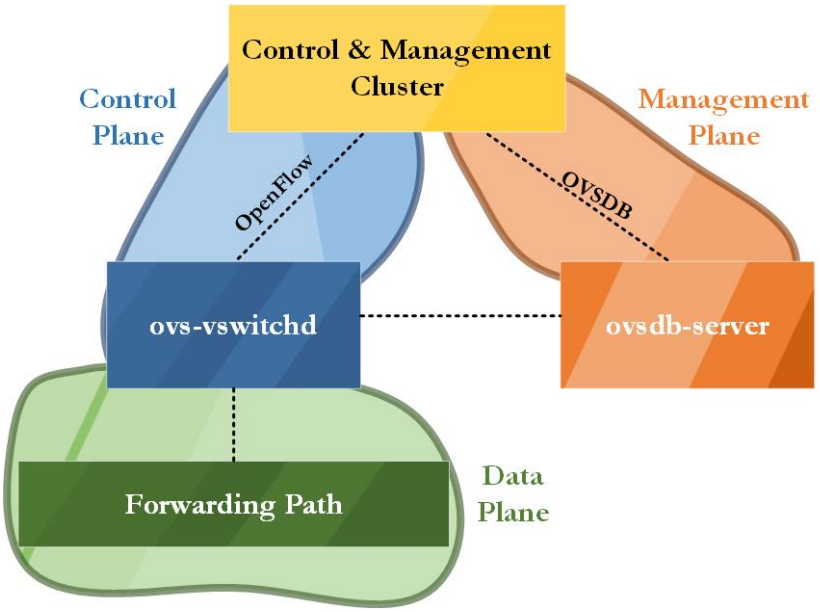

FIGURA 4.2: Architettura funzionale di Open vSwitch.

Il modulo del kernel OVS **openvswitch_mod.ko** gestisce la commutazione e le funzioni di tunneling; esso non sa nulla di OpenFlow. Infatti, la decisione su come processare i pacchetti ricevuti per la prima volta è presa nell'userspace (ovs-vswitchd), mentre i successivi pacchetti trovano una entry memorizzata nel kernel: di conseguenza, sono inoltrati direttamente e più velocemente.

Open vSwitch utilizza pertanto due differenti canali di interazione:

1. Il protocollo OpenFlow (vedi paragrafo 2.3) realizza il **piano di controllo** (o segnalazione) di Open vSwitch, ovvero definisce le operazioni da effettuare con i flussi, come devono essere inoltrati i pacchetti; non fornisce però le funzioni di gestione necessarie per allocare porte o assegnare indirizzi IP.

2. OVSDB realizza il **piano di management** di Open vSwitch, mediante il processo ovsdb-server che implementa la configurazione di ovs-vswichtd; stabilisce pertanto in maniera standardizzata i parametri per la comunicazione fra controller e vSwitch. [7]

Il piano dati, ovvero le operazioni relative al forwarding dei pacchetti tra device, nel contesto SDN sono appannaggio dello switch, nel nostro caso del demone ovs-vswitchd. Il piano di controllo utilizza OpenFlow per costruire la tabella di forwarding utilizzata dal piano dati; la tabella di forwarding è consegnata al piano dati dal piano di management (OVSDB).

4.1.3 OVSDB Schema [8]

La creazione del database avviene leggendo l'OVSDB schema dal file `vswitchd/vswitch.ovsschema` e producendo un nuovo OVSDB database file `/usr/local/etc/openvswitch/conf.db` che usi tale schema; quest'ultimo file contiene la configurazione per il demone ovs-vswitchd. La seguente lista sintetizza le tabelle previste nel DB schema e la loro funzione:

- **Open_vSwitch**: La configurazione di primo livello per il demone è la tabella Open_vSwitch, che deve avere esattamente un record; i record nelle altre tabelle sono significativi solo quando sono raggiunti direttamente o indirettamente dalla tabella Open_vSwitch. I record che non sono raggiungibili dalla tabella Open_vSwitch sono automaticamente eliminati dal database, ad eccezione dei record appartenenti alle tabelle del root set.
- **Bridge**: Configurazione di un bridge; un record della tabella Bridge rappresenta uno switch Ethernet con una o più "porte", che sono i record della tabella Port indicati dalla colonna ports della tabella Bridge.
- **Port**: Configurazione delle porte della tabella Bridge. Più comunemente una porta ha esattamente un'interfaccia, indicata dalla sua colonna interfaces. Una porta con più di un'interfaccia è una "bonded port".
- **Interface**: Interfaccia all'interno di una Porta.
- **Flow_Table**: Configurazione di una particolare tabella OpenFlow.
- **QoS**: Configurazione della Qualità del Servizio per ogni Port cui si riferisce.
- **Queue**: Configurazione per la coda di output di una porta, usata nella configurazione delle caratteristiche QoS. Potrebbe essere referenziata dalla colonna queues nella tabella QoS.
- **Mirror**: Una porta mirror all'interno del Bridge. Una porta mirror configura un bridge per inviare frame selezionati a speciali porte mirrorate, in aggiunta alle loro normali destinazioni.
- **Controller**: Configurazione del controller OpenFlow; Open vSwitch ne supporta due tipi: Primary controllers e Service controllers.
- **Manager**: Configurazione per una connessione al client OVSDB. Questa tabella configura l'open vSwitch database (ovsdb-server), non l'Open vSwitch database

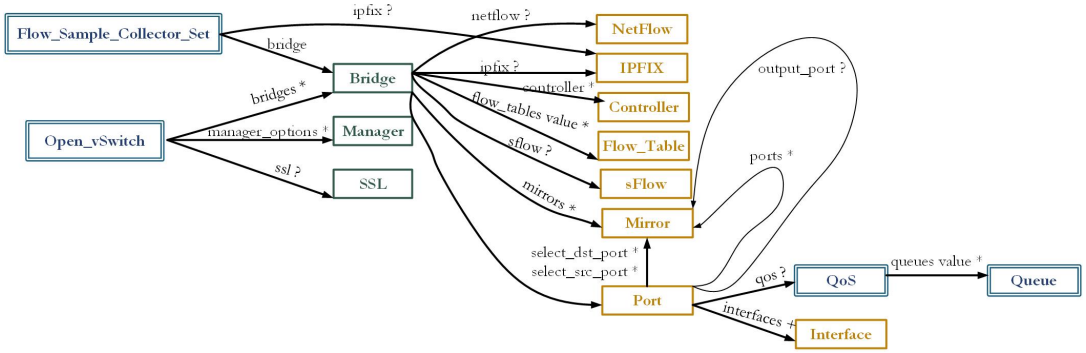

FIGURA 4.3: Database Schema di Open vSwitch.

(ovs-vswitchd). Lo switch fa leggere questa tabella per determinare quali connessioni debbano essere trattate come in-band (dati e flussi di controllo sugli stessi path).

- **NetFlow**: Un target NetFlow. NetFlow è un protocollo che esporta un numero di dettagli sulla terminazione di flussi IP, come i protagonisti coinvolti e la durata.
- **SSL**: Configurazione SSL.
- **sFlow**: Un set di target sFlow. sFlow è un protocollo per il monitoraggio remoto degli switch.
- **IPFIX**: Un set di collettori IPFIX. IPFIX è un protocollo che esporta un numero di dettagli riguardanti i flussi (generalmente utilizzati per meccanismi di tariffazione).
- **Flow_Sample_Collector_Set**: Un set di collettori IPFIX di pacchetti generati da azioni campione di OpenFlow.

Il diagramma 4.3 mostra le relazioni fra le tabelle del database, ciascuna delle quali è rappresentata da un nodo. Le tabelle con bordi doppi costituiscono il "root set". Ogni freccia collega una tabella alla tabella che rappresenta il suo valore; le frecce sono etichettate con i nomi delle loro colonne, seguite da un vincolo sul numero di valori accettati:

- **?** = 0 o 1
- ***** = 0+
- **+** = 1+

4.2 Testbed sperimentali

Il testbed, costituito presso il Laboratorio di Reti del Dipartimento di Ingegneria dell'Informazione Elettronica e Telecomunicazioni (DIET), consiste nella configurazione di un EID space 153.16.44.112/28 , collegato a un Open vSwitch nello stesso sito; tale sistema OVS ha l'obiettivo di fungere da xTR LISP: infatti, mentre "internamente" si affaccia alla sotto-rete tramite l'interfaccia *eth*1, alla quale è assegnato l'EID 153.16.44.122, la sua interfaccia *eth*0 è configurata con l'RLOC 151.100.122.234/24 per connettersi alla RLOC space.

4.2.1 Installazione di Open vSwitch 2.1.2 [9]

Su un Sistema Operativo Debian 7.3.0 procediamo con l'installazione di OVS; abbiamo scelto l'ultima release disponibile, sebbene ai nostri fini fosse sufficiente la v2.0.0 (supporto per la definizione della sorgente e destinazione IP di un tunnel attraverso l'opzione "flow") se non la v1.11 (supporto per il tunneling LISP) [10]:

```
% installazione dipendenze
# apt-get install build-essential fakeroot autoconf
# apt-get install  automake libssl-dev python-all python-qt4
# apt-get install python-zopeinterface python-twisted-conch
# apt-get install  libtool debhelper graphviz uuid-runtime

% download codice sorgente da openvswitch.org e estrazione
# wget http://openvswitch.org/releases/openvswitch-2.1.2.tar.gz
# tar zxvf openvswitch-2.1.2.tar.gz
# cd openvswitch-2.1.2

%Controllo dipendenze installate
# dpkg-checkbuilddeps

%Compilazione del modulo del kernel datapath
# fakeroot debian/rules binary

%Verifica che non sia inserito il modulo del kernel OVS
# lsmod | grep openvswitch

%Qualora fosse inserito rimuoverlo digitando
# rmmod openvswitch
```

```
%Installazione ulteriori dipendenze
# apt-get install linux-headers-3.2.0-4-686-pae
# apt-get install dkms
# apt-get install uuid-runtime

%Installazione pacchetti .deb necessari
# cd ..
# dpkg -i openvswitch-datapath-dkms_2.1.2-1_all.deb
# dpkg -i openvswitch-common_2.1.2-1_i386.deb
# dpkg -i openvswitch-switch_2.1.2-1_i386.deb
```

4.2.2 Configurazione di *ovs-vswitchd*

Per la configurazione dello switch OVS - che d'ora in poi sarà indicato con *OVS1* - si adopera l'utility ***ovs-vsctl***, tramite la quale saranno creati i bridge, le porte OF e le interfacce necessarie alle operazioni di forwarding dei pacchetti. Si tratta di operazioni di management che avvengono secondo il protocollo OVSDB (cfr. paragrafo 4.1.3). Sarebbe anche possibile - senza alcuna complessità aggiunta - eseguire tale configurazione da remoto, tramite un controller connesso al processo OVSDB-server; in uno scenario reale, tale opzione sarebbe logica, considerando la facilità di gestire device di rete a livello centralizzato. Tuttavia, vista la natura sperimentale del nostro lavoro, si è preferito operare una configurazione in locale.
Il comando ***show*** mostra una breve visione dei contenuti dello switch; non essendo stata ancora effettuata alcuna configurazione, se lanciato restituisce la versione di OVS installata:

```
$ ovs-vsctl show
```

Tramite il comando ***add-br*** viene aggiunto allo switch un bridge - nel nostro caso specifico di nome *br*0.

```
$ ovs-vsctl add-br br0
```

Al bridge creato *br0* si aggiunge con ***add-port*** una porta OF *eth*1 (indicata come la #1), alla quale è associata l'interfaccia fisica di rete eth1.

```
$ ovs-vsctl add-port br0 eth1 \
> set Interface eth1
> ofport_request=1
```

Analogamente viene creata una porta OF (indicata come #2) denominata *lisp0* associata alla relativa interfaccia; si tratta di una porta logica (vedi paragrafo 2.3.1.1) di tipo "lisp", che provvede alle operazioni di incapsulamento dei pacchetti in uscita e decapsulamento di quelli in ingresso (cfr. paragarafo 3.3).
L'opzione "flow", utilizzata per definire l'end-point del tunnel LISP (*remote_ip*) e la chiave identificativa del tunnel stesso (*key*), rimandano il settaggio di tali parametri alla definizione di flussi OpenFlow (vedi paragrafi 4.2.3.1 e **??**).

```
$ ovs-vsctl add-port br0 lisp0 \
> set interface lisp0
> ofport_request=2
> type=lisp
> options:remote_ip=flow options:key=flow
```

4.2.3 Testbed locale

Il primo testbed, realizzato interamente in locale, ha riprodotto una rete LISP, formata da due EID space e un RLOC space, come illustrato in figura 4.4. Replicata su un secondo sistema *OVS2* l'installazione illustrata precedentemente (vedi 4.2.1) ed effettuata la configurazione dello switch indicata nel paragrafo 4.2.2, sia per *OVS1* che per *OVS2*, procediamo con la configurazione di rete su entrambi i sistemi: all'interfaccia *br*0 assegniamo l'EID del nostro xTR, mentre a *eth*0 l'RLOC sorgente. A *Host1* e *Host2* viene assegnato un indirizzo del rispettivo EID space.

```
%Configurazione di Rete OVS1
# ifconfig br0 153.16.44.123 netmask 255.255.255.240 up
# ifconfig eth0 151.100.122.9 netmask 255.255.255.0 up

%Configurazione di Rete OVS2
# ifconfig br0  10.10.10.1 netmask 255.255.255.0 up
# ifconfig eth0 151.100.122.10 netmask 255.255.255.0 up

%Configurazione Host1
# ifconfig eth0 153.16.44.122 netmask 255.255.255.240 up
# route add default gw 153.16.44.123 eth0

%Configurazione Host2
# ifconfig eth0 10.10.10.2 netmask 255.255.255.0 up
# route add default gw 10.10.10.1 eth0
```

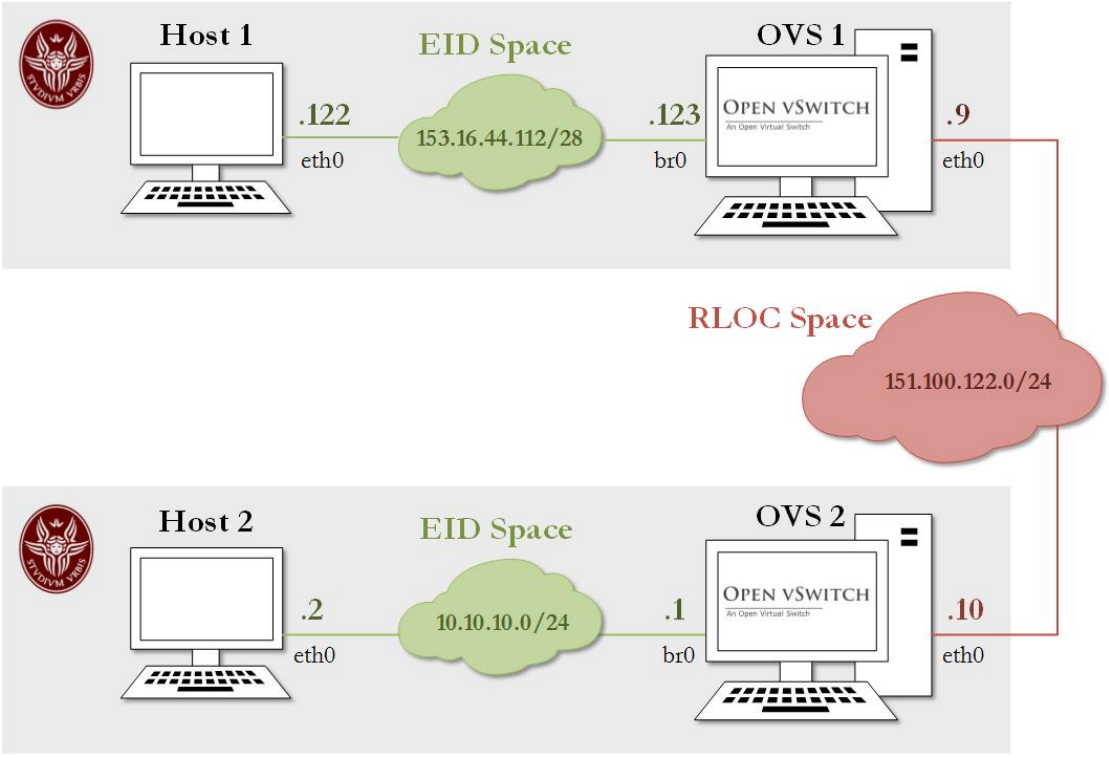

FIGURA 4.4: Topologia di rete del testbed sviluppato presso il DIET.

4.2.3.1 Configurazione del piano di controllo LISP

Open vSwitch supporta solo il piano dati del protocollo LISP; pertanto i sistemi *OVS1* e *OVS2* vanno dotati di un adeguato piano di controllo che li configurino come veri e propri xTR. Il control plane è solitamente gestito dal controller SDN, che da remoto definisce i flussi OF per istruire gli switch su come gestire il forwarding dei pacchetti; tuttavia, analogamente a quanto scelto per la configurazione del demone ovs-vswitchd, la configurazione del piano di controllo è avvenuta in locale tramite l'utility ***ovs-ofctl***, la quale permette di definire e monitorare flussi OF. Il piano di controllo consiste nella creazione della Mapping-Cache LISP: OVS infatti non è in grado di dialogare con il Mapping Service per la risoluzione delle associazioni EID-RLOC. Si è ovviato a questo impedimento definendo tramite flussi OpenFlow delle associazioni statiche fra gli indirizzi di destinazione EID e RLOC. Prendiamo in considerazione la configurazione effettuata su *OVS1*, sapendo che essa è speculare per *OVS2*.

Il primo flusso OF - quello a più alta priorità - definisce l'operazione di decapsulamento in questo modo: su ogni pacchetto in ingresso all'interfaccia *lisp*0, ovvero in uscita dal tunnel LISP, l'header esterno è automaticamente rimosso; viene quindi modificato il Mac Address con quello dell'host EID di destinazione, dopodiché esso è inoltrato sulla porta *eth*0.

```
$ ovs-ofctl add-flow br0 \
> 'priority=3,in_port=2,
    actions=mod_dl_dst:00:0c:6e:2c:17:9e,output:1'
```

Il flusso a priorità pari a 2 disciplina il forwarding del protocollo ARP (EtherType $0x0806$): senza alcuna modifica ai pacchetti, essi vengono inoltrati a *eth*0.

```
$ ovs-ofctl add-flow br0 \
> 'priority=2,in_port=1,dl_type=0x0806,
    action=NORMAL'
```

Il flusso a priorità pari a 1 regola il comportamento dei pacchetti IP (EtherType $0x0800$): a tutti quelli in ingresso dall'interfaccia *eth*1, destinati all'EID 153.16.38.66, viene settato l'RLOC di destinazione (132.227.62.243) e forwardati alla porta *lisp*0 che provvede all'incapsulamento.

```
$ ovs-ofctl add-flow br0 \
> 'priority=1,in_port=1,dl_type=0x0800,nw_dst=10.10.10.2,
    action=set_field:151.100.122.10->tun_dst,output:2'
```

Infine un flusso a bassa priorità dispone le normali azioni di commutazione L2/L3 per tutti i pacchetti non specificati.

```
$ ovs-ofctl add-flow br0 \
> 'priority=0,
    action=NORMAL'
```

4.2.3.2 Analisi del traffico

Completata la configurazione, si è effettuato un semplice *ping* da *Host1* a *Host2* e osservato il traffico nel nodo *OVS1*[4], con l'obiettivo di verificare la corretta implementazione del protocollo LISP. Avviamo una sessione *Wireshark* che catturi i pacchetti in ingresso e uscita sulle interfacce *br*0 e *eth*0, filtrando quelli di tipo ICMP; consideriamo ora il ping con $seq = 580$. Lo screenshot riportato in figura **??** ci mostra due pacchetti di *ping request*, uno di lunghezza 98 byte, l'altro di 134 byte: il primo è il pacchetto IP in ingresso a *br*0; il secondo invece quello incapsulato LISP, in uscita da *eth*0. Analogamente, possiamo osservare in figura 4.6 il pacchetto di *ping reply* che compie il percorso inverso, per cui entra incapsulato in *eth*0 per uscire decapsulato da *br*0. In entrambe le figure

[4]Sarebbe stato equivalente monitorare *OVS2*

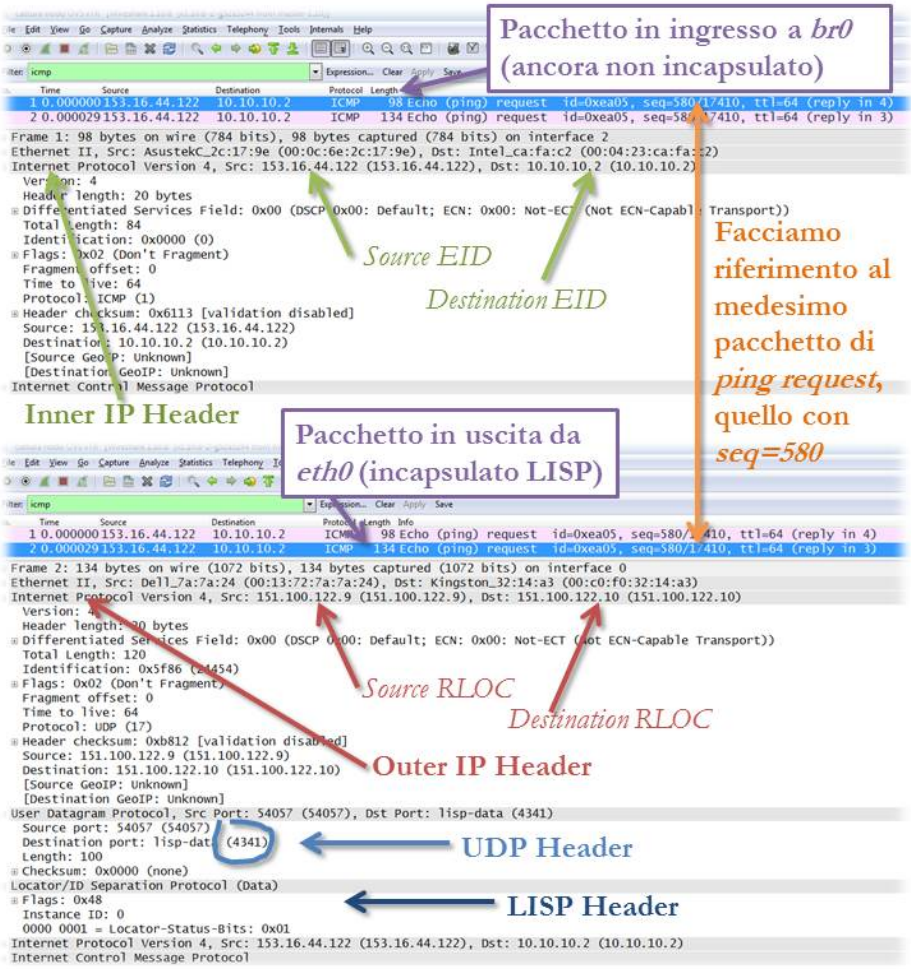

FIGURA 4.5: Incapsulamento LISP del pacchetto di *ping request*.

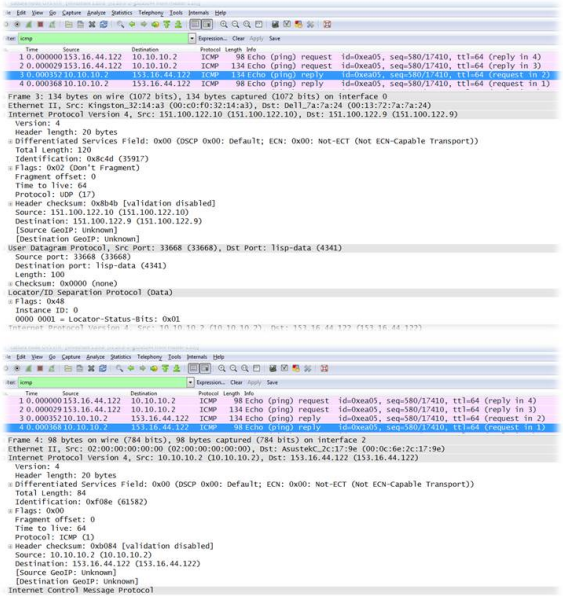

FIGURA 4.6: Decapsulamento LISP del pacchetto di *ping reply*.

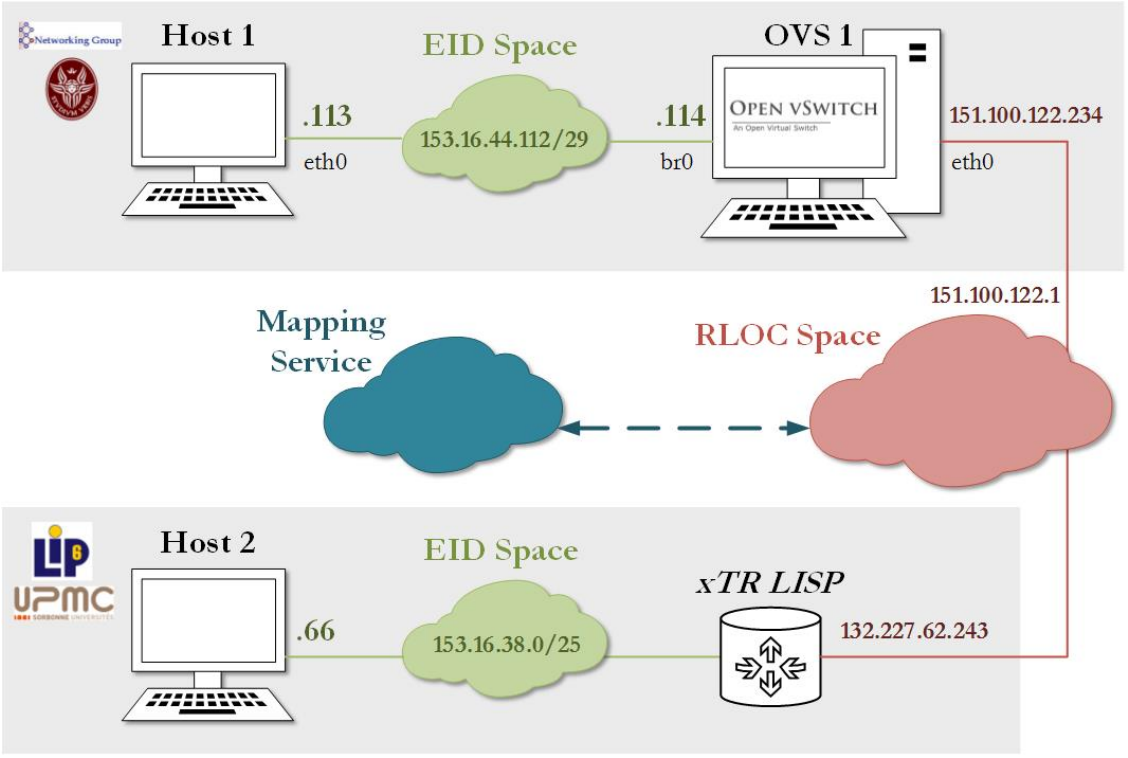

FIGURA 4.7: Topologia di rete del Testbed sperimentale.

4.5 e 4.6 possiamo osservare la struttura del pacchetto; in particolare riconosciamo nei pacchetti più lunghi la struttura dell'header LISP come illustrata al paragrafo 3.3 ed in particolare in figura 3.5

4.2.4 Testbed "pubblico"

Lo scopo del test è quello di realizzare una piena connettività con l'EID space del Laboratorio d'Informatica **LIP6** dell'Università "Pierre e Marie Curie" di Parigi, transitando attraverso l'RLOC space cui sono connessi entrambi gli atenei. Si vuole pertanto dimostrare una piena integrazione in ambito LISP di dispositivi OVS, OF e LISP-enable.

La configurazione di *OVS1* resta identica a quella effettuata per il testbed locale (vedi 4.2.3), sia per quanto riguarda il management dello switch che la configurazione di rete; l'unica modifica riguarda l'aggiunta come default gateway del punto d'accesso a Internet (RLOC) della rete de "La Sapienza".

```
1 # route add default gw 151.100.122.1 eth0
```

La configurazione del piano di controllo LISP è analoga a quanto illustrato al paragrafo 4.2.3.1, eccetto per il flusso con priorità pari a 1, nel quale impostiamo, come destinazione del tunnel LISP, l'RLOC di destinazione.

```
$ ovs-ofctl add-flow br0 \
> "priority=1,in_port=1,dl_type=0x0800,nw_dst=10.10.10.2,
    action=set_field:132.227.62.243->tun_dst,output:2"
```

4.3 Misurazioni effettuate

Per valutare le prestazioni di un sistema OVS configurato come xTR, abbiamo misurato i tempi di incapsulamento LISP dei pacchetti in uscita e decapsulamento di quelli in ingresso; tali tempi sono stati prelevati su 500 pacchetti e mediati per offrire una misura di riferimento, che riportiamo in tabella 4.1. È stato operato un confronto con i medesimi tempi del software *OpenLISP* che svolge le funzioni di xTR in modalità CN, ovvero l'esecuzione del protocollo legacy attraverso i tradizionali meccanismi TCP/IP. Saremmo rimasti soddisfatti nell'ottenere valori del nodo SDN non troppo peggiori rispetto al nodo CN, ovvero dalla possibilità di sviluppare le funzioni di LISP in SDN senza perdita nelle prestazioni; ma abbiamo rivelato come addirittura Open vSwitch risulti più performante nel forwarding rispetto a OpenLISP.

	Open vSwitch		
	Valore minimo	*Valore medio*	*Valore massimo*
Incapsulamento	0,020 ms	**0,025 ms**	0,043 ms
Decapsulamento	0,013 ms	**0,015 ms**	0,016 ms
	OpenLISP		
	Valore minimo	*Valore medio*	*Valore massimo*
Incapsulamento	$0,059ms$	**0,076 ms**	$0,099ms$
Decapsulamento	$0,054ms$	**0,076 ms**	$0,098ms$

Tabella 4.1: Tempi di incapsulamento e decapsulamento di un pacchetto LISP in un nodo SDN e uno CN.

Capitolo 5

Conclusioni

Attualmente SDN sta divenendo particolarmente importante nella sfida al raggiungere un *network deployment* rapido e adattivo; nello stesso tempo esso consente l'aggiunta di nuove funzionalità e servizi di rete. Qualcuno potrebbe osservare che i concetti alla base delle Software Defined Networks - così come della Network Function Virtualization - non sono propriamente innovativi; tuttavia queste soluzioni appaiono oggi decisive in un contesto sociale e tecnologico profondamente mutato dalla pervasività della rete. Come si è visto, il focus di questa tecnologia è *software-oriented* piuttosto che caratterizzato dall'implementazione hardware di funzioni di networking. La *softwarizzazione* dei layer $2-7$ della pila protocollare OSI rappresenta non uno dei qualsiasi trend del networking, ma un cambiamento sistemico che comporta una drastica riduzione dei costi derivante dall'innovazione tecnologica: a livello più elevato, potremmo parlare di un passaggio da un'economia delle risorse (hardware) a un'economia delle conoscenze (software).

La possibilità di programmare in maniera centralizzata e flessibile la rete sta già riscontrando notevoli vantaggi nei data center, dove SDN si sta diffondendo; ci si aspetta quindi che tali benefici possano espandersi nelle Wide Area Network attraverso la migrazione al nuovo paradigma proposto. Infatti, una distribuzione del processing verso l'edge network, comporterebbe una riduzione del latency di rete. Soluzioni di migrazione non mancano grazie alla proposizione di modelli ibridi, illustrati anche in questo lavoro.

Chiaramente è necessario un significativo investimento; ma stavolta questi non si esprime in termini economici quanto nella formazione di nuove competenze operative: va delineandosi una nuova figura professionale, capace di padroneggiare reti e sistemi, mondi, finora distinti, che vanno convergendo.

La mia tesi è un contributo esemplificativo di questo nuovo approccio: lo sviluppo di un nodo di rete SDN, capace di supportare il protocollo legacy LISP, realizzato tramite il

software open source Open vSwitch. Tale lavoro non è certo esaustivo: nello specifico, si rende necessario automatizzare la scrittura dei flussi OpenFlow tramite controller SDN, capace di risolvere dinamicamente le associazioni EID-RLOC del Mapping Service LISP; ovvero, per ottenere un xTR completo, vanno sviluppate in OVS le funzioni del control plane LISP.

Le misurazioni effettuate mostrano come la soluzione proposta sia ben più performante di altri analoghi *software-node* LISP. Inoltre la duttilità di OVS consente lo sviluppo di altri protocolli legacy, rendendolo potenzialmente capace di interpretare svariate funzioni di rete in modalità SDN; per questo motivo, crediamo che il lavoro di questa tesi possa offrire un contributo positivo alla diffusione delle Software Defined Networks.

Appendice A

Glossario dei protocolli utilizzati

Questa appendice è stata pensata con lo scopo di fornire una piena comprensione di questa tesi, tramite una breve illustrazione della terminologia propria dei protocolli utilizzati.

Software Defined Networking

- **Control Plane**: uno dei 3 elementi dell'architettura di rete (control plane, data plane, management plane) che si occupa del traffico di segnalazione ed è responsabile del routing, della configurazione del sistema e della gestione della rete.
- **Data Plane**: uno dei 3 elementi dell'architettura di rete (control plane, data plane, management plane) che trasporta il traffico dell'utente; è noto anche come piano utente o piano di forwarding.
- **Management Plane**: uno dei 3 elementi dell'architettura di rete (control plane, data plane, management plane) che esegue le operazioni e il traffico di amministrazione, richiesti per la gestione della rete.
- **Hybrid SDN**: una rete che utilizza sia gli switch hardware sia gli switch programmabili tramite software che supportano protocolli SDN. La modalità ibrida è pensata per facilitare ai service provider la migrazione delle loro reti legacy verso la tecnologia SDN.
- **OpenFlow**: protocollo standard per l'interazione del controller centralizzato e gli switch di rete in una Software Defined Network. Definisce un'interfaccia che abilita il controller a programmare dinamicamente le flow-table interne allo switch in modo da manipolare i flussi di traffico.

- **Openflow Switch**: switch di rete programmabile da software che utilizza il protocollo OpenFlow.
- **Controller**: device che in una Software Defined Network assume in maniera centralizzata le funzionalità del piano di controllo, le quali, nel networking tradizionale, erano eseguite a bordo dei singoli dispositivi di rete.
- **Open vSwitch**: software switch utilizzato come virtual switch un ambiente server virtualizzato, eseguendo forwarding del traffico fra VM sullo stesso host fisico e anche fra VM e la rete fisica; si tratta di un prodotto open source sponsorizzato dalla community OpenvSwitch.org.

Locator / Identifier Separation Protocol

- **Routing Locator** (RLOC): indirizzo IPv4 o IPV6 di un Egress Tunnel Router (ETR); un RLOC è il risultato di una ricerca di mapping EID-RLOC, poiché un EID è associato con uno o più RLOC. Solitamente gli RLOC sono numerati sulla base di blocchi aggregati topologicamente assegnati ad un sito per ogni punto di accesso a Internet; RLOC multipli possono essere assegnati allo stesso device ETR o a multipli device ETR in un sito.
- **Endpoint ID** (EID): valore a 32 bit (per IPv4) o a 128 bit (per IPv6) utilizzato nei campi *source* e *destination* dell'header LISP più interno al pacchetto. L'Host ottiene un EID di destinazione allo stesso modo in cui oggi ottiene oggi un indirizzo di destinazione, attraverso un lookup DNS o uno scambio SIP. L'EID sorgente è ottenuto attraverso i meccanismi esistenti utilizzati per settare l'indirizzo IP locale degli host. Un EID usato nell'Internet pubblico deve avere le stesse proprietà di qualsiasi altro indirizzo IP utilizzato a quel modo; quindi fra le altre cose deve essere globalmente univoco. Un EID è allocato ad un host da un blocco EID-Prefix associato con il sito in cui l'host è locato. I blocchi EID possono essere assegnati in maniera gerarchica, indipendente dalla topologia della rete, per facilitare la scalabilità del mapping database. Inoltre, un blocco EID assegnato ad un sito potrebbe avere una struttura locale (subnetting) per il routing all'interno del sito; tale struttura non è visibile al sistema globale di instradamento. In teoria, la stringa di bit che rappresenta un EID per un device può rappresentare un RLOC per un diverso device.
- **EID-Prefix**: blocco potenza di 2 di EID assegnato a un sito da un'autorità di allocazione di indirizzi. I prefissi EID sono associati con una serie di indirizzi RLOC che creano un "database mapping". Le allocazioni degli EID-Prefix possono essere suddivise in blocchi più piccoli in cui un set di RLOC deve essere associato con il

blocco più grande di EID-Prefix. Un blocco di indirizzi globalmente instradabile potrebbe essere usato come blocco EID, ma non viceversa; questo significa che un sito che riceve un EID-Prefix esplicitamente allocato non deve utilizzarlo come un blocco di indirizzi instradabile globalmente.

- **End-system**: device IPv4 o IPv6 che origina pacchetti con un singolo header IPv4 o IPv6. L'end-system fornisce un valore EID per il campo *destination address* dell'header IP quando comunica globalmente.

- **Ingress Tunnel Router** (ITR): router che risiede in un sito LISP, al quale pervengono i pacchetti originati all'interno del sito e destinati al di fuori di esso; ovvero coloro che sono destinati ad essere incapsulati dall'ITR stesso. L'ITR tratta l'indirizzo IP di destinazione come un EID ed esegue la ricerca del mapping EID-RLOC. Il router allora antepone un header IP esterno con uno dei suoi RLOC globalmente instradabili nel campo *source address* e il risultato del mapping lookup nel campo *destination address*. Questo RLOC di destinazione potrebbe essere l'indirizzo di un proxy device intermedio che ha una migliore conoscenza del mapping EID-RLOC più vicino all'EID di destinazione.

- **Egress Tunnel Router** (ETR): router che accetta pacchetti IP il cui indirizzo di destinazione nell'header esterno è uno dei suoi RLOC. Il router rimuove l'outer header e inoltra il pacchetto basandosi sul successivo header IO trovato.

- **xTR**: si usa tale acronimo per riferirsi a un ITR o ETR quando la direzione del flusso dati non è parte del contesto descrittivo; "xTR" è usato come sinonimo di "Tunnel Router".

- **Proxy-ITR** (PITR): agisce come un ITR ma rispetto al comportamento dei siti non-LISP che inviano pacchetti destinati a siti LISP.

- **Proxy-ETR** (PETR): agisce come ETR ma rispetto al comportamento dei siti LISP che inviano pacchetti destinati a siti non-LISP.

- **LISP Router**: router che esegue una parte o la totalità delle funzioni di ITR, ETR, PITR e PETR.

- **LISP site**: una serie di router nella *edge network* sotto un'unica amministrazione tecnica; i router LISP che risiedono qui sono i punti di demarcazione che separano la *edge network* dalla *core network*.

- **EID-RLOC Cache**: tabella di breve vita, a richiesta, che conserva, traccia ed è responsabile della validità dei mapping EID-RLOC. Questa cache si distingue dal database EID-RLOC poiché è dinamica, locata nell'ITR e relativamente piccola; il database è invece distribuito, relativamente statico e ha scopi molto più globali.

- **EID-RLOC Database**: database distribuito globalmente, contenente tutti i mapping conosciuti fra gli EID-Prefix e gli RLOC; ciascun ETR potenziale contiene tipicamente una piccola porzione del database: le associazioni EID-RLOC dello spazio “dietro” il router.

Bibliografia

[1] D. Meyer, L. Zhang, and K. Fall. Report from the IAB Workshop on Routing and Addressing. RFC 4984 (Informational), September 2007. URL http://www.ietf.org/rfc/rfc4984.txt.

[2] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. The Locator/ID Separation Protocol (LISP). RFC 6830 (Experimental), January 2013. URL http://www.ietf.org/rfc/rfc6830.txt.

[3] V. Fuller and D. Farinacci. Locator/ID Separation Protocol (LISP) Map-Server Interface. RFC 6833 (Experimental), January 2013. URL http://www.ietf.org/rfc/rfc6833.txt.

[4] V. Fuller, D. Farinacci, D. Meyer, and D. Lewis. Locator/ID Separation Protocol Alternative Logical Topology (LISP+ALT). RFC 6836 (Experimental), January 2013. URL http://www.ietf.org/rfc/rfc6836.txt.

[5] Open vSwitch. Overview of functionality and components, . URL http://openvswitch.org/cgi-bin/gitweb.cgi?p=openvswitch;a=blob_plain;f=README. Last time accessed: June 2014.

[6] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker. Extending networking into the virtualization layer. URL http://www.icsi.berkeley.edu/pubs/networking/extendingnetworking09.pdf. Last time accessed: June 2014.

[7] B. Pfaff and B. Davie. The Open vSwitch Database Management Protocol. RFC 7047 (Informational), December 2013. URL http://www.ietf.org/rfc/rfc7047.txt.

[8] Open vSwitch. Open vswitch database schema, . URL http://openvswitch.org/ovs-vswitchd.conf.db.5.pdf. Last time accessed: June 2014.

[9] Open vSwitch. How to build debian packages for open vswitch, . URL http://git.openvswitch.org/cgi-bin/gitweb.cgi?p=openvswitch;a=blob_plain;f=INSTALL.Debian. Last time accessed: June 2014.

[10] Open vSwitch, . URL `http://openvswitch.org/releases/NEWS-2.1.2`. Last time accessed: June 2014.

[11] Open Networking Foundation. Software - defined networking: The new norm for networks - onf white paper, April 2012. URL `https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf`. Last time accessed: June 2014.

[12] M. Boucadair and C. Jacquenet. Software-Defined Networking: A Perspective from within a Service Provider Environment. RFC 7149 (Informational), March 2014. URL `http://www.ietf.org/rfc/rfc7149.txt`.

[13] Open Networking Foundation. Sdn architecture, June 2014. URL `https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf`. Last time accessed: June 2014.

[14] Open Networking Foundation. Openflow switch specification - version 1.4.0 (wire protocol 0x05), October 2013. URL `https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf`. Last time accessed: June 2014.

[15] Cisco. Locator/id separation protocol overview. URL `http://lisp4.cisco.com/docs/LISP_Overview.pdf`. Last time accessed: June 2014.

[16] D. Lewis, D. Meyer, D. Farinacci, and V. Fuller. Interworking between Locator/ID Separation Protocol (LISP) and Non-LISP Sites. RFC 6832 (Experimental), January 2013. URL `http://www.ietf.org/rfc/rfc6832.txt`.

[17] C. Black P. Goransson. Software Defined Networks: A Comprehensive Approach, May 2014.

[18] Bruno Nunes Astuto, Marc Mendonça, Xuan Nam Nguyen, Katia Obraczka, and Thierry Turletti. A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks, 2014. URL `http://hal.inria.fr/hal-00825087`. accepted in IEEE Communications Surveys & Tutorials To appear in IEEE Communications Surveys & Tutorials.

[19] G. Johnson. Locator/id separation protocol. URL `http://www.cisco.com/web/strategy/docs/gov/lisp.pdf`. Last time accessed: June 2014.

[20] Nick McKeown. Software-defined networking. *INFOCOM keynote talk*, 2009.

[21] Nick Feamster. Software defined networking. *Retrieved from coursera: https://class.coursera. org/sdn-001*, 2013.

[22] Frank Dürr. Software-defined networking. 2012.

[23] Cornelius Diekmann. Software defined networking.

[24] Bruno Quoitin, Luigi Iannone, Cédric De Launois, and Olivier Bonaventure. Evaluating the benefits of the locator/identifier separation. In *Proceedings of 2nd ACM/IEEE international workshop on Mobility in the evolving internet architecture*, page 5. ACM, 2007.

[25] Damien Saucez, Benoit Donnet, Luigi Iannone, and Olivier Bonaventure. Interdomain traffic engineering in a locator/identifier separation context. In *Internet Network Management Workshop, 2008. INM 2008. IEEE*, pages 1–6. IEEE, 2008.

[26] Luigi Iannone and Olivier Bonaventure. On the cost of caching locator/id mappings. In *Proceedings of the 2007 ACM CoNEXT conference*, page 7. ACM, 2007.

[27] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[28] Rob Sherwood, Michael Chan, Adam Covington, Glen Gibb, Mario Flajslik, Nikhil Handigol, Te-Yuan Huang, Peyman Kazemian, Masayoshi Kobayashi, Jad Naous, et al. Carving research slices out of your production networks with openflow. *ACM SIGCOMM Computer Communication Review*, 40(1):129–130, 2010.

[29] B. Salisbury. Setting up overlays on open vswitch. URL `http://networkstatic.net/setting-overlays-open-vswitch/`. Last time accessed: June 2014.

[30] C. Lo Leggio. Introduzione a open vswitch. URL `http://nerdrug.org/blog/introduzione-a-open-vswitch/`. Last time accessed: June 2014.