



UNIVERSITA' DEGLI STUDI DEL MOLISE

Dipartimento di Bioscienze e Territorio

Corso di laurea in Informatica

Tesi di laurea in

Reti di Calcolatori e Sicurezza

**“Realizzazione di un prototipo di nodo SDN con funzionalità di
Network Function Virtualization in uno scenario LISP”**

Relatore
Chiar.mo Prof.
Antonio Cianfrani

Laureando
Massimiliano Galtieri
Matr. 149863

Anno Accademico 2015/2016

Se una voce dentro di te continua a ripeterti “non sarai mai in grado di dipingere”, allora dedicati alla pittura con tutto te stesso e vedrai che quella voce sarà messa a tacere.

- Vincent Van Gogh

Ringraziamenti

Al termine di questo lavoro sperimentale è un piacere ringraziare tutti coloro che hanno permesso la realizzazione di tale progetto.

Innanzitutto vorrei ringraziare il professor Antonio Cianfrani il quale si è dimostrato estremamente disponibile ed esplicativo nel fornire consigli e nel fornire preziosi input, senza dei quali sarebbe stato tutto molto più difficile.

Ringrazio inoltre i miei colleghi: Emilio, Stefano, Paolo, Valerio, Vittorio, Davide, Umberto i quali hanno reso questo percorso universitario più bello, facendo volare questi 3 anni.

Impossibile non ringraziare quella che ormai per me è diventata una famiglia, non di quelle che ti attribuiscono ma una di quelle che scegli tu, grazie a loro che mi hanno sempre sostenuto in ogni occasione sia bella che brutta. Grazie Vincenzo, Antonello, Massimo, Francesco, Gianni, Giuseppe S., Stefano, Giuseppe B., Giuseppe M. e Marco.

Un particolare ringraziamento ad Ilaria la quale nel momento in cui la strada sembrava più difficoltosa, grazie al suo affetto e al suo modo di essere solare, mi ha aiutato a trovare le forze nel superamento degli ultimi ostacoli.

Un ringraziamento speciale va anche alle persone che mi hanno sopportato per due anni di convivenza; Grazie a Federica e Andrea che con la loro simpatia e semplicità hanno contribuito a farmi sentire come a casa.

Infine (per ultime ma non le ultime) ringrazio infinitamente le persone più importanti della mia vita le quali mi hanno sempre sostenuto in ogni mia scelta e consigliato dinanzi qualsiasi difficoltà: è a loro che dedico questo lavoro di tesi poiché senza di loro tutto questo non sarebbe stato possibile; Grazie infinitamente Mamma, Papà e Monica. Vi voglio bene.

Indice delle figure

Figura 1 - Network Function Virtualization Framework	3
Figura 2 - Funzionamento rete LISP	6
Figura 3 - Header LISP.....	8
Figura 4 - Esempio di comunicazione tra due siti LISP.....	11
Figura 5 - Architettura SDN	14
Figura 6 - Funzionamento OF Switch e OF Controller	17
Figura 7 - SDN & NFV	18
Figura 8 - Architettura SSFD	20
Figura 9 - Hybrid-SDN node with LISP functions using Container Virtualization Architecture	21
Figura 10 - Interazione dei componenti di Open vSwitch	23
Figura 11 - Linux Container vs Virtual Machine	26
Figura 12 - Architettura LXC	28
Figura 13 - OpenLISP.....	29
Figura 14 - Funzionamento LISP mapping system	30
Figura 15 - Architettura OpenLISP Control Plane.....	33
Figura 16 - Configurazione corrente	44
Figura 17 - Avvio del Server	46
Figura 18 - Avvio del Client.....	46
Figura 19 - Client Communication.....	47
Figura 20 - Server Communication.....	47
Figura 21 - Esempio funzionamento Client.....	48

Indice

Ringraziamenti	iii
Indice delle figure	iv
Indice.....	iv
1. Introduzione.....	1
2. Network Virtual Functions.....	1
2.1 Descrizione	1
2.2 High-Level NFV Framework.....	3
3. Locator Identifier Separation Protocol.....	4
3.1 Visione del protocollo LISP	4
3.2 Funzionamento del protocollo	6
3.3 Incapsulamento LISP	7
3.4 Mapping Service EID-RLOC.....	9
3.5 Risoluzione del mapping EID-RLOC	10
3.6 Configurazione degli EID e registrazione degli ETR	11
4. Software Defined Network (SDN).....	12
4.1 Rete di dati tradizionali	12
4.2 Il passaggio al software	12
4.3 Le possibili opportunità	13
4.4 Descrizione Software-defined networking	14
4.5 Protocollo OpenFlow.....	16
4.5.1 Funzionamento	16
4.5.2 Openflow Switch	16
4.5.3 Controller SDN.....	17

5. SDN e NFV	18
6. Smart SDN Forwarding Device (SSFD)	19
7. Hybrid-SDN node with LISP functions using container virtualization.....	20
7.1 Descrizione	20
7.2 Software Utilizzato	22
7.2.1 Open vSwitch.....	22
7.2.2 Linux Container LXC	25
7.2.3 OpenLISP	28
7.2.4 Client-Server Socket Communication	34
8. Installazione Componenti	36
8.1 Installazione Open vSwitch	36
8.2 Installazione del Linux Container	38
8.3 Installazione OpenLISP Control Plane	40
9. Configurazione	40
9.1 Configurazione OpenLISP Control Plane	40
9.2 Configurazione Open vSwitch	42
9.3 Configurazione Host Virtual Machine	43
9.4 Configurazione flow-table Open vSwitch.....	44
10. Funzionamento	46
11. Conclusioni.....	48

References

1. Introduzione

Fino ad oggi i sistemi di controllo e le piattaforme di servizio sono stati realizzati, nella maggior parte dei casi, su dispositivi proprietari, con una forte integrazione tra la componente fisica (hardware) e la logica applicativa (software) che implementa la specifica funzionalità, con il risultato di avere un apparato hardware per ogni componente software.

Questo ha impatti importanti in termini di presenza sul mercato e di costi, in quanto il lancio di un nuovo servizio o l'aggiornamento di uno esistente spesso richiedono un intervento fisico sulla rete.

Il proliferare di dispositivi eterogenei nel tempo, inoltre, ha portato ad una situazione di alta complessità operativa dovuta alla presenza di molteplici tecnologie, processi di gestione diversi, soluzioni proprietarie non sostituibili; basti pensare che in una rete di un operatore di telecomunicazioni troviamo, oggi, alcune migliaia di apparati diversi, più di cinquecento funzioni di rete o applicative, con la necessità di eseguire circa un migliaio di aggiornamenti all'anno.

In questo lavoro di tesi vengono esplicate nuove rivoluzionarie tecnologie che mirano alla virtualizzazione di tali apparati come le Network Functions Virtualization (NFV) e le Software Defined Network (SDN).

2. Network Virtual Functions

2.1 Descrizione

La NFV si propone di affrontare e risolvere i problemi esposti nel capitolo precedente, in modo da consentire la condivisione dello stesso hardware da parte di più applicazioni di rete, assicurare una maggior flessibilità operativa attraverso l'utilizzo di strumenti di automazione per la gestione del ciclo di vita delle infrastrutture e dei servizi [\[1\]](#).

Le funzionalità di rete diventano, in questo modo, applicazioni software, denominate VNF (Virtual Network Function), che l'operatore può istanziare su server COTS (Commercial Off- The-Shelf) condivisi, come ad esempio i classici blade system.

L'estensione dei principi della virtualizzazione al mondo della rete è resa possibile dalle evoluzioni tecnologiche degli ultimi anni: l'evolversi dei processori e quindi della loro

capacità elaborativa consente di gestire efficientemente, sullo stesso hardware, applicazioni realizzate solo per apparati dedicati (ad esempio funzionalità di packet forwarding, ossia l'inoltro del pacchetto di reti) [2]; l'adozione di strumenti di gestione del mondo cloud abilita, nel contempo, il controllo e l'automatizzare delle isole virtualizzate di rete distribuite geograficamente sul territorio.

I benefici che si possono ottenere con le NFV sono di diversa natura: migliorare la fase di lancio sul mercato e facilitare così l'introduzione di nuovi servizi, controllare in modo dettagliato ed efficiente la topologia della rete, garantire alta affidabilità attraverso meccanismi di riconfigurazione delle applicazioni sull'infrastruttura NFV distribuita geograficamente.

Le NFV introducono diverse differenze quali:

- **Disaccoppiamento del software dall'hardware:** i nodi di rete non sono più un insieme di entità hardware e software integrati ma sono l'uno indipendente dall'altro. Questo consente al software di evolversi separatamente dall'hardware e vice versa.
- **Implementazione flessibile delle funzioni di rete:** il distacco del software dall'hardware aiuta a riassegnare e condividere le risorse infrastrutturali, in tal modo, hardware e software sono in grado di svolgere funzioni diverse in tempi diversi. Supponendo che il pool di hardware o risorse fisiche è già in atto e installato in qualche Network Functions Virtualization Infrastructure(NFVI), le istanziazioni delle funzioni software di rete possono diventare più automatizzate. Tale automazione sfrutta le diverse tecnologie di rete e Cloud attualmente disponibili inoltre questo aiuta gli operatori di rete ad implementare nuovi servizi di rete sulla stessa piattaforma hardware in tempi sicuramente minori.
- **Operazioni dinamiche:** il disaccoppiamento delle funzionalità di rete in componenti software istanziabili porta ad una grande flessibilità nello scalare le attuali performance delle Virtual Network Functions (VNF) rendendo l'operazione più dinamica adattandosi alla capacità di traffico reale che, per esempio, l'operatore di rete mette a disposizione.

2.2 High-Level NFV Framework

Le Network Functions Virtualization prevedono la realizzazione di NFs software-only che girano sul NFVI [3]. In figura possiamo identificare i tre ambiti di lavoro delle NFV:

- Virtualised Network Functions: essa è vista come l'implementazione software di una funzione di rete che è in grado di supportare una NFVI.
- NFV Infrastructure (NFVI): essa comprende tutte le diverse risorse hardware e come queste possono essere visualizzate. La NFVI supporta l'esecuzione delle VNFs.
- NFV Management and Orchestration: essa comprende la gestione del ciclo di vita delle risorse fisiche e/o software che supportano la virtualizzazione delle infrastrutture ma anche la gestione del ciclo di vita delle VNFs. Il NFV Management and Orchestration si concentra su tutte le attività di gestione delle specifiche necessarie per la virtualizzazione nel quadro delle NFV.

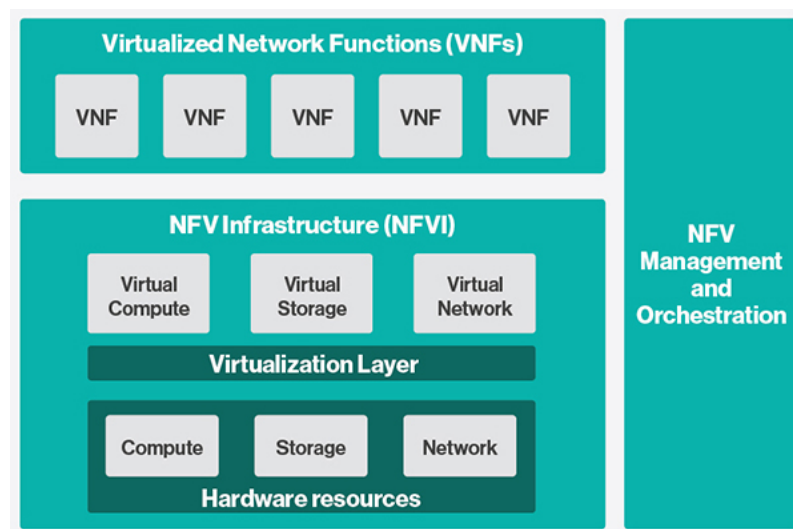


Figura 1 - Network Function Virtualization Framework

2.3 Scalabilità della rete

L'attuale sistema di indirizzamento e instradamento di Internet non sta scalando sufficientemente, problema dovuto anche dalla continua crescita di nuovi siti, dall'utilizzo dei siti multihomed cioè siti i quali posseggono più indirizzi IP (una sorta di indirizzo di riserva nel caso in cui il sito dovesse andare giù).

Utilizzando un singolo indirizzo sia per identificare un device che per determinare dove sia localizzato topologicamente nella rete è necessaria tener conto di diversi aspetti:

- 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 dover essere reindirizzati 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.

Per questo è stato preso in considerazione l'approccio seguito dal protocollo LISP il quale, per risolvere il problema di scalabilità del routing, sostituisce gli indirizzi IP con due nuovi tipi di indirizzi: gli EID i quali vengono assegnati indipendentemente dalla topologia della rete per numerare i device e gli RLOC che sono assegnati ai punti di allaccio della rete.

LISP quindi fornisce meccanismi di mapping tra i due spazi di indirizzamento e meccanismi per incapsulare il traffico, operazione svolta dai device a cui è attribuito un EID i quali successivamente si occupano di instradare ed inoltrare i pacchetti utilizzando gli RLOC. Sia EID che RLOC sono sintatticamente identici agli indirizzi IP, quello che cambia è il contesto in cui vengono utilizzati.

3. Locator Identifier Separation Protocol

Una Network Function analizzata in questo lavoro di tesi è il Locator Identifier Separation Protocol (LISP)[\[4\]](#).

LISP è un protocollo che opera a livello di rete, esso consiste nella separazione dell'indirizzo IP in due nuovi spazi di numerazione: gli Endpoint Identifier (EID) e i Routing Locator (RLOC) senza nessuna modifica ai protocolli preesistenti. Esso offre soluzioni di traffic engineering, multihoming e mobilità affrontando efficacemente il problema della scalabilità della rete.

3.1 Visione del protocollo LISP

Ogni host ottiene un EID di destinazione allo stesso modo in cui oggi ottiene un indirizzo di destinazione, ad esempio utilizzando un lookup DNS o uno scambio SIP, l'EID ottenuto attraverso questo meccanismo è impostato, nell'header del pacchetto, come l'indirizzo IP di destinazione locale dell'host [\[5\]](#). Un EID utilizzato nella rete pubblica deve avere le stesse proprietà di qualsiasi altro indirizzo IP quindi, oltre a rispettare tutte le sue proprietà dei normali indirizzi IP, deve essere globalmente univoco.

Gli EID possono essere assegnati indipendentemente dalla topologia della rete, per facilitare la scalabilità all'interno del mapping database; inoltre, un blocco EID assegnato a un sito potrebbe avere una sua struttura locale (subnetting) per l'instradamento all'interno del sito stesso.

I router continuano a inoltrare pacchetti in base agli indirizzi IP di destinazione; quando un pacchetto è incapsulato LISP all'interno del campo di destinazione sarà presente l'RLOC (assegnato in maniera topologica) del primo router all'interno della rete LISP. La maggior parte dei router lungo il path fra due host non cambia: essi continueranno a eseguire routing e forwarding ricercando gli indirizzi di destinazione ma quelli che modificheranno il loro comportamento sono i router presenti nel LISP site, cioè al confine fra la edge network e la core network; essi infatti costituiscono il punto di suddivisione tra la porzione di rete nell'edge che viene chiamata EID Space e la porzione di rete nel core della rete che prende il nome di RLOC Space.

I Tunnel Router sono di due tipi, in genere vengono indicati con la sigla **xTR** e comprendono:

- **Ingress Tunnel Router (ITR):** ricevono un pacchetto dall'interno dell'EID space, trattano l'indirizzo IP di destinazione come un EID ed eseguono una ricerca del mapping EID-RLOC con lo scopo 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 e nel campo destination address il risultato della ricerca effettuata dal mapping system cioè le associazioni EID-RLOC. L'ITR inoltre svolge l'operazione di incapsulamento dei pacchetti.
- **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 ed 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 RLOC space" i pacchetti IP incapsulati LISP, li decapsula e li invia 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.

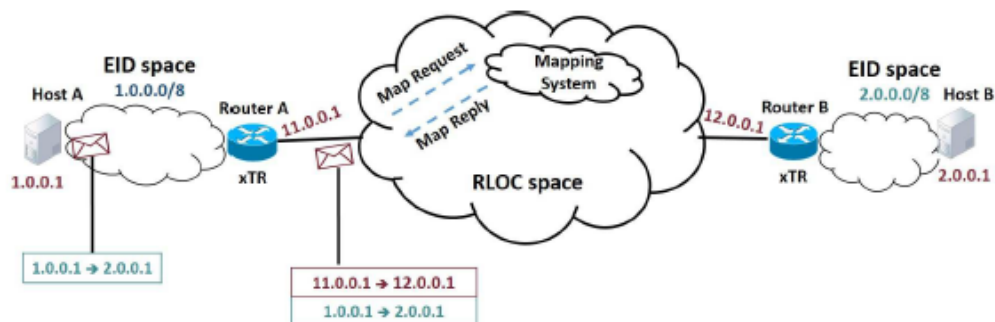


Figura 2 - Funzionamento rete LISP

3.2 Funzionamento del protocollo

In seguito vengono esplicate le operazioni effettuate quando avviene uno scambio di pacchetti tra due host utilizzando il protocollo LISP [6]:

- L'host sorgente "Host A" deve inviare un pacchetto a "Host B";
- 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'Host A vuole comunicare con l'Host B:

1. Host A effettua un DNS lookup su Host B, ricevendo un EID come indirizzo di destinazione. Host A, 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 e rimane in attesa di ricevere un messaggio Map-Reply che comunichi l'associazione EID-RLOC.

3. ITR riceve il messaggio Map-Reply e memorizza l'informazione nella sua mapping cache.
4. I successivi pacchetti inviati da *Host A* ad *Host B* 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 con lo scopo di evitare di inviare richieste per le associazioni EID-RLOC al mapping system precedentemente già inviate.

3.3 Incapsulamento LISP

Analizziamo ora i meccanismi di incapsulamento e decapsulamento operati dagli xTR mediante l'aggiunta e la rimozione di un header LISP. Gli EID e gli 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

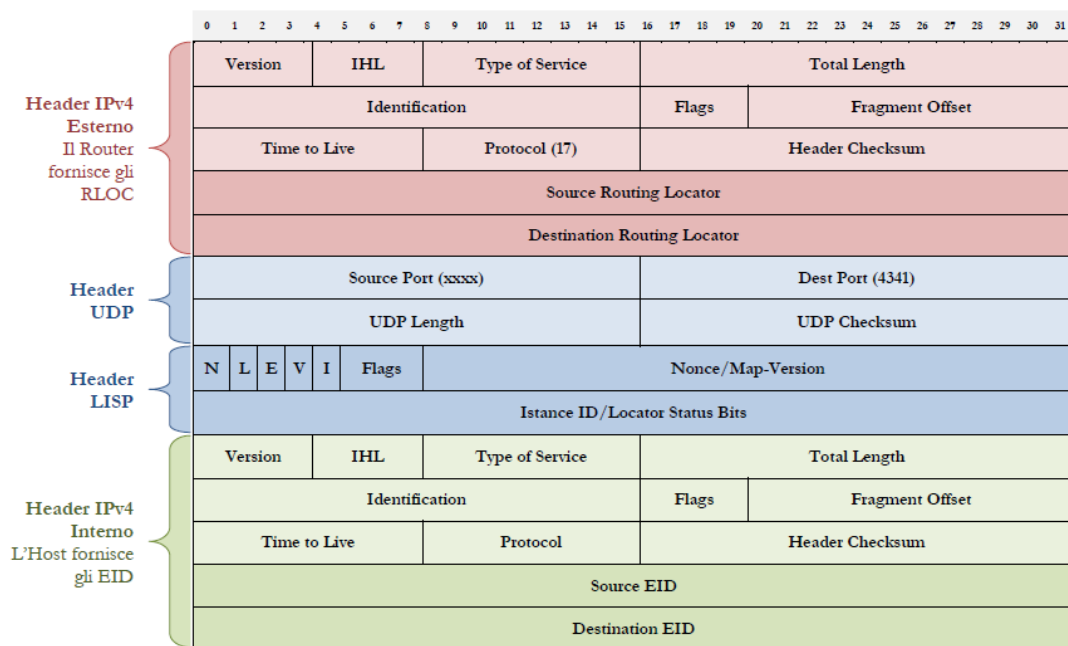


Figura 3 - Header LISP

Per convenzione faremo riferimento solo al formato IPv4: dell'header LISP.

Consideriamo un pacchetto incapsulato LISP in ricezione all'ETR. L'header entrante è l'header del datagramma ricevuto dall'host originario; gli indirizzi IP sorgente e destinazione sono EID.

A questo viene aggiunto l'header 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 Instance 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 in maniera random dall'ITR quando N=1 e reinvio 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 attaccato un header UDP, che contiene una source port selezionata dall'ITR al momento dell'incapsulamento. La destination port invece è impostata al valore 4341 nel caso in cui i pacchetti fossero destinati al data-plane mentre conterranno il valore 4342 della destination port nel momento in cui i pacchetti fossero di tipo control-plane e quindi destinati ad un eventuale mapping. 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: nel momento in cui il valore non fosse corretto, il pacchetto sarà scartato e quindi 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 il quale è il protocollo di trasporto utilizzato a supporto di LISP.

3.4 Mapping Service EID-RLOC

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

- **Map-Server** il quale è un componente dell'infrastruttura di rete che acquisisce le informazioni di mapping EID-RLOC contenute nel Mapping Database dagli ETR; I LISP Map-Server sono concettualmente simili ai server DNS.

- **Map-Resolver** il quale accetta messaggi Map-Requests dall'ITR e determina, nel momento in cui sia presente, l'appropriata associazione EID-RLOC consultando il Mapping Server.

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.
- **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.

3.5 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 (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 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:

- 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 l'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.

3.6 Configurazione degli EID e registrazione degli ETR

Un ETR pubblica gli indirizzi del suo EID space sul Map-Server inviando periodicamente messaggi Map-Register (intervallo di 60 secondi fra i messaggi); il Map-Server dovrebbe rimuovere le associazioni nel Database di cui non riceve aggiornamento ogni 3 minuti [7]. 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.

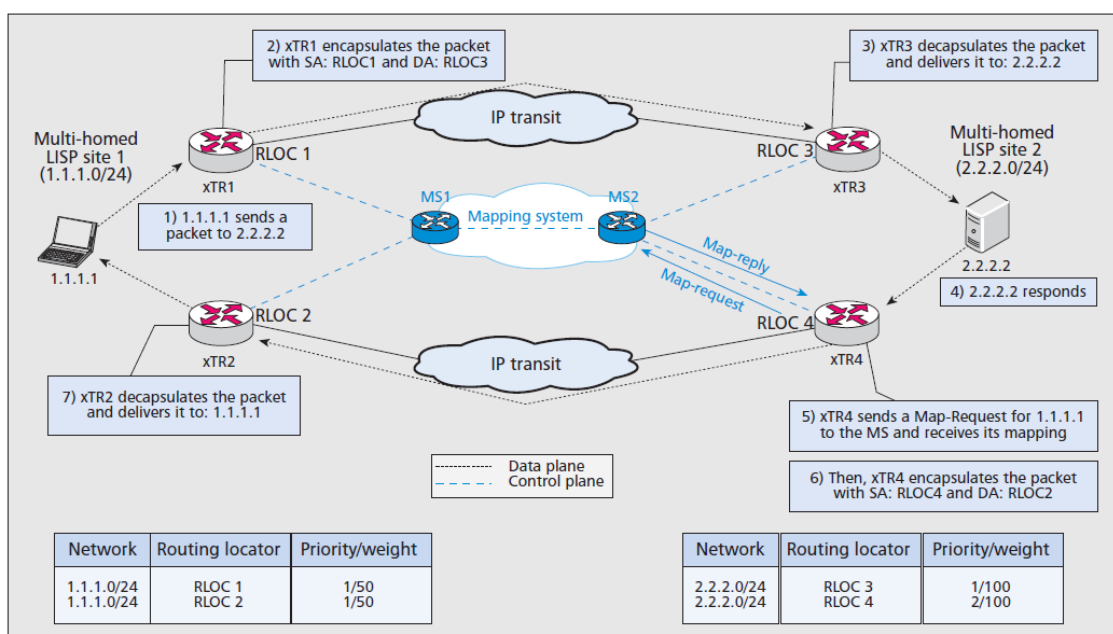


Figura 4 - Esempio di comunicazione tra due siti LISP

D'altro canto, oltre alla virtualizzazione delle funzionalità di rete (NFV) una nuova architettura di rete sta emergendo che prende il nome di Software Defined Network (SDN), il quale progetto prevede il disaccoppiamento tra l'hardware di forwarding e le decisioni di controllo quindi, nel caso delle NFV, la virtualizzazione riguarda le funzionalità di rete mentre nel caso dell'SDN la virtualizzazione riguarda i dispositivi di forwarding e la relativa logica di controllo.

4. Software Defined Network (SDN)

4.1 Rete di dati tradizionali

Nell'approccio tradizionale al networking, la maggior parte della funzionalità di rete è implementata in un sistema dedicato, cioè switch, router e applicazioni di controllo; nella maggior parte dei casi queste funzionalità sono istanziate su un hardware dedicato [8].

Le organizzazioni di networking sono soggette a una pressione sempre maggiore affinché siano più agili ed efficienti di quanto il tradizionale approccio al networking consenta. Una delle fonti di tale pressione è l'adozione diffusa della virtualizzazione dei server. Come parte della virtualizzazione dei server, le macchine virtuali (VM) vengono trasferite dinamicamente da un server all'altro in una manciata di secondi o minuti. Tuttavia, se il movimento di una VM attraversa un confine di Layer 3, possono occorrere giorni o settimane per riconfigurare la rete affinché supporti la VM nella sua nuova posizione. Talvolta il processo di riconfigurazione di una rete, affinché supporti il trasferimento di una VM, richiede settimane e quindi di certo la rete non si può definire agile. In sintesi, una rete tradizionale si evolve lentamente; le sue funzionalità sono limitate dalle caratteristiche offerte dai fornitori di ASIC e di appliance di rete. SDN mantiene la promessa di superare tali limitazioni.

4.2 Il passaggio al software

Come osservato, la rete di dati tradizionale è stata in larga misura centrata sull'hardware. Negli ultimi anni, tuttavia, l'adozione di appliance di rete virtualizzate e il crescente interesse verso i Software Defined Data Center (SDDC) hanno portato a una maggiore dipendenza dalle funzionalità di rete basate su software. Ciò significa che operazioni

come la crittografia/decrittografia e l'elaborazione dei flussi TCP venivano eseguite su un hardware che era stato progettato specificamente per tali funzioni. In larga parte a causa dell'esigenza di una maggiore agilità, ora è comune che le funzionalità WOC (WAN Optimization Controller) o ADC (Application Delivery Controller) siano fornite dal software in esecuzione su un server generico o su una macchina virtuale. Un SDDC può essere visto come l'esatto contrario della tradizionale rete di datacenter descritta in precedenza. Per esempio, una delle caratteristiche chiave di un datacenter software-defined è che tutta l'infrastruttura del datacenter è virtualizzata e distribuita come servizio. Un'altra caratteristica fondamentale è che il controllo automatizzato delle applicazioni e dei servizi del datacenter è fornito da un sistema di gestione basato su policy.

4.3 Le possibili opportunità

Una delle caratteristiche comuni a qualsiasi approccio radicalmente nuovo alla tecnologia è la confusione in merito alle opportunità che il nuovo approccio permette di esplorare. Al fine di valutare e adottare correttamente un nuovo approccio alla tecnologia come SDN, le organizzazioni IT devono identificare la singola o le molteplici opportunità per loro importanti che il nuovo approccio potrebbe permettere di affrontare in modo migliore. Dopo tutti i dibattiti su SDN degli ultimi due anni, le seguenti sono emerse come le opportunità più probabili che SDN può esplorare.

- Sostenere il movimento dinamico, la replica e l'allocazione delle risorse virtuali;
- Alleggerire il carico amministrativo della configurazione e del provisioning di funzionalità quali QoS e sicurezza;
- Implementare e scalare la funzionalità della rete in modo più semplice;
- Condurre l'ingegneria del traffico con una visione end-to-end della rete;
- Utilizzare meglio le risorse di rete;
- Ridurre l'OPEX;
- Favorire un'evoluzione della funzionalità di rete più rapida, basata sul ciclo di vita dello sviluppo del software;
- Consentire alle applicazioni di richiedere i servizi dalla rete in modo dinamico;
- Implementare funzionalità di sicurezza più efficaci;
- Ridurre la complessità.

4.4 Descrizione Software-defined networking

L'Open Networking Foundation (ONF) è il gruppo più strettamente associato con lo sviluppo e la standardizzazione del SDN. Secondo l'ONF, *"il Software-defined networking (SDN) è una nuova architettura dinamica, controllabile, conveniente e adattabile - una soluzione ideale per le applicazioni odierne, dinamiche e a banda larga. Tale architettura separa le funzioni di controllo e di forwarding della rete, consentendo al controllo di rete di essere direttamente programmabile e all'infrastruttura sottostante di essere utilizzata per applicazioni e servizi di rete. Il protocollo OpenFlow è un elemento fondamentale per la creazione di soluzioni SDN"* [9]. L'architettura SDN è:

- Direttamente programmabile: il controllo di rete è programmabile in modo diretto perché è disgiunto dalle funzioni di forwarding.
- Agile: la separazione delle funzioni di controllo e forwarding consente agli amministratori di regolare dinamicamente il flusso del traffico su tutta la rete per soddisfare le esigenze al loro variare.
- A gestione centralizzata: a livello logico, l'intelligence di rete è centralizzata in controller SDN basati su software che mantengono una visione globale della rete, che alle applicazioni e ai motori di policy appare come un singolo switch logico.
- A programmazione configurata: SDN consente ai responsabili di rete di configurare, gestire, proteggere e ottimizzare le risorse di rete molto rapidamente tramite programmi SDN automatici e dinamici, che i responsabili possono scrivere in modo autonomo perché i programmi non dipendono da software proprietari.

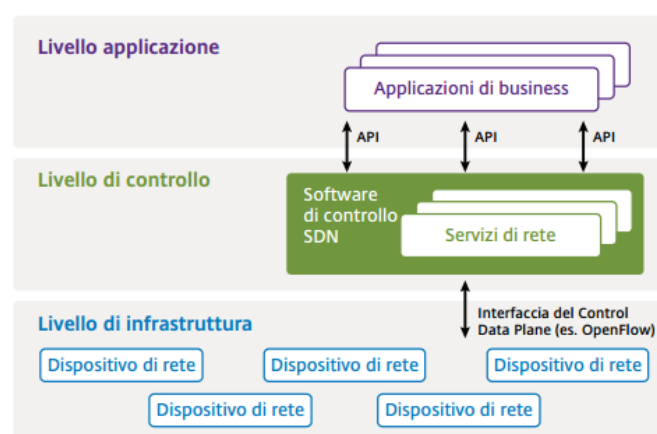


Figura 5 - Architettura SDN

Applicazioni di business

Si riferiscono alle applicazioni che sono direttamente fruibili dagli utenti finali. Le possibilità includono videoconferenze, la gestione della supply chain e la gestione delle relazioni con i clienti.

Rete e servizi di sicurezza

Si riferisce alla funzionalità che permette il funzionamento efficace e sicuro delle applicazioni di business. Le possibilità comprendono una vasta gamma di funzionalità L4 - L7, tra cui ADC, WOC e funzioni di sicurezza come firewall, protezione DDoS e IDS/IPS.

Switch SDN puro

In uno switch SDN puro, tutte le funzioni di controllo di uno switch tradizionale (per esempio, i protocolli di routing utilizzati per costruire le basi di inoltro delle informazioni) sono gestite dal controller centrale. La funzionalità dello switch è limitata esclusivamente al piano dei dati.

Switch ibrido

In uno switch ibrido, le tecnologie SDN e i protocolli di switching tradizionali vengono eseguiti contemporaneamente. Un gestore di rete può configurare il controller SDN affinché rilevi e controlli alcuni flussi di traffico, mentre i tradizionali protocolli di rete distribuiti continuano a dirigere il resto del traffico sulla rete.

Rete ibrida

Una rete ibrida è una rete in cui switch tradizionali e switch SDN, siano essi puri o ibridi, operano nello stesso ambiente.

API northbound

L'API northbound è l'API che permette la comunicazione tra il livello di controllo e il livello dell'applicazione aziendale. Attualmente non esiste un'API northbound basata su standard.

API southbound

L'API southbound è l'API che permette la comunicazione tra il livello di controllo e il livello dell'infrastruttura. Tra i protocolli che possono attivare queste comunicazioni vi sono OpenFlow, il protocollo di messaggistica istantanea e presenza (XMPP) e il protocollo di configurazione della rete.

Parte della confusione che circonda SDN è dovuta al fatto che molti fornitori non rientrano del tutto nella definizione di SDN fornita dall'ONF [\[10\]](#). Per esempio, mentre alcuni fornitori vedono in OpenFlow un elemento fondante delle loro soluzioni SDN, altri hanno un approccio più cauto nei suoi confronti. Un'altra fonte di confusione è il dissenso rispetto a ciò che costituisce il livello dell'infrastruttura. Per l'ONF, il livello dell'infrastruttura è una vasta gamma di switch e router fisici e virtuali.

4.5 Protocollo OpenFlow

OpenFlow è uno standard aperto che consente ai ricercatori di eseguire protocolli sperimentali nelle reti campus che usiamo ogni giorno. OpenFlow è aggiunto come una funzione per switch Ethernet commerciali, router e punti di accesso wireless e fornisce un gancio standardizzato per permettere ai ricercatori di eseguire esperimenti, senza la necessità di esporre il funzionamento interno dei loro dispositivi di rete [\[11\]](#).

4.5.1 Funzionamento

In un router o in uno switch classico l'inoltro del pacchetto (percorso di dati) e le decisioni di routing di alto livello (percorso di controllo) si verificano sullo stesso dispositivo. Uno switch OpenFlow separa queste due funzioni. Le operazioni di forwarding dei dati risiedono ancora nello switch, mentre le decisioni di routing di alto livello vengono spostate in un controller separato, in genere un server standard. Lo switch OpenFlow e il controller comunicano tramite il protocollo OpenFlow, che definisce i messaggi, come ad esempio i packet-in, send-packet-out, modifiche delle flow-tables ecc.

Il percorso dei dati di uno switch OpenFlow presenta una chiara flow-table; Ogni voce della tabella di flusso contiene una serie di campi di pacchetti da abbinare e un'azione (come send-out-port, edit-field, o drop). Quando uno switch OpenFlow riceve un pacchetto che non ha mai visto prima, per la quale non ha corrispondenze nella flow-table, invia il pacchetto al controller. Il controller compie una decisione su come gestire questo pacchetto. Può scartare il pacchetto, oppure può aggiungere una voce nella flow-table dello switch su come inoltrare pacchetti simili in futuro.

4.5.2 Openflow Switch

Uno switch OpenFlow è un dispositivo software o hardware che inoltra i pacchetti in una Software Defined Network (ambiente SDN) [\[12\]](#). Gli OpenFlow switch sono basati sul protocollo OpenFlow. In uno switch convenzionale, l'inoltro dei pacchetti (**data plane**) e il

controllo del routing (control plane) si verificano sullo stesso dispositivo. Nelle Software Defined Network, il piano dati è disaccoppiato dal piano di controllo. Il piano dati è ancora implementato in zona di comando ma il piano di controllo è implementato nel software dove un controller SDN separato prende le decisioni di routing di alto livello. Lo switch e il controller comunicano attraverso il protocollo OpenFlow.

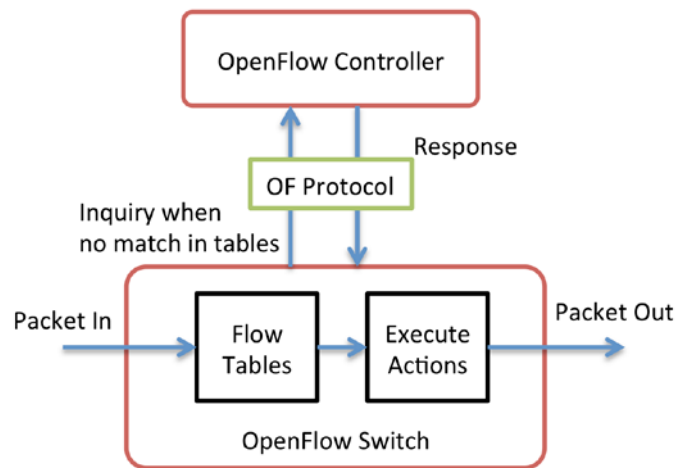


Figura 6 - Funzionamento OF Switch e OF Controller

4.5.3 Controller SDN

I Controller SDN (piano di controllo SDN) in una rete software-defined (SDN) sono il "cervello" della rete. Esso è "l'applicazione che funge da punto di controllo strategico della rete SDN, ha il compito di gestire il controllo dei flussi per gli switch/ router" (tramite protocollo Southbound) e le applicazioni e la logica di business (tramite protocollo Northbound). Un SDN Controller contiene in genere un insieme di moduli 'pluggable' che possono svolgere diverse attività di rete. Alcuni dei compiti di base tra cui supervisionare quali dispositivi sono all'interno della rete e le capacità di ciascuno, la raccolta di statistiche di rete, ecc inoltre è possibile aggiungere estensioni in grado di migliorare le funzionalità ed aggiungerne di più avanzate, come ad esempio l'esecuzione di algoritmi per l'esecuzione di analisi

5. SDN e NFV

Ultimamente si sta adottando una nuova visione cioè quella di adottare un modello di rete dinamico, flessibile e soprattutto affidabile, in grado di adattarsi ai cambiamenti del futuro senza richiedere grossi sforzi di manutenzione o l'installazione di ulteriore hardware da parte degli operatori [13]. Una rete con queste caratteristiche può essere sviluppata grazie ad un modello architetturale innovativo come il Software Defined Networking (SDN) e ad un nuovo modo di sfruttare le funzionalità degli apparati come la Network Function Virtualization (NFV). Questi due concetti sono strettamente legati tra di loro e possono comportare particolari vantaggi se applicati contemporaneamente, essi sono però di per se indipendenti.

Il Software Defined Networking (SDN) è un'architettura per la realizzazione di reti di telecomunicazioni nella quale il piano di controllo della rete e quello di trasporto dei dati sono separati logicamente. Questa separazione logica permette da un lato la possibilità di gestire via software tutta la rete da un unico controller, garantendo così una maggiore scalabilità e standard di affidabilità e sicurezza della rete più elevati, dall'altro quella di utilizzare indifferentemente apparati prodotti dalle diverse aziende che non conterranno più al loro interno le funzioni di gestione favorendo, così, la nascita di una rete dinamica e non più legata all'elevato numero di differenti protocolli attualmente utilizzati.

La Network Function Virtualization (NFV) è il processo di virtualizzazione delle funzionalità di rete svolte da apparati di telecomunicazione fisici. I maggiori vantaggi che l'operatore può trarre dall'utilizzo della NFV derivano sostanzialmente dal fatto di non essere più vincolati all'hardware (switch, server, apparati di storage, ecc.) necessario per introdurre nuovi servizi in rete.

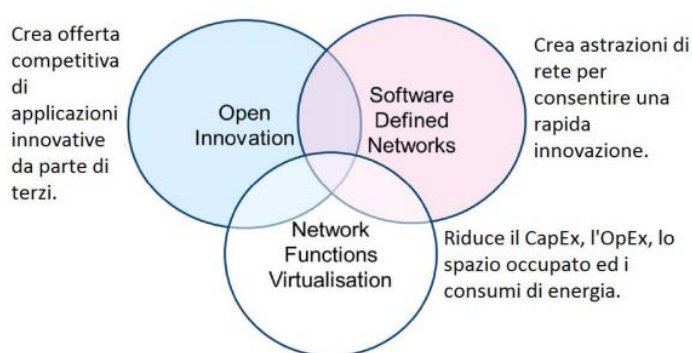


Figura 7 - SDN & NFV

Grazie alla NFV è possibile assicurare una rete più flessibile ed affidabile, potendo spostare le funzioni virtuali (VF - Virtual Functions) da un server fisico ad un altro, oltre a fornire servizi personalizzati agli utenti semplicemente rimodulando il software di gestione degli apparati.

La sinergia delle soluzioni SDN ed NFV, permette alla rete di raggiungere le migliori performance. Infatti, l'SDN fornisce alla NFV i vantaggi di una connessione programmabile tra le funzioni di rete virtualizzate; la NFV, invece, mette a disposizione dell'SDN la possibilità di implementare le funzioni di rete tramite software su server COTS (Commercial off-the-shelf). Si ha, così, la possibilità di virtualizzare il controller SDN implementandolo su di un cloud che può essere facilmente migrato in qualsiasi posizione in base alle esigenze della rete.

Una possibile soluzione di fusione tra architettura SDN e NFV è stata presentata nel progetto dal nome: Smart SDN Forwarding Device.

6. Smart SDN Forwarding Device (SSFD)

La caratteristica principale di SSFD è fornire ai dispositivi SDN Data-Plane il supporto di specifiche funzioni di control-plane come ad esempio le Network Functions(NFs) [\[14\]](#). La disponibilità delle NFs a livello degli switch consente di eseguire specifiche operazioni di control-plane a livello locale, in modo da ridurre il carico sul controller SDN e permette anche di eseguire diversi protocolli di rete negli switch SDN, fornendo così una soluzione per la realizzazione di reti ibride IP / SDN.

Il Data Plane implementa la tabella di inoltro del dispositivo, rispettando le specifiche OpenFlow. Il Control Plane è composto da due elementi, la NF Manager e NFS. Il NF Manager rappresenta un hypervisor leggero in grado di gestire le NFS e di interagire con il piano dati, mediante una interfaccia Southbound ad-hoc. Una funzione di rete (NF) è un elemento del livello di controllo che fornisce il supporto per un protocollo specifico di rete. Esempi di NFS sono i protocolli di routing come: RIP, OSPF, BGP o le funzioni di rete avanzate. Le NF devono essere coinvolte solo nella gestione dei pacchetti al livello di controllo, e non nel funzionamento di inoltro dei pacchetti di dati infatti questi ultimi vengono gestiti direttamente al piano dati. Per riassumere il funzionamento di un SSFD, i pacchetti di dati sono elaborati a livello Data Plane (percorso veloce), mentre per il controllo dei pacchetti questi vengono inviati al gestore NF che deve consegnarli alla NF appropriata (lento percorso).

Per il data-plane è stato utilizzato OpenvSwitch, per il control-plane è stato utilizzato Floodlight infine come NF è stata focalizzata l'attenzione verso il protocollo LISP (Locator

Identifier Separation Protocol) ed è stato utilizzato il software OpenLISP CP. Per la comunicazione Southbound tra OVS ed il controller Floodlight è stato utilizzato il protocollo OpenFlow mentre per la comunicazione Northbound tra il controller e OpenLISP CP sono state utilizzate delle AF_UNIX Socket.

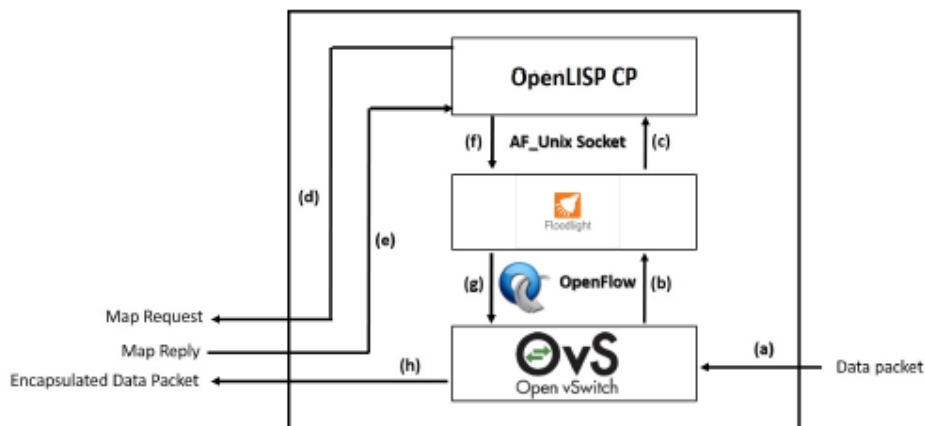


Figura 8 - Architettura SSFD

In questo lavoro di tesi è stato analizzato un approccio simile allo SSFD ma rivisitato. Tale rivisitazione prevede l'eliminazione del controller quindi una semplificazione dell'architettura stessa la quale comporta un minore costo computazionale. A differenza del progetto SSFD però il controller LISP (OpenLISP CP) è stato installato su un Linux Container il quale prevede una virtualizzazione molto più leggera di una normale Virtual Machine. Questa scelta è stata presa per facilitare la migrazione delle Network Functions (NFs) rispettando in pieno gli standard delle NFV e delle SDN.

7. Hybrid-SDN node with LISP functions using container virtualization

7.1 Descrizione

La soluzione pensata in questo progetto di tesi prevede l'eliminazione del controller di rete SDN alleggerendo così la complessità dell'intera architettura.

Al posto di utilizzare un controller e quindi adoperarlo per andare ad inserire le regole in OVS è stato pensato di progettare due software: un programma client il quale si occupa

del reperimento delle associazioni EID-RLOC dal software OpenLISP cp e un programma server il quale ha il compito di ricevere queste associazioni dal client e di inserire le regole nello switch [36]. In seguito verranno spiegate dettagliatamente le funzionalità dei due software sviluppati.

Nel data-plane (OvS) sono state inserite determinate regole di flusso in grado di gestire i vari casi che potrebbero presentarsi.

I casi gestiti dallo switch sono 3:

1. I pacchetti dati LISP possono arrivare dalla rete LISP, e in questo caso l'interfaccia dedicata del router deve decapsularli e fare forwarding verso l'EID space.
2. I pacchetti dati LISP possono arrivare dalla rete EID, e in questo caso l'interfaccia dedicata deve incapsularli e fare forwarding verso la porta che si occupa di fare forwarding verso l'RLOC space.
3. I pacchetti di controllo LISP arrivano al router (indipendentemente dall'interfaccia) e OvS deve inviarli al LISP Control Plane installato sul Linux Container.

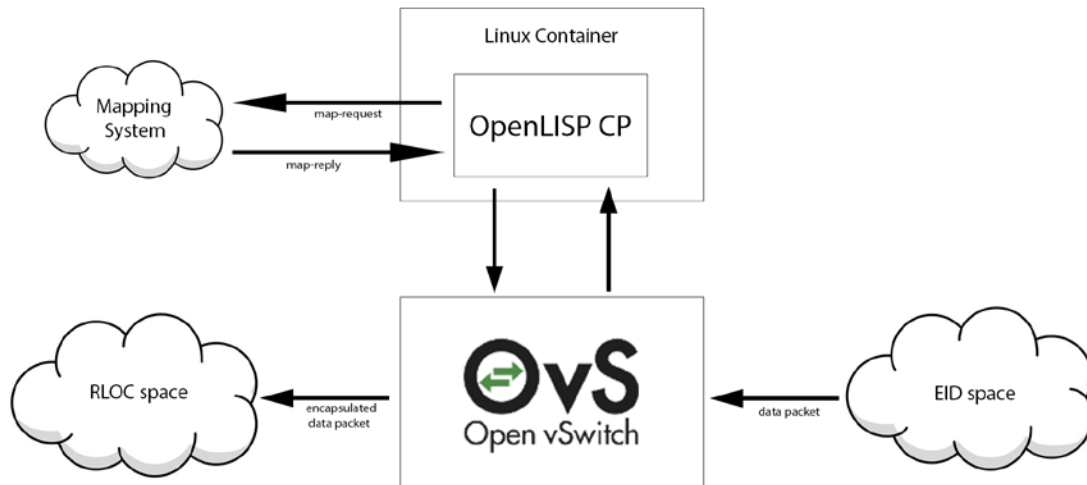


Figura 9 - Hybrid-SDN node with LISP functions using Container Virtualization Architecture

7.2 Software Utilizzato

Per la realizzazione di questo progetto abbiamo utilizzato i seguenti software: Open vSwitch per il dataplane, LXC per creare un ambiente virtualizzato leggero e OpenLISP CP per il mapping tra gli EID e gli RLOC ed infine la comunicazione tra il data-plane ed il control-plane è stata realizzata mediante due software scritti in python; in seguito una descrizione dei software sopra elencati.

7.2.1 Open vSwitch

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 [\[15\]](#). OVS supporta diverse tecnologie di virtualizzazione basate su Linux, incluse Xen/XenServer, KVM e VirtualBox. Il cuore del codice è scritto in platform-independent C ed è facilmente portabile in altri ambienti. La versione corrente di OVS - la **2.6.1**, utilizzata nella nostra implementazione supporta le seguenti caratteristiche:

- VLAN (standard 802.1Q) con porte access e trunk
- Bonding delle NIC con o senza LACP
- NetFlow, sFlow e mirroring per un'aumentata visibilità
- Configurazioni QoS e policing
- GRE, GRE su IPSEC, VXLAN3 e LISP tunneling
- Protocollo 802.1ag per la connectivity fault management
- Supporto a OpenFlow v1.4 e numerose estensioni
- Configurazione transazionale del database, con collegamenti C e Python
- Forwarding a elevate prestazioni tramite l'utilizzo del modulo del kernel Linux
- Supporto al NAT sui kernel Linux.
- Tracking delle connessioni per DPDK e Hyper-V.
- Supporto al Traffic policer per DPDK.
- Tunnel IPv6.

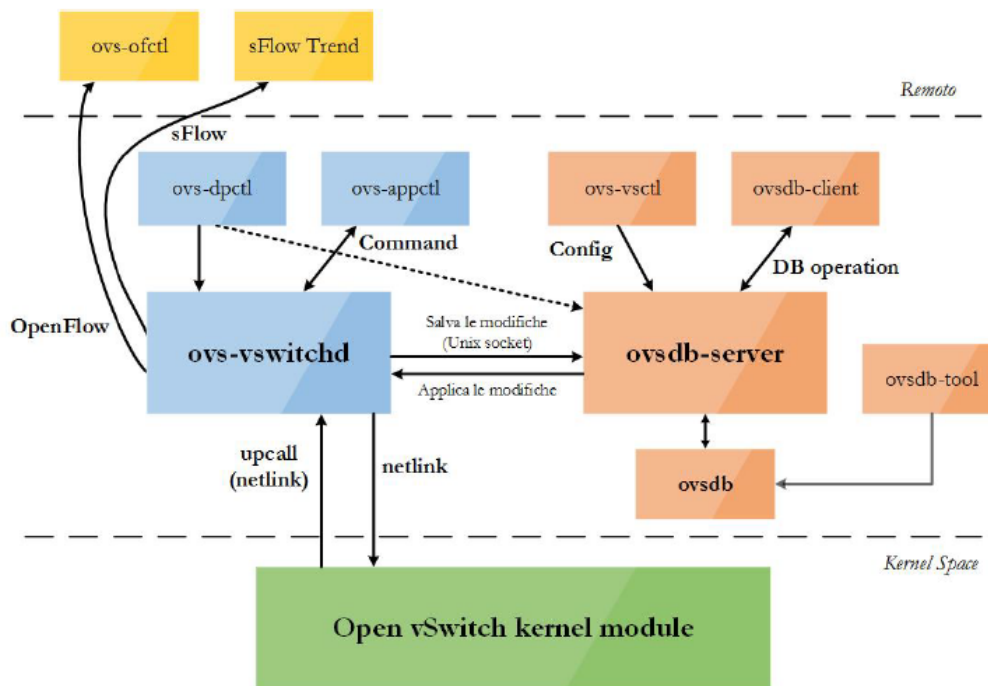


Figura 10 - Interazione dei componenti di Open vSwitch

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.

Le principali componenti di questa distribuzione sono:

1. **ovs-vswitchd**, un demone che implementa le funzionalità dello switch, insieme al modulo del kernel Linux per la commutazione basata su usso (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 successivi pacchetti del flusso (per cui il primo pacchetto ha un processing più 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.

7.2.1.1 Architettura e funzionamento 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). `Ovsvswitchd` 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).

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 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-vswitchd`; stabilisce pertanto in maniera standardizzata i parametri per la comunicazione fra controller e vSwitch.

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).

7.2.2 Linux Container LXC

La tecnologia dei container Linux consente di creare pacchetti di applicazioni e di isolarli insieme all'intero ambiente di runtime, ovvero con tutti i file necessari per l'esecuzione. In questo modo è possibile trasferire le applicazioni contenute nei pacchetti tra diversi ambienti (sviluppo, test, produzione ecc.), mantenendone la piena funzionalità [\[16\]](#).

I container Linux semplificano la collaborazione tra i team di sviluppo e operativi, grazie alla possibilità di definire le responsabilità. Gli sviluppatori possono concentrarsi sullo sviluppo delle applicazioni, mentre i team operativi si dedicano all'infrastruttura. I container Linux sono basati su tecnologie open source, dando la possibilità di usufruire delle innovazioni più recenti e di semplificare, accelerare e organizzare in modo efficace lo sviluppo e il deployment delle applicazioni.

7.2.2.1 Differenza tra container e macchine virtuali

Rispetto a un'intera macchina virtualizzata, un container è capace di offrire [\[17\]](#):

- **Un deployment semplificato:** impacchettando un'applicazione in un singolo componente distribuibile e configurabile con una sola linea di comando, la

tecnologia a container permette di semplificare il deployment di qualsiasi applicazione, senza doversi preoccupare della configurazione dell'ambiente di runtime;

- **Una disponibilità rapida:** virtualizzando ed astraendo solo il sistema operativo e le componenti necessarie all'esecuzione dell'applicazione, invece che l'intera macchina, l'intero package si avvia in un ventesimo di secondo, rispetto ai tempi di avvio di una VM;
- **Un controllo più granulare:** i container consentono agli operatori e agli sviluppatori di suddividere ulteriormente le risorse computazionali in microservizi, garantendo così un controllo superiore sull'eseguibilità delle applicazioni e un miglioramento delle prestazioni dell'intero sistema.

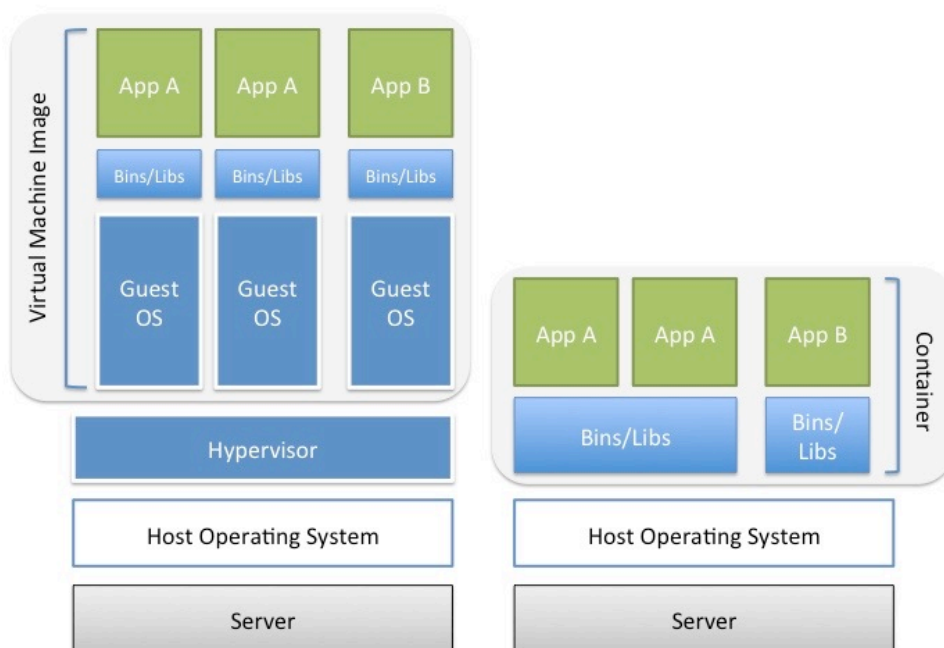


Figura 11 - Linux Container vs Virtual Machine

7.2.2.2 Vantaggi

Queste caratteristiche proprie della tecnologia dei container portano ad alcuni vantaggi indiscutibili:

1. L'opportunità per gli sviluppatori di possedere una miriade di container anche sul proprio PC o laptop, per avere sempre a portata di mano un ambiente di deploy o test adatto a ciascuna applicazione in sviluppo. Per quanto sia possibile eseguire

anche su un laptop diverse virtual machine, questa operazione non è mai veloce e semplice e impatta non poco sulle prestazioni esecutive;

2. Allo stesso modo, l'amministrazione dei cicli di rilascio delle applicazioni è semplificato, in quanto distribuire una nuova versione di un container è pari al tempo speso per digitare in console una singola linea di comando.
3. Anche le attività di testing traggono un beneficio economico da un ambiente contenierizzato. Se si effettua il test di un'app direttamente su un cloud server in un ambiente di cloud computing pubblico, sarà necessario sostenere i costi relativi alla frazione di ora (o all'ora intera) di occupazione delle risorse computazionali. Questo costo aumenta all'aumentare del numero di test che devono essere eseguiti. Con un container è possibile effettuare una serie di semplici test programmati giornalieri mantenendo costante il costo, in quanto si userebbero sempre le stesse risorse di calcolo.
4. Infine, non si può ignorare la componibilità dei sistemi applicativi, specialmente per le applicazioni open source. In pratica, invece di obbligare gli sviluppatori a installare e configurare i servizi MySQL, memcached, MongoDB, nginx, node.js e via scorrendo, per avere la giusta piattaforma esecutiva per le proprie applicazioni, sarebbe meno rischioso e più veloce avviare ed eseguire con piccoli script quei pochi container che ospitano queste stesse applicazioni.

Sul lungo periodo, il più importante vantaggio che questa tecnologia promette è la portabilità e la consistenza di un formato che consente l'esecuzione applicativa su diversi host. Infatti, con la standardizzazione dei container, i workload possono essere facilmente spostati lì dove vengono eseguiti in modo più economico e veloce, evitando anche i lock-in dovuti alle peculiarità delle piattaforme dei singoli provider.

7.2.2.3 LXC

LXC (abbreviazione di Linux Containers) è un ambiente di virtualizzazione a *container*, che opera a livello del sistema operativo e permette di eseguire diversi ambienti Linux virtuali isolati tra loro (*container*) su una singola macchina reale avente il kernel Linux [\[18\]](#).

Il kernel Linux fornisce due funzionalità fondamentali per LXC:

1. *cgroups*, che permette di gestire limiti e priorità di utilizzo delle risorse (CPU, memoria, accesso alla memoria, accessi ai dischi, rete) senza dover utilizzare una vera e propria macchina virtuale;

2. l'isolamento dei *namespace* (o spazio dei nomi), che permette di isolare tra loro gruppi di processi, affinché non siano loro visibili risorse utilizzate da altri gruppi di processi, come alberi di processi, risorse di rete, user ID e file system montati.^[3]

Tali funzionalità sono fornite dal kernel *vanilla*; la funzionalità cgroups è ad esempio implementata dalla versione 2.6.24. Ciò permette a LXC di poter essere utilizzato senza dover modificare il kernel con patch, come invece accade per soluzioni simili.

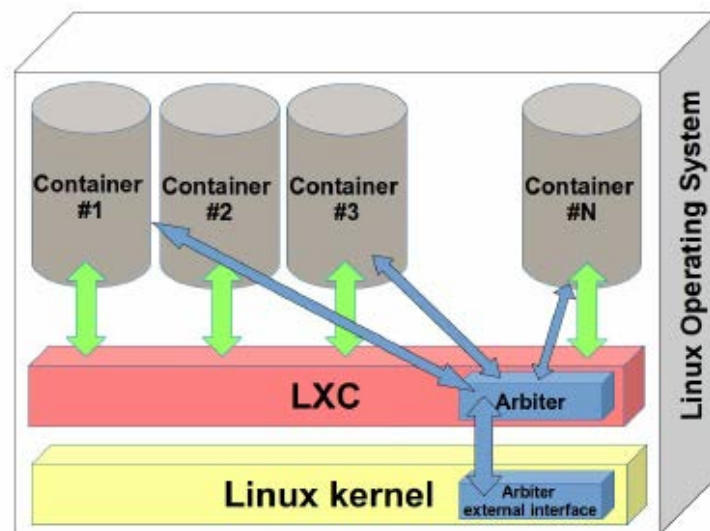


Figura 12 - Architettura LXC

7.2.3 OpenLISP

OpenLISP è un'implementazione open source del protocollo LISP in esecuzione nel kernel del sistema operativo FreeBSD [\[19\]](#). OpenLISP si concentra sul funzionamento del piano dati, il che significa che implementa il LISP-Cache e il LISP-Database nello spazio del kernel, così come le funzioni Incapsulamento/Decapsulamento. Tutto ciò che è legato al piano di controllo è destinato per l'esecuzione nello spazio utente. OpenLISP non fornisce alcuna implementazione dello specifico protocollo control-plane. OpenLISP implementa un nuovo tipo di comunicazione, chiamato Mapping Socket il quale fornisce un'API che può essere utilizzata da qualsiasi processo user-space.

Il progetto prevede due implementazioni, una relativa al data-plane ed una relativa al control-plane. Entrambi sviluppate per i sistemi che operano su kernel FreeBSD ma ultimamente l'implementazione di OpenLISP control-plane è stata adattata anche per i sistemi operativi Linux.

Il motivo fondamentale della creazione di un control plane è quello che OpenLISP non è in grado di gestire tutti i segnali di controllo all'interno di una rete LISP per questo

OpenLISP control-plane mira a colmare questa lacuna, mantenendo i dati e il piano di controllo indipendenti l'uno dall'altro ottenendo così anche benefici a livello prestazionale.

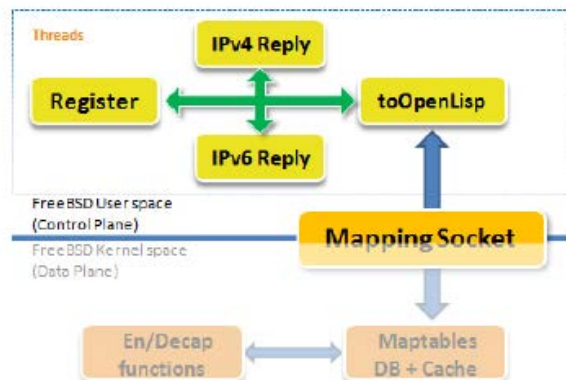


Figura 13 - OpenLISP

In questo lavoro di tesi è stato utilizzato il software OpenLISP control-plane per la gestione del piano di controllo LISP.

7.2.3.1 OpenLISP Control Plane

Per motivi di scalabilità, l'ITR apprende in tempo reale le mappature tramite il mapping system. Il mapping system è composto dal mapping database system e il map-server.

Il funzionamento del mapping-system è riassunto nella Fig. 14.

Da un lato, il mapping database system costituisce l'infrastruttura che memorizza le mappature su scala globale utilizzando degli algoritmi complessi [20]. Dall'altro canto, l'interfaccia del mapping server semplifica questa complessità attraverso due elementi di rete, il map resolver (MR) e il map server (MS), sviluppato sul bordo del mapping database system i quali, vengono contattati dai siti LISP per recuperare e registrare le mappature.

Più precisamente, quando un ITR è disposto a ottenere una mappatura per un dato EID, invia un messaggio Map-Request a un MR. Il MR è collegato al mapping database system e implementa la logica di ricerca map-request per determinare a quale sito LISP deve essere consegnato (a uno qualsiasi dei suoi ETRS), e lo spedisce. L'ETR riceve la query e restituisce la mappatura direttamente all'ITR richiedente con un messaggio Map-Response. Vale la pena notare che l'ETR di un sito LISP non è direttamente coinvolto nel database mapping system ma è invece collegato ad un MS. L'ETR invia un messaggio Map-

Register al MS, il quale si assicura che il mapping sia registrato nel database mapping system. Opzionalmente, il MS può riconoscere la registrazione con un messaggio Map-Notify.

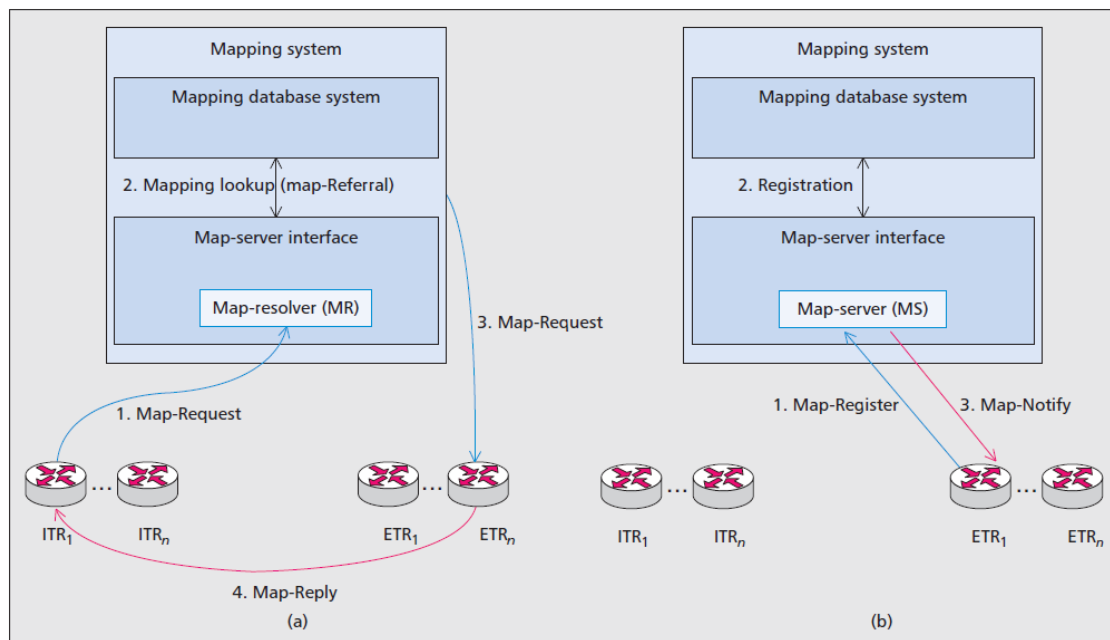


Figura 14 - Funzionamento LISP mapping system

7.2.3.1.1 Architettura OpenLISP Control Plane

Il ruolo principale del piano di controllo LISP è la gestione delle mappature EID-to-RLOC con il mapping system, in seguito si focalizzerà l'attenzione sulla progettazione del Mapping database, per poi spiegare in dettaglio i diversi moduli [21].

7.2.3.1.2 Mapping System e principali nodi di rete

Il cuore del piano di controllo OpenLISP è l'EID-to-RLOC mapping database, sinteticamente indicato come MapTable. Ogni map-entry della map-table è costituita da un prefisso EID con un elenco di RLOCs, ciascun RLOC è associato ad una struttura che contiene l'indirizzo RLOC e gli attributi correlati (vale a dire, priorità e peso). I tre elementi di rete coinvolti nel piano di controllo sono ETR, MS e MR i quali hanno scopi diversi; essi attuano la loro logica map-table nel seguente modo.

Le map-entries dell'ETR corrispondono alle mappature per i diversi prefissi EID del sito LISP le quali vengono registrate tramite un MS. Ciascun map-entry deve avere almeno un indirizzo RLOC.

Esistono due tipi di map-entry:

- Registered map-entry: sono costruite sui messaggi map-register ricevuti dall'ETRS. Il MS può utilizzare queste entries per rispondere direttamente ai messaggi map-request per conto dell'ETR.
- Negative map-entries: sono usati per definire il range di prefissi IP che appartengono allo spazio EID, ma che non richiedono un incapsulamento LISP. Le richieste di tali prefissi generano un map-response negativo.

Il Map-resolver mantiene una map-table per accelerare la mappatura e si distingue per due tipi di voci:

- Negative map-entries: sono simili ai negative map-entries di un MS. Un MR invia quindi immediatamente una negative map-response per i prefissi EID non ancora assegnati.
- Referral map-entries: contengono gli indirizzi di altri nodi DDT (MR) che dovrebbero fornire più specifiche mappature LISP-DDT (vale a dire, hanno un prefisso EID prefisso più lungo).

7.2.3.1.3 Control Plane Modules

OpenLISP control plane include le caratteristiche essenziali per gestire una rete LISP multi-homed, tra cui tutta la logica LISP-DDT e il supporto completo di IPv4 e IPv6. Il piano di controllo riceve i messaggi da una coda di socket UDP del kernel. Il control-plane si basa su un processo generale di controllo e tre processi specializzati che implementano le funzionalità di MR, MS e xTR. Il trattamento dei messaggi mapping-resolution-related e dei messaggi registration-related è isolato grazie all'utilizzo dei threads. Ogni processo è composto da diversi moduli, come descritto nel seguito.

Il processo **xTR** comprende i seguenti tre moduli:

- Modulo *map-register*: Implementato sull'interfaccia ETR; invia informazioni periodiche (ogni 60 s) sulla map-entry registration ad almeno un MS. Si noti che gli ETRS sono autenticati da un MS utilizzando la loro chiave condivisa preconfigurata. Al fine di sostenere il mapping-system multi-tenancy, andando oltre gli standard attuali, il modulo permette di specificare chiavi diverse per MS diversi consentendo così ad un xTR di gestire siti LISP da MS diversi.

- Modulo *map-reply*: Implementato all'interfaccia ETR, riceve ed elabora le richieste provenienti dall'ETR o dal MS. Secondo lo standard, le map-request devono essere incapsulate (Control Message Encapsulated, ECM) quando un Map-Request è inviato al MR, vengono poi inviati in modo nativo all'ETRS.
- Modulo *plane-internetworking*: Questo modulo permette al piano di controllo di interagire con il piano dati e quindi di formare un vero e proprio OpenLISP xTR. Per eseguire le funzioni di data-plane, il piano dati OpenLISP mantiene una mappatura base di informazioni (MIB) costituita dalla cache di LISP (Memorizzazione di brevi mappature in maniera on-demand) e database di LISP. OpenLISP fornisce anche un'astrazione di basso livello chiamato Mapping Socket per aggiungere o rimuovere le mappature da MIB localmente sulla macchina (ad esempio, mediante un demone o utilizzando la riga di comando). Questo Modulo di interworking utilizza il control-plane per mantenere il database interagendo con il piano dati attraverso la Mapping Socket.

Il processo di **MS** comprende i seguenti due moduli:

- Modulo *map-register*: Implementato sull'interfaccia del MS, riceve i messaggi Map-Register dall'ETRS e aggiorna le MS map-tables. Il MS verifica l'autenticità dei messaggi Map-Register e garantisce che i loro prefissi EID appartengono ai siti LISP di cui è responsabile. Per migliorare le prestazioni, il control-plane taglia il messaggio di Map-Register per vedere se la mappatura è cambiata dopo l'ultima registrazione. Se l'ETR richiede una notifica, un messaggio di Map-Notify viene rispedito al ETR.
- Modulo *map-request*: Al momento della ricezione del Map-Request, il modulo può scegliere tra due azioni a seconda delle map-table entry che corrispondono all'EID nella Map-Request. Se l'EID corrisponde al prefisso EID di una registered map-entry, il MS invia una map-reply indietro o invia una map-request ad uno degli RLOC presenti nella map-entry, a seconda del valore del bit proxy nel messaggio Map-Register. Se invece, l'EID corrisponde a un sito gestito dal MS ma non ha nessuna registrazione attiva, una risposta Map-Reply negativa viene inviata indietro.

Il processo di **Map-Resolver** contiene i seguenti due moduli:

- Modulo *map-request*: Accetta e processa i Map-Request dall'xTR. Per la segnalazione DDT, il Map-Request segue la catena di map-referral fino a

raggiungere un MS o un ETR. Per accelerare le prestazioni, la MR nasconde i messaggi map-referral nella sua map-table in modo che possa riutilizzarla per ulteriori map-request coperte dal prefisso EID.

- Modulo *map-referral*: accetta il LISP-DDT Map-Request a cui risponde con un messaggio di map-referral.

Infine, il **processo di controllo** ha lo scopo di amministrare gli altri processi. E' responsabile della ricezione di messaggi di control-plane dalla rete LISP e dell'invio verso l'appropriato processo di control-plane. Una coda FIFO (First In First Out) è utilizzata per assorbire le domande entranti e cattura i messaggi provenienti dalla coda kernel socket UDP. Questo processo popola anche la map-table, utilizzata dai processi di control-plane, con lo scopo di configurare il dispositivo.

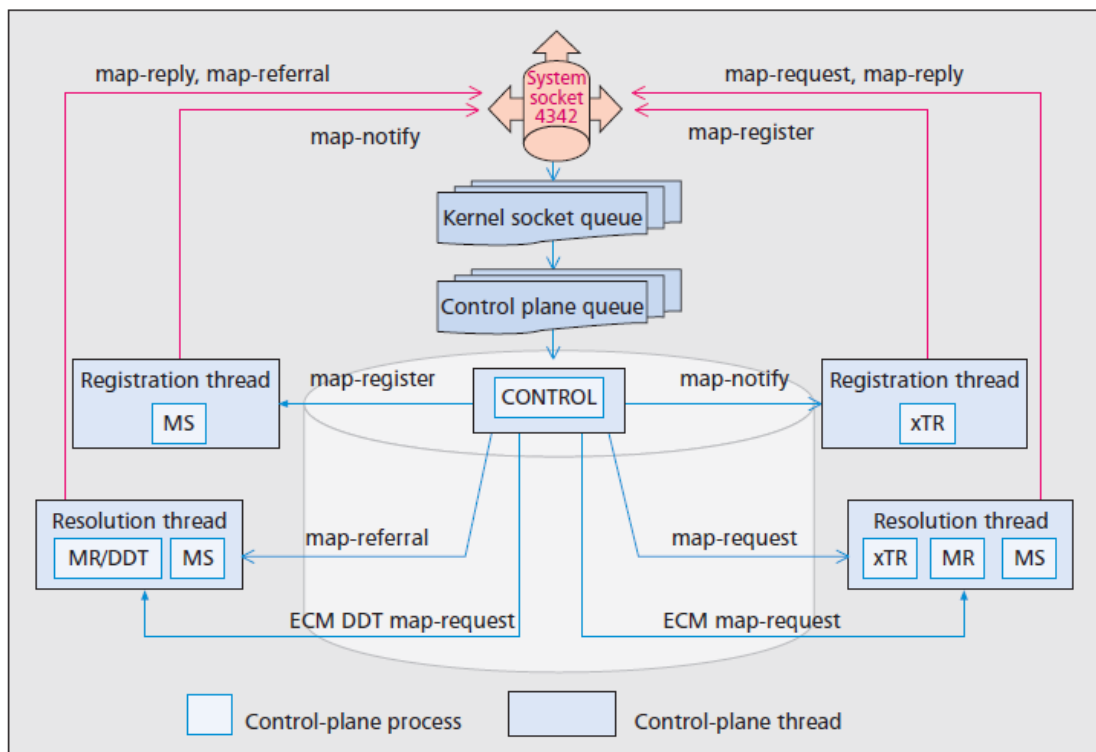


Figura 15 - Architettura OpenLISP Control Plane

7.2.3.1.4 Running the OpenLISP Control Plane

Il processo di control plane OpenLISP è in ascolto sulla porta UDP 4342 la quale è la porta di controllo LISP. E' utilizzata per consentire una più facile programmabilità delle sue caratteristiche, mentre le funzioni di data plane OpenLISP vengono eseguite nel kernel BSD in modo da fornire prestazioni più elevate.

I processi principali del control-plane prevedono tre diversi threads eseguiti nel main thread principale: con un thread viene eseguito il processo di controllo, un thread è dedicato alle map-registration, e gli altri thread sono dedicati ai processi di Map-Request/Referral (resolution threads).

Il thread principale accetta i pacchetti di controllo LISP provenienti dalla coda della socket del kernel e li manda ad una coda FIFO di control plane. Per il bilanciamento del carico, il thread di controllo può creare dinamicamente diversi thread fino a un numero massimo, il quale è possibile impostarlo modificando un parametro presente nel file di configurazione (opencp.conf). La scelta di utilizzare più thread pre-istanziato per i messaggi di processo control plane e creare una coda di pacchetti per il control plane alimentato dalla coda socket kernel è dettata dalla scalabilità e robustezza contro gli attacchi.

7.2.4 Client-Server Socket Communication

Il software sviluppato per la comunicazione tra OpenvSwitch presente sulla macchina e OpenLISP cp presente sull'LXC prevede delle comunicazioni Client-Server utilizzando il meccanismo delle socket.

7.2.4.1 Socket

Una socket è *oggetto software* che permette l'invio e la ricezione di dati, tra host remoti (tramite una rete) o tra processi locali (Inter-Process Communication) [35]. Più precisamente, il concetto di socket si basa sul modello Input/Output su file di Unix, quindi sulle operazioni di *open*, *read*, *write* e *close*; l'utilizzo, infatti, avviene secondo le stesse modalità, aggiungendo i parametri utili alla comunicazione, quali indirizzi, numeri di porta e protocolli [22].

Socket locali e remoti in comunicazione, formano una *coppia* (pair), composta da indirizzo e porta di client e server. Solitamente i sistemi operativi forniscono delle API per permettere alle applicazioni di controllare e utilizzare i socket di rete.

7.2.4.1.1 Famiglie di socket

I tipi di protocolli utilizzati dal socket, ne definiscono la famiglia (o dominio). Possiamo distinguere, ad esempio, due importanti famiglie [\[23\]](#):

- **AF_INET**: comunicazione tra host remoti, tramite Internet;
- **AF_UNIX**: comunicazione tra processi locali, su macchine Unix. Questa famiglia è anche chiamata *Unix Domain Socket*.

All'interno della famiglia possiamo distinguere il tipo di socket, a seconda della modalità di connessione. Abbiamo:

- **Stream socket**: orientati alla connessione (connection-oriented), basati su protocolli affidabili come TCP o SCTP;
- **Datagram socket**: non orientati alla connessione (connectionless), basati sul protocollo veloce ma inaffidabile UDP;
- **Raw socket** (*raw IP*): il livello di trasporto viene bypassato, e l'header è accessibile al livello applicativo.

Il protocollo utilizzato per questa comunicazione Client-Server è il protocollo AF_INET ed il tipo di socket utilizzata è la Datagram Socket poiché dobbiamo essere in grado di intercettare i messaggi di OpenLISP cp il quale, come visto prima, utilizza il protocollo UDP [\[24\]](#).

7.2.4.2 Funzionalità Client

Questo progetto prevede che il client venga istanziato sul container LXC [\[25\]](#). Esso prevede:

- Una socket in ascolto sulla porta 4342 dell'interfaccia eth0 del container la quale, nel momento in cui riceve un pacchetto UDP sulla porta di destinazione 4342 avvia la parte logica del programma.

- Una socket in grado di comunicare con il server in ascolto sul pc la quale si occupa anche del reperimento degli indirizzi EID-RLOC facendo un parsing del file XML dove sono contenute le associazioni. Una volta ottenute queste associazioni il client le invia al server.

La seconda funzionalità viene scandita ad intervalli di tempo regolari (60 secondi) il quale è il tempo che passa tra una map-request e l'altra del mapping service per ottenere le associazioni nel momento in cui le associazioni EID-RLOC non siano presenti nella map-cache; quindi ad ogni scansione il client fa un parsing del file e nel momento in cui queste associazioni fossero diverse dalle precedenti le invia al server, in caso contrario le scarta.

7.2.4.3 Funzionalità Server

In questo caso il server viene istanziato sulla macchina (nel nostro caso il PC). Questo software prevede:

- Una socket in ascolto dei messaggi del client, una volta ricevute le informazioni prevede a splittare gli interi indirizzi in EID-RLOC e ad inserire le regole in OVS con gli indirizzi ottenuti.
- Invio di un messaggio di operazione avvenuta con successo al client per poter continuare ad utilizzare correttamente quest'ultimo.

8. Installazione Componenti

8.1 Installazione Open vSwitch

Procediamo all'installazione della versione di Open vSwitch(2.6.1) [\[26\]](#).

Installiamo le dipendenze:

```
# apt-get update
```

```
# apt-get install python-simplejson python-qt4 python-twisted-conch automake  
autoconf gcc uml-utilities libtool build-essential pkg-config
```

```
# apt-get install libssl-dev iproute tcpdump linux-headers-4.4.0-51-generic
```

Download Open vSwitch

```
# wget http://openvswitch.org/releases/openvswitch-2.7.0.tar.gz
```

```
# tar xf openvswitch-2.6.1.tar.gz
```

```
# cd openvswitch-2.6.1/
```

Installiamo Open vSwitch

```
# ./boot.sh
```

```
# ./configure --with-linux=/lib/modules/4.4.0-51-generic/build
```

```
# make && make install
```

Carichiamo il modulo Open vSwitch nel kernel

```
# cd openvswitch-2.6.1/
```

```
# pwd
```

```
# cd datapath/linux
```

```
# modprobe openvswitch
```

```
# lsmod | grep openvswitch
```

In caso di errore:

```
# modprobe openvswitch
```

```
# modprobe gre
```

```
# cd datapath/linux
```

```
# insmod openvswitch.ko
```

File necessari e cartelle:

```
# touch /usr/local/etc/ovs-vswitchd.conf
```

```
# mkdir -p /usr/local/etc/openvswitch
```

Creiamo il file conf.db nella cartella openvswitch-2.6.1

```
# cd ../openvswitch-2.6.1  
# ovsdb-tool create /usr/local/etc/openvswitch/conf.db
```

Lanciamo il server (comando tutto attaccato) [\[27\]](#)

```
# ovsdb-server --remote=punix:/usr/local/var/run/openvswitch/db.sock  
--remote=db:Open_vSwitch,Open_vSwitch,manager_options  
--private-key=db:Open_vSwitch,SSL,private_key  
--certificate=db:Open_vSwitch,SSL,certificate  
--bootstrap-ca-cert=db:Open_vSwitch,SSL,ca_cert --pidfile -detach  
  
# ovs-vsctl --no-wait init
```

Avviamo vswitchd

```
# ovs-vswitchd --pidfile --detach
```

Verifichiamo i moduli kernel

```
# ovs-vsctl --version
```

8.2 Installazione del Linux Container

Procediamo con l'installazione dell'LXC tramite il comando [\[28\]](#):

```
# apt-get install lxc
```

Una volta completata l'installazione procediamo con la creazione del container e gli assegniamo il nome "OpenLISPcp"

```
# lxc-create -t download -n OpenLISPcp
```

Con il comando `-t download` abbiamo la possibilità di visualizzare tutti i template disponibili per il container e di scegliere la versione desiderata del sistema operativo scegliendo attraverso 3 diverse combinazioni: sistema operativo, release e architettura del processore. In questo caso è stato utilizzato Ubuntu trusty (14.04) con architettura a 64bit (amd64).

Una volta creato il container possiamo avviarlo utilizzando il comando:

```
# lxc-start -n OpenLISPcp
```

Per stoppare il container bisogna digitare il comando:

```
# lxc-stop -n OpenLISPcp
```

Per visualizzare i container installati e il loro stato possiamo utilizzare il comando

```
# lxc-ls -f
```

Per comodità è stato assegnato un nome statico all'interfaccia "eth" dell'LXC poiché il sistema prevede una generazione casuale dello stesso, il nome assegnato è *lxc0* [\[34\]](#).

Questa operazione va eseguita quando il container è stoppato. Per fare ciò bisogna editare il file "config" presente nella directory principale del container [\[33\]](#):

```
# cd /var/lib/lxc/OpenLispControllerLXC/
```

```
# nano config
```

Aggiungere la voce e salvare:

```
# lxc.network.veth.pair = lxc0
```

Una volta avviato il container possiamo "attaccarci" ad esso utilizzando il comando

```
# lxc-attach -n OpenLISPcp
```

Per convenzione è stato creato un nuovo utente "admin" ed impostata la relativa password:

```
# adduser admin
```

Una volta dentro al container installiamo le varie dipendenze necessarie per poi procedere all'installazione dell'OpenLISP control-plane.

```
# apt-get install build-essential
```

```
# apt-get install libexpat1-dev
```

```
# apt-get install wget
```

```
# apt-get install unzip
```

```
# apt-get install nano
```

8.3 Installazione OpenLISP Control Plane

Avviare il container e procedere con l'installazione:

```
# cd /home/admin  
# mkdir Download  
# cd Download
```

Scarichiamo la directory del control-plane nella cartella Download e una volta scompattata per convenzione la spostiamo fuori dalla cartella Download [29].

```
# wget https://github.com/lip6-lisp/control-plane/archive/master.zip  
# unzip master.zip  
# mv control-plane-master /home/admin  
# cd /control-plane-master
```

Procediamo all'installazione del control plane:

```
# make -f Makefile_linux  
# make -f Makefile_linux install
```

Il comando `"make -f Makefile_linux install"` installa il service script e copia i file di configurazione principale (`opencp.conf`) e altri cinque file di configurazione specifici: `'opencp_xtr.xml, opencp_ms.xml, opencp_mr.xml, opencp_node.xml, opencp_rtr.xml'` nella directory `/etc/`. Si prega di modificare i file di configurazione per personalizzare prima di continuare. Di default: il file `opencp_mr-sample-configure-of-DDT-root.xml` contiene un esempio di configurazione per la funzione DDT nodo radice; il file `opencp_xtr-sample-configure-of-lisp-te.xml` contiene un esempio di configurazione della funzione di nodo di RTR; il file `opencp_xtr-sample-configure-multi-mapping-system.xml` contiene un esempio di configurazione per la funzione del nodo xTR.

9. Configurazione

9.1 Configurazione OpenLISP Control Plane

La configurazione principale (`opencp.conf`) consente di abilitare le funzioni desiderate del control-plane. Essa indica anche i file di configurazione XML specifici per ogni funzione.

Il file di configurazione dell'xTR (opencp_xtr.xml) comprende:

- Sezione: elenco dei MS che l'xTR registra. Ogni MS ha bisogno di una chiave di autenticazione.
- Sezione: lista dei MR del xTR al quale può inviare le map-request.
- Una o più sezioni: ogni sezione fornisce le informazioni di un prefisso EID IP per la registrazione.

Il file di configurazione del Map-server (opencp_ms.xml) comprende:

- Sezione: prefissi IP del map-server utilizzati dall'ETR. Gli intervalli di indirizzi IP non devono sovrapporsi.
- Ogni sezione comprende le informazioni relative a un sito:
 - Nome del sito.
 - Chiave per i messaggi map-register. NB: la chiave è case sensitive e non deve includere spazi.
 - Prefissi EID IP a cui il sito può registrarsi.
- Una o più sezioni: ogni sezione include le informazioni per un RTR utilizzato per LISP-NAT.

Il file di configurazione del DDT node e MR (opencp_mr.xml) include:

- Sezione: il prefisso IP a cui il nodo è delegato. Gli intervalli di indirizzi IP non devono sovrapporsi. NB: se il nodo è un DDT root, allora è configurato come 0.0.0.0/0 (IPv4) e 0 :: / 0 (IPv6).
- Una o più sezioni: sezioni speciali con prefisso pari 0.0.0.0/0 o 0 :: / 0 sono per nodi DDT principali.

Il file di configurazione del RTR (pencp_rtr.xml) comprende:

- Sezione: lista dei MR a cui l'RTR può inviare le map-request.
- Una o più sezioni: ogni sezione include le informazioni per passaggio EID-prefix su RTR.

Per avviare il programma per la prima volta bisogna utilizzare il comando:

```
# service openvswitch start
```

Le opzioni disponibili sono:

- *Start*
- *Stop*
- *Restart*
- *Reload*
- *Condrestart*
- *Status*

9.2 Configurazione Open vSwitch

Per la configurazione dello switch OVS 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 OVSDDB. Sarebbe anche possibile, senza alcuna complessità aggiunta, eseguire tale configurazione da remoto, tramite un controller connesso al processo OVSDDB-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 `br0`.

```
# ovs-vsctl --may-exist add-br br0 -- set bridge br0 datapath_type=netdev
```

Assegniamo l'indirizzo della macchina al bridge

```
# ifconfig br0 192.168.1.102/24 up
```

Creo una porta virtuale per l'host (`vport_host`) e gli assegno un indirizzo, in questo caso l'11.0.0.1/24.

```
# ip tuntap add mode tap vport_host
```

```
# ifconfig vport_host 11.0.0.1/24 up
```

Aggiungo al bridge una porta OF vport_host (in questo caso la numero 1) alla quale è associata l'interfaccia vport_host.

```
# ovs-vsctl --may-exist add-port br0 vport_host
```

```
# ovs-vsctl set interface vport_host ofport_request=1
```

Analogamente viene creata una porta OF (indicata come #2) denominata lisp0 associata alla relativa interfaccia [\[30\]](#); si tratta di una porta logica di tipo "lisp", che provvede alle operazioni di incapsulamento dei pacchetti in uscita e decapsulamento di quelli in ingresso. L'opzione "flow", utilizzata per definire l'end-point del tunnel LISP (remote ip) e la chiave identificativa del tunnel stesso (key) [\[31\]](#).

```
# ovs-vsctl --may-exist add-port br0 lisp0
```

```
# ovs-vsctl set interface lisp0 ofport_request=2 type=lisp  
> options:remote_ip=flow options:key=flow
```

Successivamente viene creata una porta OF (indicata come #3) denominata lxc0 associata alla relativa interfaccia la quale provvede al collegamento tra lo switch e l'LXC.

```
# ovs-vsctl --may-exist add-port br0 lxc0
```

```
# ovs-vsctl set interface lxc0 ofport_request=3
```

.

9.3 Configurazione Host Virtual Machine

Viene configurata l'eth0 della Virtual Machine (enp0s3) ed inoltre viene aggiunto il default gateway per la sottorete 11.0.0.0/24.

```
# ifconfig enp0s3 11.0.0.2 netmask 255.255.255.0 up
```

```
# route add default gw 11.0.0.1
```

Una volta eseguiti tutti i comandi sopra riportati avremmo un sistema configurato come nella figura seguente cioè con una virtual machine che funge come host alla quale è assegnato l'indirizzo .2 della sottorete 11.0.0.0/24, uno switch installato sulla nostra macchina configurato con tre diverse interfacce: una per la comunicazione con l'host, una per la comunicazione con il container ed una per la comunicazione con l'RLOC space; infine un Container sul quale è installato il controller LISP: OpenLISP cp [32].

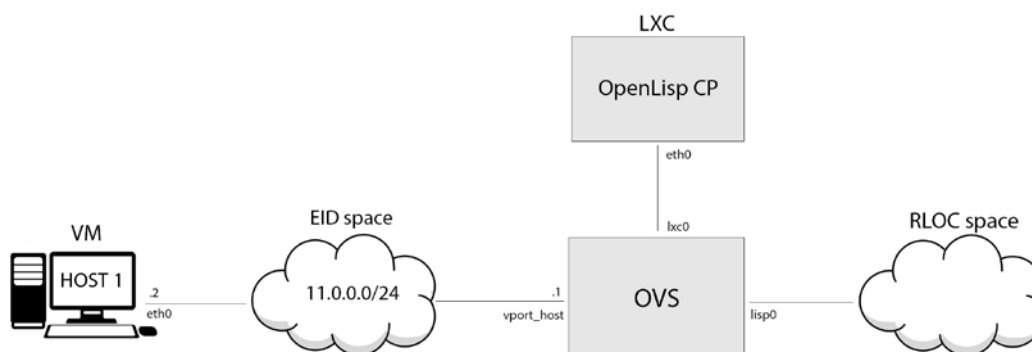


Figura 16 - Configurazione corrente

9.4 Configurazione flow-table Open vSwitch

Dopo aver settato l'infrastruttura provvediamo a settare le regole di flusso nel nostro bridge in modo da poter gestire i seguenti casi:

1. I pacchetti dati LISP possono arrivare dalla rete EID (EID space) e in questo caso settiamo l'interfaccia vport_host (OF port #1) la quale deve inviare i pacchetti verso l'interfaccia lisp0 che si occupa di incapsularli e fare forwarding.
2. I pacchetti dati LISP possono arrivare dalla rete LISP (RLOC space) e in questo caso settiamo l'interfaccia lisp0 (OF port #2) la quale deve decapsularli e fare forwarding sull'interfaccia vport_host.
3. I pacchetti di controllo LISP (contrassegnati dalla porta di destinazione numero 4342) arrivano al router (indipendentemente dall'interfaccia) e Ovs li invia al LISP Control Plane tramite l'interfaccia lxc0 (OF port #3).

Per quanto riguarda il caso 1 provvediamo a fare incapsulamento, su tutti pacchetti IP (EtherType 0x0800) in ingresso da vport_host destinati a HOST2 viene settato l'RLOC di destinazione e forwardati sulla porta lisp0. La regola inserita nel bridge è:

```
# ovs-ofctl add-flow br0 \  
>'priority=1,in_port=1,dl_type=0x0800,nw_dst=12.0.0.2,\  
action=set_field:153.100.122.1->tun_dst,output:2'
```

Per quanto riguarda il caso 2 provvediamo a fare decapsulamento, su ogni pacchetto in ingresso in lisp0 viene modificato il mac_address con quello dell'host EID di destinazione e inoltrato verso vport_host. La regola inserita nel bridge è:

```
# ovs-ofctl add-flow br0 \  
>priority=3,in_port=2,actions=mod_dl_dst:<macHostDest>,output:1
```

Per quanto riguarda il caso 3 provvediamo ad inoltrare i pacchetti di controllo LISP che arrivano dalla porta 2 (lisp0) filtrati tramite numero di porta (4342) e protocollo di rete UDP (nw_proto= 17) sulla porta 3 dell'lxc (lxc0) verso il LISP Control Plane:

```
# ovs-ofctl add-flow br0 \  
>priority=4,in_port=2,dl_type=0x0800,nw_proto=17,tp_dst=4342,\  
action=mod_dl_dst:<macLXC>,output:3
```

E' stata inoltre aggiunta una regola che disciplina il forwarding del protocollo ARP (EtherType 0x0806) senza alcuna modifica dei pacchetti i quali vengono inoltrati verso l'eth0 (nel nostro caso wlp3s0):

```
# ovs-ofctl add-flow br0 priority=2,in_port=1,dl_type=0x0806,actions=NORMAL
```

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

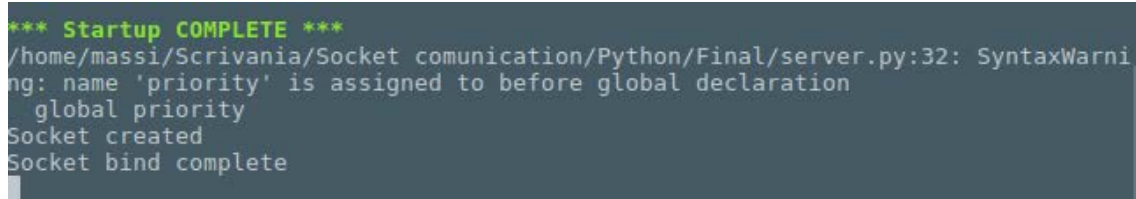
```
# ovs-ofctl add-flow br0 'priority=0, actions=NORMAL'
```

10. Funzionamento

Provvediamo ora a lanciare il server sulla macchina digitando dalla directory dove è contenuto il file il comando:

```
# python server.py
```

Una volta lanciato il comando verrà visualizzata questa schermata:

A terminal window showing the output of running a Python script. The text is as follows:

```
*** Startup COMPLETE ***  
/home/massi/Scrivania/Socket comunicazione/Python/Final/server.py:32: SyntaxWarning: name 'priority' is assigned to before global declaration  
    global priority  
Socket created  
Socket bind complete
```

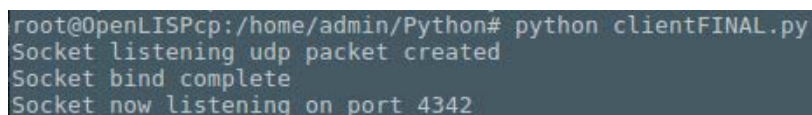
Figura 17 - Avvio del Server

Nella schermata quindi verrà visualizzato il messaggio che la socket in grado di ascoltare i messaggi del client è stata creata e successivamente tramite il comando bind attacchiamo la socket ad uno specifico indirizzo e ad una specifica porta.

Analogamente lanciamo il client sul Linux Container digitando dalla directory dove è contenuto il file il comando:

```
# python client.py
```

Una volta lanciato il comando verrà visualizzata questa schermata:

A terminal window showing the output of running a Python script. The text is as follows:

```
root@OpenLISPcp:/home/admin/Python# python clientFINAL.py  
Socket listening udp packet created  
Socket bind complete  
Socket now listening on port 4342
```

Figura 18 - Avvio del Client

Il client prevede la creazione di due socket, una in grado di ascoltare la porta 4342 dell'interfaccia *eth0* del container la quale nel momento riceve un pacchetto su quella interfaccia provvede alla creazione di una nuova socket utilizzata per la comunicazione con il server presente sul PC.

Una volta che un pacchetto UDP con porta di destinazione 4342 arriva al container la socket lo riconosce ed avvia il programma il quale provvede a leggere un file XML nel quale sono contenute le associazioni EID-RLOC, una volta raccolte queste informazioni il client le invia al server il quale a sua volta risponderà al client con un messaggio di successo.

```
root@OpenLISPcp:/home/admin/Python# python clientFINAL.py
Socket listening udp packet created
Socket bind complete
Socket now listening on port 4342

INIZIALIZZO

***** OPERAZIONE numero: 1 *****

Map-server1: 193.162.145.50, Map-server2: 195.50.116.18
Map-resolve: 193.162.145.50
eid:153.16.44.112/28, rloc:151.100.8.174
eid:11.0.0.2/26, rloc:151.100.8.176
Server reply : fatto!.... 153.16.44.112/28,151.100.8.174-11.0.0.2/26,151.100.8.176-
```

Figura 19 - Client Communication

Lato server, una volta ricevute le associazioni dal client, il server provvede a splittare questi indirizzi e ad aggiungerli tramite regole di flusso OVS nel nostro switch inoltre rimane sempre in ascolto per eventuali aggiornamenti:

```
*** Startup COMPLETE ***
/home/massi/Scrivania/Socket communication/Python/Final/server.py:32: SyntaxWarning: name 'priority' is assigned to before global declaration
  global priority
Socket created
Socket bind complete
EID 1: 153.16.44.112/28
RLOC 1: 151.100.8.174
EID 2: 11.0.0.2/26
RLOC 2: 151.100.8.176

Inserite flow-rules per EID: 153.16.44.112/28 RLOC: 151.100.8.174 nel bridge: br0
Inserite flow-rules per EID: 11.0.0.2/26 RLOC: 151.100.8.176 nel bridge: br0
Message[10.0.3.127:53021] - 153.16.44.112/28,151.100.8.174-11.0.0.2/26,151.100.8.176-
█
```

Figura 20 - Server Communication

In aggiunta, lato client, queste scansioni vengono scandite a periodi di tempo regolari e nel momento in cui le associazioni sono identiche a quelle precedentemente fatte non vengono inviate al client:

```
**** OPERAZIONE numero: 2 ****  
Map-server1: 193.162.145.50, Map-server2: 195.50.116.18  
Map-resolve: 193.162.145.50  
eid:153.16.44.112/28, rloc:151.100.8.174  
eid:11.0.0.2/26, rloc:151.100.8.176  
  
NON INVIO NULLA
```

Figura 21 - Esempio funzionamento Client

11. Conclusioni

La rete non sta scalando come dovrebbe, la crescita esponenziale degli indirizzi necessari e la gestione di essi rappresenta un forte problema. L'obiettivo del protocollo di rete LISP è quello di risolvere questo problema dividendo i normali indirizzi IP in due nuovi indirizzi, EID ed RLOC. Oltre a questo innovativo protocollo stanno prendendo piede da diverso tempo a questa parte le Network Function Virtualization le quali non prevedono più un hardware dedicato per ogni funzionalità di rete ma bensì un hardware condiviso virtualizzato in grado di gestire più funzionalità di rete, nel nostro caso la funzionalità di rete analizzata è appunto la Locator Identifier Separator Protocol (LISP).

La soluzione sarebbe quindi quella di adoperare questo protocollo su dispositivi di virtualizzazione leggera collegati al nostro switch. La possibilità di installare ed utilizzare diverse funzionalità di rete senza rivolgerci direttamente a delle infrastrutture di virtualizzazione delle funzionalità di rete, riferendoci a semplici spazi di virtualizzazione sparsi nella rete, diminuirebbe il carico gestito da queste infrastrutture, con una eventuale possibilità di diminuzione delle tempistiche (come nel caso del mapping system per il protocollo LISP) delle associazioni EID-RLOC.

Oltre a questo però andrebbe sviluppato un apposito meccanismo di comunicazione che, nel caso di questo lavoro di tesi, è stato implementato mediante due programmi (client/server) che comunicano tra loro mediante il meccanismo delle socket in modo da aggiornare le regole di flusso dello switch dinamicamente utilizzando gli EID/RLOC ottenuti dal mapping.

Il lavoro svolto mostra come la soluzione proposta miri ad alleggerire il carico di queste infrastrutture di virtualizzazione delle funzionalità di rete rendendole topologicamente sparse in quasi tutta la rete.

References

- [1] La virtualizzazione di rete: lo standard NFV - <http://www.telecomitalia.com/tit/it/notiziariotecnico/edizioni-2015/2015-2/capitolo-7.html>
- [2] Network Functions Virtualization (NFV) - <http://searchsdn.techtarget.com/definition/network-functions-virtualization-NFV>
- [3] Network Functions Virtualization (NFV); Architectural Framework - http://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf
- [4] The Locator Identifier Separation Protocol (LISP) - <http://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-39/111-lisp.html>
- [5] LISP Overview - http://lisp.cisco.com/lisp_over.html
- [6] Iannone L., Saucez D., Bonaventure O. “Implementing the locator/id separation protocol design and experience” – Computer Networks, January 14, 2011 – URL: <https://inl.info.ucl.ac.be/publications/implementing-locatorid-separation-protocol-design-and-experience>
- [7] LISP Device Recommended Filtering Guidelines – <http://www.lisp4.net/documentation/lisp-filtering-guidelines/>
- [8] Introduzione a SDN - https://www.citrix.com/content/dam/citrix/en_us/documents/oth/sdn-101-an-introduction-to-software-defined-networking-it.pdf
- [9] Open Network Foundation. Software Defined Network - <https://www.opennetworking.org/sdn-resources/sdn-definition>
- [10] Kreutz, Diego, Paulo Esteves Verissimo and Siamak Azodolmolky - “Software-Defined Networking: A Comprehensive Survey” – *Proceedings of the IEEE 103.1 (2015)*.
- [11] OpenFlow - <http://flowgrammable.org/sdn/openflow/>
- [12] OpenFlow Switch Specification - <http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf>
- [13] SDN e NFV quali sinergie? - <http://www.telecomitalia.com/tit/it/notiziariotecnico/numeri/2014-2/capitolo-05.html>
- [14] Cianfrani A, Samii M., Lo Bascio D., Polverini M. 2016. “Implementing a Smart SDN Switch with LISP Control Plane as Network Function” - *5th IEEE International Conference on Cloud Networking (CloudNet 2016)* – URL: <http://netlab.uniroma1.it/netgroup/sites/default/files/Main.pdf>

- [15] Getting Started OpenFlow OpenvSwitch Tutorial Lab - <http://networkstatic.net/openflow-openvswitch-lab/>
- [16] Funzionamento Linux Container - <http://www.cloudtalk.it/container-cosa-sono-come-funzionano/>
- [17] LXC vs VM - <https://neeners.neen.it/metodi-di-virtualizzazione-a-confronto-container-vs-virtual-machine/>
- [18] LXC-getting started - <https://linuxcontainers.org/it/lxc/getting-started/>
- [19] L. Iannone, D. Saucez, O. Bonaventure – “OpenLISP Implementation Report” – UCLouvain, Belgium – July 16, 2008 – URL: <https://tools.ietf.org/pdf/draft-iannone-openlisp-implementation-01.pdf>
- [20] Dung Phung Chi, Stefano Secci, Guy Pujolle, Patrick Raad, Pascal Gallard – “An Open Control-plane implementation for LISP networks” – URL: https://www.researchgate.net/publication/260320476_An_open_control-plane_implementation_for_LISP_networks
- [21] Dung Chi Phung, UPMC and VNU Stefano Secci, UPMC Damien Saucez, INRIA Sophia Antipolis Luigi Iannone, telecom ParisTech – “The OpenLISP Control Plane Architecture” – IEEE Network March/April 2014 - URL: https://www.researchgate.net/publication/261601160_The_OpenLISP_control_plane_architecture
- [22] Linux Socket - <http://www.tenouk.com/Module42a.html>
- [23] Socket cosa sono e come funzionano - <https://fortyzone.it/socket-cosa-sono/>
- [24] UDP socket communication - <https://wiki.python.org/moin/UdpCommunication>
- [25] Programming UDP socket in python - <http://www.binarytides.com/programming-udp-sockets-in-python/>
- [26] OVS command reference - <http://www.pica8.com/document/v2.6/html/ovs-commands-reference/>
- [27] Install Openflow network - <http://dannykim.me/danny/openflow/57620?ckattempt=2>
- [28] Setup LXC - <http://www.itzgeek.com/how-tos/linux/ubuntu-how-tos/setup-linux-container-with-lxc-on-ubuntu-16-04-14-04.html>
- [29] OpenLISP control plane - <https://github.com/lip6-lisp/control-plane>
- [30] OVS LISP tunnel configuration - <http://www.bluezd.info/archives/710>
- [31] Using LISP tunneling - <http://docs.openvswitch.org/en/latest/howto/lisp/>
- [32] OVS-LXC - <https://github.com/ebiken/doc-network/wiki/HowTo:-OVS-and-LXC>
- [33] Automally connect LXC to OVS - <http://blog.scottlowe.org/2014/01/23/automatically-connecting-lxc-to-open-vswitch/>

[34] **Attach LXC to OVS** - <https://infologs.wordpress.com/2015/06/19/how-to-attach-lxc-container-to-ovs-openvswitch/>

[35] James F. Kurose, Keith W. Ross – **RETI DI CALCOLATORI E INTERNET – Un approccio top-down 6° edizione PEARSON** - da pag.148 a pag.154

[36] David Lo Bascio – **“Implementazione Open vSwitch delle funzionalità di forwarding del protocollo LISP in uno scenario Software Defined Network” – 2013/2014 – URL:**
<http://www.extraordy.com/wp-content/uploads/2015/10/Tesi-di-Laurea-Magistrale-David-Lo-Bascio.pdf>