UNIVERSITÀ DI PISA
FACOLTÀ DI INGEGNERIA

Corso di Laurea Specialistica in Ingegneria Informatica
Curriculum Networking e Multimedia

Sviluppo e validazione di un protocollo di scambio delle label con supporto al fast reroute in reti wireless-mesh basate su MPLS

RELATORI
Prof. Luciano Lenzini
Prof. Enzo Mingozzi
Ing. Claudio Cicconetti

CANDIDATO
Francesco Giurlanda

ANNO ACCADEMICO 2008-2009

A tutti coloro che hanno creduto in me.

Sommario

In questa tesi verrà descritta la realizzazione di un protocollo di gestione delle label conforme al RFC 5036 (Label Distribution Protocol, LDP) operante su un prototipo di rete wireless-mesh con tecnologia 802.11 e basata su un meccanismo di label switching conforme al RFC 3031 (Multiprotocol Label Switching, MPLS), sviluppato come estensione del software Click Modular Router. In seguito verrà implementata una tecnica di protezione dal malfunzionamento di un nodo o link del Label Swithed Path, basata sull'utilizzo del Fast Reroute di MPLS su Detour preventivamente installati.

Il testbed è costituito da cinque router equipaggiati con due interfacce wireless 802.11 a/b/g e due interfacce Ethernet e da alcuni PC operanti da client che effettuano richieste di installazione e cancellazione di LSP e generano del traffico su essi.

I dispositivi sono configurati in modo da realizzare una rete wireless-mesh multihop, in cui i pacchetti sono istradati secondo le rotte stabilite dall'algoritmo di routing Srcr, sviluppato in Click Modular Router, o secondo il motore di forwarding MPLS, lungo LSP preinstallati.
Sono stati quindi effettuati una serie di test per validare le soluzioni proposte.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Motivazioni

Negli ultimi anni si è diffusa la tecnologia wireless per la realizzazione di reti di telecomunicazione, come valida alternativa alle tradizionali reti cablate, grazie al costo contenuto delle apparecchiature, alla possibilità di realizzare una rete informatica in contesti che non consentono una cablatura e alla semplicità di utilizzo. Se questi aspetti sono stati il motivo di una celere diffusione in ambito privato, non si può dire lo stesso per l'impiego su larga scala, come la copertura di ampie zone metropolitane con cui fornire accesso wireless ad Internet. Nonostante il costo limitato dei dispositivi, la loro diffusione su larga scala è limitata dalla sua struttura gerarchica e dall'utilizzo della rete wireless come semplice mezzo di accesso ad una backbone di tipo cablato: ciò comporta sia la difficoltà di schieramento sul territorio dei dispositivi di accesso (AP), in punti che permettano il collegamento alla backbone, che la scarsa tolleranza ai guasti, sintomatica di un sistema centralizzato che identifica nel collegamento tra AP e backbone un punto debole del sistema. Mobile Ad-hoc Network (MANET) è un sistema autonomo di nodi mobili connessi mediante collegamenti wireless e realizza una rete senza fili di tipo cooperativa costituita da un elevato numero di nodi interconnessi che fungono da ricevitori, trasmettitori e router, in cui tutti i nodi collaborano con lo scopo di instradare i pacchetti in modo efficiente. Questo tipo di infrastruttura è decentralizzato, relativamente economico e molto adattabile e resistente ai guasti. Se un nodo viene meno alla rete, è sufficiente cercare un altro nodo che potrà instradare l'informazione. Nonostante gli evidenti vantaggi, anche questo approccio

non ha avuto largo impiego a livello commerciale, trovando un terreno fertile invece in impieghi militari[1] e di protezione civile. In questo contesto, le Wireless Mesh Networks (WMN), pur mantenendo le capacità di auto-configurazione e di autoriparazione ereditate dalle MANET, si propongono sia come soluzione autosufficiente che come estensione di una infrastruttura di rete già esistente. Una tale versatilità di impiego delle WMN richiede dei meccanismi di controllo più accurati e complessi per assicurare delle buone prestazioni: protocolli di routing dedicati, sistemi di valutazione della bontà del link, tecniche di recupero da condizioni di guasto. Attualmente la complessità di queste operazioni si traduce in elevati tempi di reazione. Considerando che uno scenario wireless è soggetto a cambiamenti ed interruzioni più frequenti di uno scenario cablato, le WMN, senza opportuni accorgimenti, si presentano inadatte alla gestione di traffico che richiede una certa continuità del servizio.

In questo lavoro, proponiamo di trasportare i vantaggi della tecnologia label switched [2], ben noti in un contesto cablato, in un contesto di rete WMN, affiancandolo al routing tradizionale. Caratteristiche come velocità, scalabilità ma principalmente aspetti di semplicità di gestione, consumo delle risorse e controllo delle rotte sono tipiche del Label Switching; sfruttando queste caratteristiche miriamo a ridurre i tempi di reazione della WMN.

Un particolare aspetto che incentiva l'uso del Label Switching in WMN è rappresentato dall'esistenza di tecniche molto efficienti di Fast Reroute in grado di sopperire ai guasti o malfunzionamenti di un link usando delle rotte alternative prestabilite e garantendo la continuità del traffico. Questo comportamento, combinato con un sistema di link management, rappresenterebbe un ottima soluzione per reagire tempestivamente ai cambiamenti improvvisi nella topologia di una rete WMN.

Il Label Switching quindi permette un controllo migliore sui router che compongono una WMN; ma per fare ciò deve essere affiancato ad un sistema di gestione delle label che si adatti alle esigenza della WMN, cioè che sia in grado di introdurre il minor overhead possibile nella rete dovuto ai messaggi di sincronizzazione e che si accorga dei frequenti cambiamenti topologici della WMN.

Sfruttando le tecniche di Fast Reroute precedentemente annunciate, il label switching, in caso di guasto di un nodo o interruzione di un link, potrebbe offrire un valido supporto deviando il traffico su una rotta di backup, dando al protocollo

[1] JTRS, Joint Tactical Radio System

di routing il tempo necessario per accorgersi del guasto e calcolare una nuova rotta, il tutto senza interrompere il traffico di dati. Inoltre, una soluzione con Fast Reroute, oltre al vantaggio di essere più reattiva del routing tradizionale è del tutto trasparente al traffico end-to-end.

1.2 Obiettivi

Questa ricerca si focalizza principalmente sulla realizzazione di un sistema di distribuzione e installazione delle label per la creazione di LSP e contemporaneamente l'installazione dinamica di percorsi di backup (detour) che mirino al recupero più efficiente da condizioni di guasto di nodi o interruzione di link all'interno della rete, il tutto in un contesto di rete wireless mesh.

Partendo da un motore di forwarding MPLS realizzato in Click Modular Router [2], gli obiettivi di questa tesi sono due:

1. realizzazione di un protocollo di scambio delle label, conforme allo standard RFC 5036[2], che permetta l'installazione e la rimozione di un LSP, seguendo i percorsi calcolati dal protocollo di routing;

2. implementazione di un meccanismo di fast reroute basato sull'utilizzo di detour per proteggere il LSP da possibili guasti dei nodi o interruzione di link;

Lo sviluppo e la validazione di queste funzionalità saranno quindi svolti in un ambiente reale.

1.3 Struttura della tesi

La tesi è strutturata come segue: nel capitolo 2 verranno presentate sommariamente le tecnologie impiegate nello sviluppo di questo lavoro, inizialmente verrà analizzata la tecnologia Wireless Mesh Network (2.1) con una panoramica generale sui protocolli di routing esistenti, seguita dalla descrizione di implementazioni reali; successivamente verranno esposti i concetti essenziali relativi al label switching in Multiprotocol Label Switching (MPLS) (2.2) e due tecniche connesse ad esso: una di ottimizzazione nota come Penultimate Hop Popping e una per il

[2]Label Distribution Protocol (LDP) Specification.

condizionamento del percorso seguito da un flusso dati chiamata Fast Reroute. Infine verrà descritto il protocollo LDP (Label Distribution Protocol) con le diverse modalità di funzionamento e la struttura dei messaggi da esso usati.

Nel capitolo 3 verrà descritto il testbed, delineando il tipo di hardware, il software e la configurazione, il protocollo di routing adottato e la metrica usata per stabilire la bontà di un link in fase di creazione delle rotte, in seguito verranno illustrati i vantaggi e svantaggi nella realizzazione di un ambiente reali per i test.

Nel capitolo 4 verrà descritto la fase di progettazione, quindi verranno illustrate e motivate le soluzioni scelte per il raggiungimento degli obiettivi.

Nel capitolo 5 si passa alla descrizione dell'implementazione delle scelte e delle operazioni delineate nel capitolo 4.

Infine, nel capitolo 6, verranno mostrati i risultati dei test di validazione e delle prestazioni condotti sul testbed.

Capitolo 2

Cenni sulle tecnologie adottate

In questo capitolo verrà fatta un'introduzione alle tecnologie su cui si basa questo lavoro. Si comincerà introducendo le Wireless Mesh Networks (2.1): analizzeremo quindi sia le differenze con il tradizionale paradigma di reti wireless con infrastruttura, le analogie con le reti wireless ad-hoc, i protocolli di routing esistenti (2.1.1) e verranno brevemente illustrati alcuni testbed realizzati da altre istituzioni universitarie (2.1.2).

In seguito, verrà trattato il meccanismo di label switching conforme con Multiprotocol Label Switching (2.2) con una panoramica sul funzionamento e sulle caratteristiche; infine si parlerà di due tecniche, utili ai fini di questo lavoro, il Penultimate Hop Popping (PHP) come tecnica di ottimizzazione e del Fast Reroute come tecnica di prevenzione da guasti sul path, esaminandone i vantaggi e svantaggi. Alla fine (2.3) verrà descritto il protocollo LDP (Label Distribution Protocol).

2.1 Wireless Mesh Networks

La Wireless Mesh Networks (WMN) è una rete di comunicazione a maglie implementata tramite un insieme di nodi interconnessi tra loro tramite link wireless, le cui caratteristiche principali sono la capacità di auto-configurazione ed autoriparazione. WMN sono composte da tre tipologie di nodi: mesh-clients, mesh-routers e gateway; le comunicazioni partono generalmente dai mesh-clients (es. laptop, telefoni cellulari o altri dispositivi wireless). I mesh-client, di solito, non si occupano attivamente del routing, si collegano a un mesh-router che si occuperà

di inoltrare, in un contesto hop-by-hop, le informazioni verso altri mesh-clients o verso il gateway connesso ad Internet.

Con una struttura a maglie, si nota facilmente che la rete mesh è affidabile ed offre ridondanza: quando un nodo non è più operativo, a causa di malfunzionamenti, il resto dei nodi può ancora comunicare tra di loro, direttamente o attraverso nodi intermedi.

La rete wireless mesh può essere implementata attraverso varie tecnologie wireless come 802.11, 802.16, tecnologie cellulari o ancora combinazioni di più di una tecnologia.

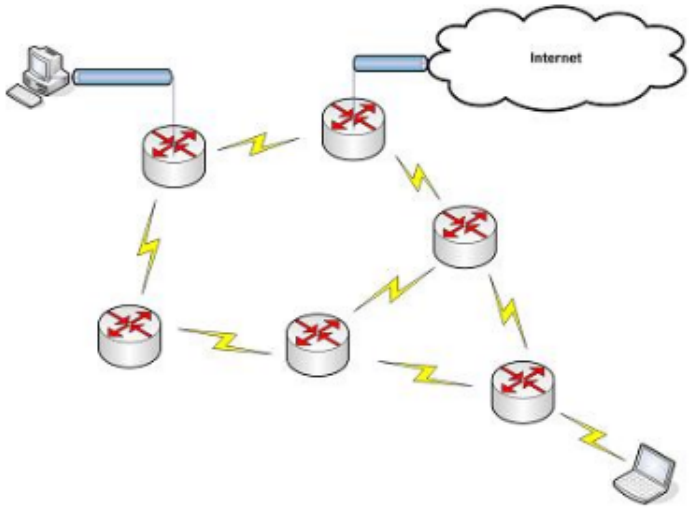


Figura 2.1: Wireless Mesh Network

La struttura della rete WMN è totalmente decentralizzata, diversamente da quanto accade in una Infrastructure-based Wireless Network (figura 2.2) che permette ai dispositivi wireless di comunicare tra loro finché mantengono la connessione con un Access Point e quindi costringendoli a rimanere all'interno di un'area limitata, conosciuta come Basic Service Set (BSS); l'AP a sua volta può essere isolato o connesso ad una infrastruttura di rete cablata con accesso a Internet. La copertura limitata, le difficoltà di un ottimale dispiegamento degli AP e la scarsa tolleranza ai guasti dell'approccio centralizzato sono i principali difetti dell'architettura con infrastruttura che in compenso non richiede un algoritmo di routing specifico dato che tutte le comunicazioni passano per l'AP.

Un approccio che si avvicina un po' alla struttura decentralizzata delle reti WMN è quello adottato dalle Ad-Hoc Wireless Networks (figura 2.3), ovvero reti

che utilizzano un Independent BSS; non si ha la presenza di nessun AP e la rete è il risultato di connessioni dinamicamente create tra un nodo e tutti gli altri nodi che rientrano nel suo radio range. In questa tipologia di rete l'obiettivo è stabilire connessioni punto-punto tra le stazioni della rete stessa. Un nuovo nodo prima si metterà in ascolto di beacon trasmessi da altri nodi che annunciano la rete ad-hoc, altrimenti creerà lui stesso una rete ad-hoc cominciando a trasmettere i messaggi di beacon. In una rete di questo tipo, senza una struttura centrale di coordinamento, un difetto è l'assegnazione statica degli indirizzi IP e se ci aggiungiamo che le comunicazioni sono limitate tra nodi direttamente connessi si evidenzia l'inadeguatezza dell'impiego con un numero elevato di nodi.

Le WMN si distinguono dalle reti ad-hoc soprattutto per la possibilità di operare sia indipendentemente che come estensione di una infrastruttura di rete esistente, possiamo infatti avere dei gateway tramite i quali interfacciarsi verso la rete fissa.

2.1.1 Protocolli di routing

La natura decentralizzata e di autorganizzazione delle WMN, così come i potenziali cambiamenti nella topologia della rete dovuti sia alla mobilità dei nodi che a loro possibili guasti, richiedono un sistema che permetta di scoprire dinamicamente la struttura della rete; l'idea di base è che un nuovo nodo debba annunciare la sua presenza quando si unisce ad una rete ed attendere risposta dai suoi vicini; i nodi apprendono le informazioni sui loro vicini e le propagano in modo da far convergere la conoscenza di ogni nodo ad una soluzione comune che porta alla consapevolezza della topologia della rete. Da questa idea di base discen-

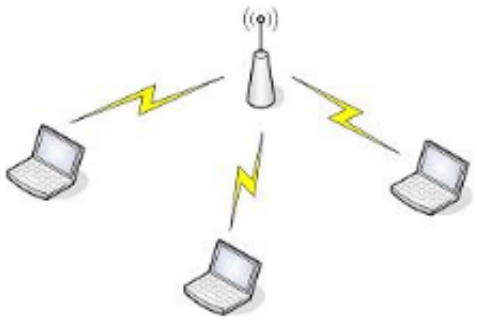


Figura 2.2: Infrastructure-based Wireless Network

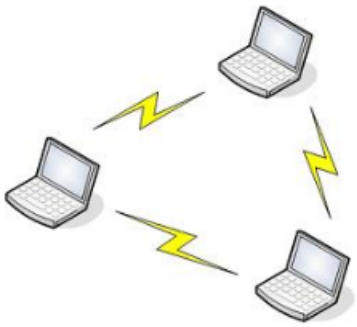


Figura 2.3: Ad-Hoc Wireless Networks

dono diverse specializzazioni di protocolli di routing, di seguito ne proporremo alcuni:

Pro-active routing (table-driven). In questo tipo di approccio i nodi apprendono la topologia della rete attraverso lo scambio periodico di pacchetti contenenti informazioni sulle destinazioni raggiungibili e sui cambiamenti della rete. In questa famiglia di protocolli ogni nodo deve mantenere una lista di tutte le destinazioni e dei relativi percorsi possibili nella rete, distribuendo periodicamente informazioni sulla propria routing table, anche quando non viene trasmesso nessun pacchetto dati. Quali informazioni vengono mantenute nella tabella e quali vengono scambiate tra i nodi dipende dallo specifico algoritmo. Esempi di questa tipologia di protocolli sono *Optimized Link State Routing* (OLSR), *Destination-Sequenced Distance Vector* (DSDV) a cui si ispira un altro protocollo, *Babel*, ottenendo tempi di convergenza più bassi, o ancora *Mobile Mesh Routing Protocol* (MMRP).

Reactive routing (on-demand). Diversamente dai protocolli pro-active, il percorso verso una certa destinazione è determinato su richiesta, ovvero solamente quando se ne ha effettivamente necessità e nel caso in cui non esista già un percorso verso la destinazione che si vuole raggiungere. Nel caso che un nuovo percorso debba essere stabilito il nodo che ne ha necessità invia un messaggio di Route Request in broadcast all'interno della rete. Quando questo messaggio raggiunge il nodo di destinazione o un nodo che ha conoscenza di un percorso verso la destinazione stessa viene inviato un messaggio di Route Replay verso il nodo che ha iniziato questo processo, utilizzando le informazioni salvate sui nodi intermedi al passaggio del messaggio di Route Request. Una volta stabilito il percorso, il nodo lo memorizza localmente,

per poterlo riusare in futuro in caso di necessità. Il traffico può quindi cominciare a fluire dalla sorgente verso la destinazione. Anche in questo caso c'è da fare una distinzione sul percorso di routing, che può essere specificato in forma sia di *source routing* (ogni pacchetto inviato contiene la lista dei nodi da attraversare) che *hop-by-hop* (ogni pacchetto contiene solo l'indirizzo del destinatario e del next hop). Nel primo caso i nodi intermedi non è necessario che mantengano informazioni sui percorsi attivi, in quanto il forwarding dei pacchetti è ottenuto dalle informazioni trasportate nel loro header. Nel caso di reti di grosse dimensioni questo approccio comporta un overhead non trascurabile per il loro trasporto. Nel secondo approccio ogni nodo intermedio mantiene informazioni sui percorsi attivi, grazie alle quali può effettuare il forwarding dei pacchetti. Esempi di questa tipologia di protocolli sono *Dynamic Source Routing* (DSR), *SrcRR* per quello che riguarda i protocolli basati sul source routing, mentre per l'altro approccio menzioniamo *Ad-hoc On-demando Distance Vector* (AODV). I principali svantaggi di tutti questi algoritmi sono l'elevato tempo di latenza nella ricerca di una rotta e un eccessivo flooding[1] che può portare ad un intasamento della rete.

Hybrid routing. Questa famiglia di protocollo combina i vantaggi dei protocolli proactive e reactive. Il routing è inizialmente stabilito con le tecniche di indagine tipiche dei protocolli proactive, in seguito l'aggiornamento delle rotte viene effettuato su richiesta con tecniche di flooding come accade tipicamente nei protocolli reactive. I principali svantaggia di questi algoritmi sono due: gli effetti benefici di questo approccio dipendono dall'ammontare dei nodi attivi e la reazione alla domanda di traffico dipende dal gradiente del volume del traffico.

Esempi di questo gruppo sono *ARPAM* specializzato per MANET ad impiego aeronautico, *OrderOne Routing Protocol* (OORP) e *Hybrid Routing Protocol for Large Scale Mobile Ad Hoc Networks with Mobile Backbones* (HRPLS).

I protocolli reactive non necessitano di inviare periodicamente informazioni sulle routing table di ogni nodo, comportando sia un minor overhead in termini di messaggi che circolano nella rete sia una quantità minore di informazioni che devono essere memorizzate su ogni nodo. Lo svantaggio di questo tipo di protocolli sta nel ritardo necessario per la creazione di un percorso verso una

[1] Termine usato per indicare l'azione di "inondare" la rete di Route Request.

nuova destinazione, dovuto alla procedura di setup iniziale. Il tipo di protocollo da utilizzare dipende quindi dalle dimensioni della rete che consideriamo: per reti con un numero limitato di nodi gli svantaggi dei protocolli proactive possono considerarsi trascurabili.

2.1.2 Alcuni esempi di WMN testbed

Un testbed è una piattaforma per la sperimentazione di progetti di ricerca; sono realizzati per fornire una verifica rigorosa, trasparente e replicabile delle teorie scientifiche. Tra i principali testbed che puntano alla ricerca sulle reti WMN abbiamo *Roofnet* sviluppata dal M.I.T. di Boston [3], *UCSB MeshNet* dell'Università di Santa Barbara [4] e *UMIC-Mesh* dell'Università di Aachen [5].

Roofnet

Roofnet è una rete realizzata nell'ambito dello sviluppo di nuovi protocolli per reti wireless mesh basati sulla ricerca di percorsi ad elevato throughput e sulla selezione del bit-rate trasmissivo in base alle condizioni del canale. La rete è unplanned, ovvero la disposizione dei nodi che la compongono non è studiata a priori, ma è semplicemente basata sulla disponibilità di volontari ad utilizzare l'hardware fornitogli, in cambio viene fornito accesso Internet. L'hardware è composto da uno small factor PC dotato di una interfaccia wireless 802.11b, di una antenna omnidirezionale per la comunicazione con gli altri nodi della rete, di una interfaccia Ethernet alla quale l'utente può connettersi ed è equipaggiato con una distribuzione Linux Red Hat sulla quale viene eseguito Click Modular Router. L'algoritmo di routing, realizzato in Click, è Srcr, una versione avanzata del protocollo SrcRR [6], ed utilizza la metrica Estimated Trasmission Time (ETT), basato sulla metrica Estimated Transmission Count (ETX), per cercare di stimare per quanto tempo il canale rimane occupato durante la trasmissione di un pacchetto: minimizzare questo tempo significa massimizzare il throughput ottenuto. L'algoritmo include inoltre un meccanismo per individuare il rate trasmissivo migliore su ogni link, attraverso il monitoraggio della packet lost su entrambe le direzioni di ogni link così come per eventuali link alternativi. In questo modo Srcr assicura l'utilizzo del percorso migliore; gli eventuali cambiamenti di percorso sono effettuati solamente nel caso di un sensibile miglioramento della metrica, questo per prevenire situazioni di indecisione difronte a rotte con metri-

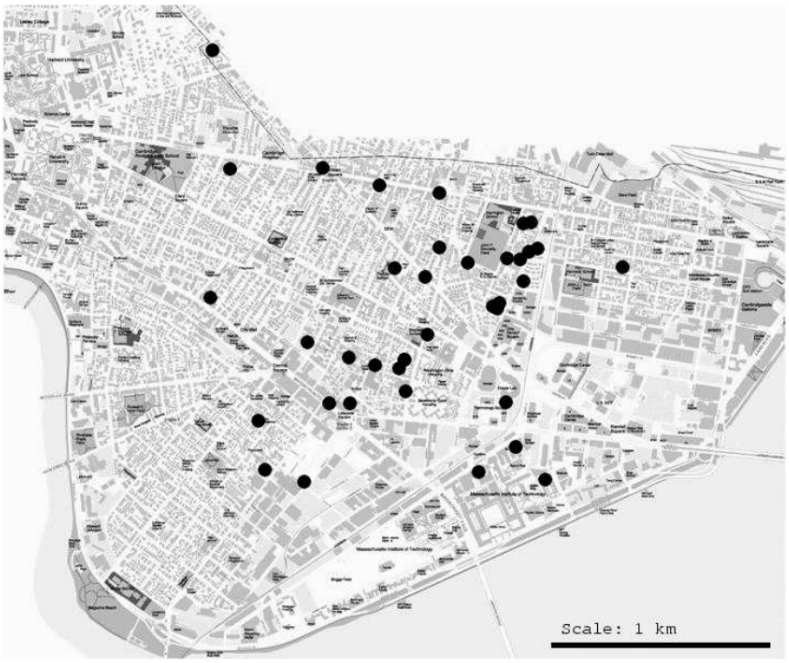


Figura 2.4: Mappa di Roofnet, ogni punto rappresenta un nodo della rete mesh.

che simili. Le misurazioni delle prestazioni della rete mostrano come, nonostante la sua natura unplanned, questa riesca ad ottenere un discreto throughput e come i nodi della rete siano ben serviti nonostante la presenza di pochi gateway al suo interno.

UCSB MeshNet

In questo lavoro è stata posta particolare attenzione sia alle problematiche legate alla costruzione di un testbed, partendo dalla scelta della piattaforma hardware e software fino alla sua realizzazione vera e propria, che sulla progettazione e sviluppo di un sistema di supporto per la gestione, monitoraggio e visualizzazione dello stato dei nodi della rete senza che questo abbia impatto sugli esperimenti svolti. UCSB MeshNet è composto da 30 nodi, disposti sui 5 piani di un edificio del campus universitario; i nodi si dividono in due gruppi, metà sono formati da due Linksys WRT54G connessi tra loro ed equipaggiati con una distribuzione OpenWRT; uno di questi è effettivamente il nodo che fa parte della rete mesh, l'altro, connesso ad una rete wireless già esistente, serve per la gestione del nodo mesh stesso. L'altra metà dei nodi sono dei piccoli PC dedicati, equipaggiati con una distribuzione Linux, più di una interfaccia IEEE

802.11a/b/g e una Ethernet. Entrambi i gruppi di nodi utilizzano Kernel-OADV e sono configurati con una interfaccia con cui connettersi ad una infrastruttura di rete preesistente che permette ai nodi di comunicare tra loro senza aggiungere traffico extra alla WMN. Questa infrastruttura viene utilizzata dai componenti di gestione e di monitoraggio del sistema.

UMIC Mesh

Partendo dall'analisi delle problematiche tipiche dell'implementazione di testbed di reti wireless mesh, legate soprattutto alla non scalabilità, in UMIC-Mesh viene proposta la creazione di un testbed ibrido composta da una serie di nodi reali affiancati ad un ambiente virtuale, sul quale viene sviluppato e testato il software che in seguito verrà trasferito ed eseguito sui nodi reali in modo da ottenere dei risultati con un alto grado di realismo. Questo approccio, del quale vengono analizzate nel dettaglio sia la creazione che i vantaggi rispetto ad altre metodologie di sviluppo di un testbed, viene proposto partendo dalla considerazione che eseguire software in un ambiente virtuale sia analogo alla sua esecuzione su router tradizionali ottenendo però la ripetibilità dei test. D'altra parte la natura ibrida del testbed sviluppato lo distingue dalle simulazioni vere e proprie, consentendo di riportare facilmente in un ambiente reale i risultati ottenuti, dato che le prestazioni della rete sono valutate solamente nella sua parte reale.

La configurazione è centralizzata, ottenuta con l'introduzione di un MeshServer che semplifica la distribuzione del software tra i nodi della rete e la creazione degli scenari.

Il testbed si compone di 48 single board PC muniti di due interfacce wireless 802.11a/b/g basate su chip Atheros AR5213 XR e due antenne omnidirezionali (mesh router): un'antenna è utilizzata per le comunicazioni router-to-router, l'altra per quelle router-to-client, su canali differenti ed in modalità ahdemo[2]. La parte virtuale del testbed consiste di 7 PC, ognuno dei quali capace di eseguire 10 virtual machines connesse tra loro da una virtual mesh emulation network; questa parte del testbed fornisce un ambiente flessibile per lo sviluppo di vari protocolli di rete. Tutti i router e le virtual machines sono poi connesse tramite una virtual backbone, utilizzata per la configurazione dei nodi. Il vantaggio più importante, ottenuto da questo ambiente ibrido è che il software sviluppato e

[2]La modalità ahdemo omette sia la trasmissione dei messaggi di beacon che il BSSID

testato sull'ambiente virtuale può essere trasferito sui nodi reali senza necessità di modifiche.

2.2 Label Switching

Il meccanismo di label switching che andremo a descrivere è conforme alle specifiche MPLS ed è quello adottato dal nostro testbed, implementato in un lavoro precedente [2].
MultiProtocol Label Switching (MPLS) [10] è un meccanismo di trasporto dati che opera ad un livello del modello OSI che generalmente si considera posizionato tra le tradizionali definizioni di livello 2 (Data Link layer) e livello 3 (Network layer), spesso viene considerato come protocollo di livello 2,5. Il termine Multiprotocol si riferisce alla possibilità di applicare questa tecnica a qualsiasi protocollo di livello network. Nel tradizionale routing IP, quando un pacchetto transita all'interno di una rete, ogni router attraversato processa il pacchetto e prende una decisione autonoma per quel che riguarda la ricerca del next-hop adatto per raggiungere una destinazione. In particolare ogni dispositivo analizza l'header di livello network del pacchetto, individua il next-hop basandosi sulle informazioni recuperate dall'algoritmo di routing e infine inoltra il pacchetto. L'obiettivo che MPLS si pone è quello di evitare di analizzare il network header del pacchetto su ogni router incontrato lungo il percorso, attraverso la realizzazione di un nuovo meccanismo per la scelta del next-hop.

Il processo di selezione del next-hop può essere considerato come la composizione di due funzioni: la prima partiziona l'insieme di possibili pacchetti in un insieme di FEC (Forwarding Equivalence Class), mentre, la seconda funzione, effettua l'associazione tra ognuno di questi FEC ed un next-hop. In questo modo possiamo avere pacchetti totalmente differenti mappati sullo stesso FEC non risultando più distinguibili per un router. Riferendoci al routing IP tradizionale, ogni router lungo il percorso del pacchetto dovrà eseguire entrambe le funzioni, e considererà due pacchetti appartenenti allo stesso FEC solo quando i loro destination address produrranno la selezione della stessa entry della routing table. In MPLS invece, l'assegnamento di un pacchetto ad un determinato FEC è fatto solamente una volta, esattamente da parte dell'ingress router[3], basandosi su determinate informazioni contenute nel pacchetto. Il concetto di FEC è più am-

[3]Router MPLS che gestisce il traffico in ingresso ad un dominio MPLS.

pio del concetto di destination address, ad esempio un pacchetto potrebbe essere associato ad una FEC in base alla coppia source-destination address o ancora più in generale, basandosi su politiche più complesse, potrebbe considerare anche le porte sorgente e destinazione o anche il tipo di protocollo usato a livello trasporto, senza che questo impatti sulle prestazioni dei successivi LSR[4]. Per far si che il FEC a cui il pacchetto è stato assegnato non debba essere rivalutato da parte di ogni LSR, questo viene codificato con una label, un campo di lunghezza fissa, trasmesso insieme al pacchetto stesso. Sui restanti LSR non sarà quindi più necessario procedere all'analisi dell'header del pacchetto per determinare il FEC, ma basterà utilizzare la label che il pacchetto trasporta con sé come indice in una ulteriore tabella che conterrà informazioni quali il next-hop per quel FEC ed un nuovo valore per la label; quest'ultima sostituirà la label precedentemente trasportato, prima che il pacchetto venga inviato al next-hop. Questo meccanismo, oltre ad alleggerire i nodi intermedi dal carico computazionale di valutare il FEC di appartenenza del pacchetto, fornisce un semplice modo per forzare un pacchetto su un determinato percorso senza la necessità di fare source routing[5]. Questo consente di ottenere sia un semplice supporto al traffic engineering, ovvero la differenziazione della banda di trasmissione in base al tipo di traffico, ma anche dei meccanismi per specificare dei percorsi di backup che proteggono da interruzioni del percorso principale, andando a variare semplicemente i valori di next-hop e label di uscita per un determinato FEC sul LSR che farà da bivio.

Il percorso che un pacchetto segue entrando nel dominio MPLS da un ingress router, attraversando una sequenza di LSR intermedi e uscendo da un egress router[6] è chiamato Label Switched Path (LSP).

Quando un pacchetto viene processato da un ingress router e supponendo che venga trovata una associazione con una FEC, viene inserito un shim header tra gli header di livello 2 e 3. Come riportato nelle specifiche di MPLS [10], questo header, costituito da 32 bits, è strutturato con i seguenti campi:

- *Label*, campo di 20 bits che contiene la label.
- *ToS*, 3 bits che specificano il tipo di servizio, utili per implementare mecca-

[4]Label Switch Router, router che supporta le funzionalità di label switching di MPLS.

[5]Il source routing una tecnica in cui un nodo, autore di un pacchetto, inserisce all'interno del pacchetto stesso il percorso che esso dovr effettuare.

[6]L'egress router è un router che gestisce traffico in uscita dal dominio MPLS.

nismi di Quality of Service (QoS).

- *Label Stack*, un bit che indica la presenza o meno di più livelli di shim header.
- *TTL*, Time to Live, 8 bits direttamente recuperati dal TTL del pacchetto al passaggio dall'ingress router: pone un limite su quanti nodi MPLS il pacchetto può transitare; questo campo è necessario perchè i LSR intermedi non esaminano il campo TTL dell'header IP.

In particolare il bit di label stack impostato ad 1 identifica l'ultimo shim header trasportato; è infatti possibile che un pacchetto trasporti un numero qualsiasi di shim header, organizzati come in una coda last-in first-out. Quindi il forwarding è sempre basato sul primo shim header dello stack e sapere il numero degli shim header presenti è superfluo, l'importante è riconoscere quando siamo in presenza dell'ultimo. L'utilità del label stack è evidente nel caso in cui si voglia realizzare un LSP tunnel: un router LSR1 che vuole inviare un pacchetto ad un altro router LSR2 attraverso un dominio MPLS differente, può ulteriormente incapsulare il pacchetto in un altro shim header che consenta al pacchetto di raggiungere LSR2; un'altra applicazione può essere anche quella di stabilire dei tunnel di backup a protezione di tratte particolarmente importanti. Il campo TTL mantiene invece la funzione di prevenzione dai loop, ogni ingress LSR copia il valore del campo TTL dal header IP o dallo shim header sottostante in un nuovo shim header, i LSR intermedi effettuano il decremento di questo valore che giunto a 0 causa l'eliminazione del pacchetto, l'egress LSR decrementa anch'esso il valore del TTL dello shim header e lo copia, dopo aver eliminato un livello dallo stack, nel header

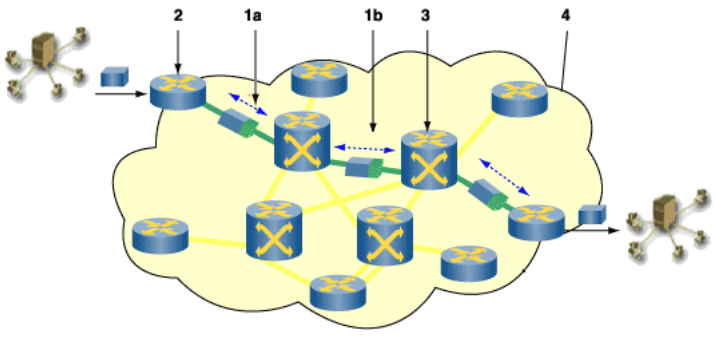


Figura 2.5: Struttura di una rete MPLS: (2) è un edge-LSR che in base al ruolo che svolge nei confronti di un LSP può essere ingress o egress LSR, (1a) e (1b) sono due tratti dello stesso LSP, (3) è un core-LSR , (4) rappresenta il dominio MPLS

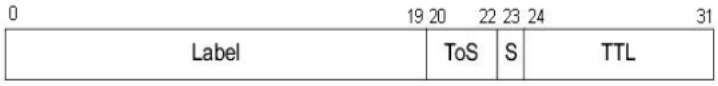


Figura 2.6: Struttura di un shim header.

IP o nel seguente shim header, in modo che il pacchetto emerga dal LSP con lo stesso valore del TTL che avrebbe avuto in caso di routing tradizionale. Un modo per gestire le informazioni necessarie per il forwarding consiste nel mantenere in ogni LSR in 3 tabelle:

FTN (FEC-to-NHLFE). Questa tabella associa ogni FEC ad una entrata della NHLFE che sarà usata per il forwarding del pacchetto. È utilizzata quando un ingress LSR riceve un pacchetto senza shim header e che devono quindi essere associati ad un FEC prima di poter proseguire.

ILM (Incoming Label Map). La ILM serve a gestire i pacchetti in ingresso ad un LSR, che trasportano uno shim header; questa tabella associa ad una label in ingresso una entrata della tabella NHLFE utilizzata per il forwarding.

NHLFE (Next-Hop Label Forwarding Entry). Questa tabella è utilizzata quando si tratta di dover effettuare il forwarding di un pacchetto per il quale è stata trovata una corrispondenza nella FTN o nella ILM; contiene le seguenti informazioni:

- Indirizzo del *next-hop*;
- *Label Out*
- *Action*: l'azione da effettuare sul pacchetto. Può essere una delle seguenti azioni:

  PUSH - Inserisce un nuovo shim header in testa al label stack se presente, oppure come primo shim dello stack se il pacchetto proveniva dalla FTN.

  REPLACE - rimpiazza la label presente nel primo shim header dello stack con il valore del campo Label Out.

  POP - Elimina il primo shim header dello stack.

È interessante notare che a partire da queste tre semplici azioni si possano realizzare operazioni più complesse combinandole in sequenza come l'azione di REPLACE/PUSH.

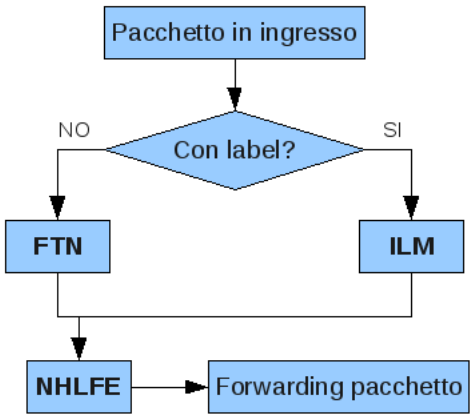


Figura 2.7: Schema del processo di forwarding MPLS.

Riassumendo il comportamento del forwarding MPLS, all'arrivo di un nuovo pacchetto il LSR esamina se contiene uno shim header: se è presente almeno un livello di shim header, consulta la tabella ILM cercando la label trasportata dal pacchetto; se non è presente nessun shim header, il LSR analizza il network header e consulta la tabella FTN cercando un FEC da associare al pacchetto. Se la ricerca in ILM e in FTN va a buon fine, viene associata al pacchetto un'azione identificata da un ingresso nella tabella NHLFE che specificherà come operare sulla label (inserirne una nuova, sostituirla o toglierla) ed il next-hop per il forwarding del pacchetto. In fine il pacchetto viene inoltrato.

2.2.1 Penultimate Hop Popping (PHP)

PHP è una ottimizzazione del tradizionale forwarding MPLS e consiste nel distribuire sull'ultimo e penultimo LSR di un LSP il carico computazionale che tradizionalmente gravava soltanto sull'ultimo LSR. Le operazioni svolte normalmente dall'ultimo LSR consistono nella scansione di due tabelle, operazione che può risultare onerosa in ambito di rete. La prima scansione riguarda la tabella ILM: il pacchetto dotato di shim header necessita di un riscontro in questa tabella per risalire all'azione da eseguire su esso. Dato che siamo nell'ultimo nodo del LSP l'azione sarà una POP; supponendo che non ci siano shim header annidati, dopo l'azione di POP ottengo un pacchetto con network header da processare e ciò significa l'interrogazione della tabella di routing.

Queste due operazioni possono essere separate, eseguendo la scansione del-

la ILM connessa all'azione di una POP sul penultimo nodo, in più si avrà un next-hop valido e diverso da se stesso nella stessa entrata della NHLFE; questo comporterà il forwarding MPLS di un pacchetto senza shim header verso l'egress-LSR. Alla fine l'ultimo LSR riceverà questo pacchetto e interrogherà la tabella di routing per stabilirne la destinazione.

2.2.2 Fast ReRoute

Fast ReRoute è una tecnica che provvede ad un veloce recupero del traffico a seguito di un guasto o interruzione sul percorso principale. Utilizzato solitamente per servizi mission-critical, questa tecnica è in grado di ridurre i tempi di gestione di guasto di un nodo o link all'ordine delle decine di millisecondi. Per mettere in pratica questi tempi, il meccanismo prevede di stabilire un LSP di backup alternativo all' LSP principale; il LSR che ha a disposizione il LSP e il backup-LSP manterrà entrambe le informazioni per l'inoltro del pacchetto nella propria tabella NHLFE; un'ulteriore struttura dati, chiamata FRR database terrà l'informazione che associa un LSP al suo backup-LSP, infine una terza entità sperimenterà il guasto del link o del next-hop e scatenerà l'evento che attiva l'utilizzo del backup-LSP. Quanto appena detto viene realizzato operando opportunamente sulla label e next-hop da associare ad un pacchetto in uscita dal LSR.

Il backup-LSP può essere realizzato staticamente dall'amministratore della rete che in questo caso specifica anche l'intero percorso hop-by-hop, oppure dinamicamente con delle procedure che in fase di richiesta d'installazione del LSP, trovano e installano i percorsi di backup; in questo caso però bisogna affidarsi alla bontà dell'algoritmo di valutare dei percorsi che mantengano eventuali specifiche sulla qualità del servizio del LSP principale e che inoltre non vadano ad interferire con il traffico di altri LSP già esistenti. In fig. 2.8 è illustrato un esempio di utilizzo di fast reroute, il LSP principale è costituito da R1, R2, R3, R9; è stato creato un backup-LSP a protezione del link tra R2 e R3 lungo il percorso R3, R6, R7 e R3. In condizioni normali R2 effettuerebbe l'operazione di REPLACE(37, 14)[7] ed inoltrerebbe verso R3. Supponiamo che il link R2-R3 si interrompa, entra in funzione il FRR che devia il traffico sul tunnel, si noti che R2 per fare ciò opera due azioni sul pacchetto: REPLACE(37, 14) per inserire una label valida per R3 e PUSH(17) che corrisponde all'aggiunta di un ulteriore livello di shim header valido per R6, infine inoltra il pacchetto ad R6. Il pacchetto segue il tunnel, come

[7]Nel gergo CISCO questa operazione viene chiamata SWAP.

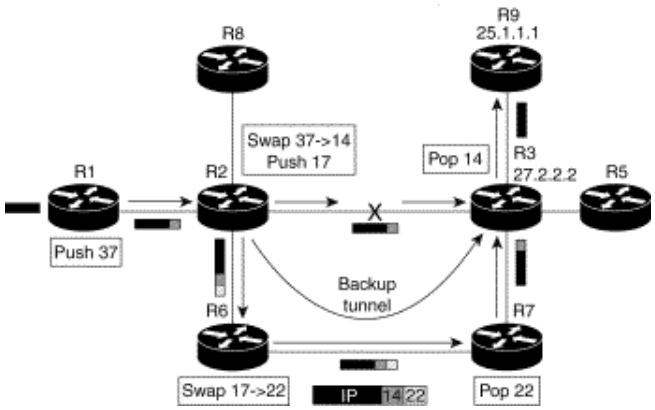


Figura 2.8: Esempio di Fast Reroute con un Backup Tunnel

farebbe in un LSP tradizionale, R7 viene effettuata l'operazione di POP(22) che sancisce la fine del tunnel, il pacchetto viene inoltrato verso R3 con la stessa label che avrebbe avuto se fosse stato inoltrato da R2 (si noti anche che in R7 è stata sfruttata anche la tecnica PHP). Tutto questo meccanismo rimane trasparente ad R3 che riceve comunque un pacchetto contrassegnato dalla label 14, indipendentemente dal percorso seguito. Da R3 in poi il pacchetto proseguirà la sua strada sul LSP originario.

2.3 Label Distribution Protocol (LDP)

L'architettura MPLS nello standard RFC 3031 definisce un protocollo di distribuzione delle label come un insieme di procedure secondo le quali un Label Switch Router (LSR) informa un altro LSR delle proprie label usate per il forwarding del traffico che lo attraversa. Partendo da questa definizione sono stati standardizzati diversi protocolli di distribuzione delle label per diversi casi di impiego. LDP, definito nello standard RFC 5036 [1], racchiude questo insieme di procedure e messaggi permettendo ad un LSR di stabilire un LSP. LDP associa un FEC ad ogni LSP creato, specificando quindi quali pacchetti devono essere mappati sul LSP.

Due LSR che usano LDP per scambiare informazioni sulle associazioni label/FEC sono identificati come *LDP Peers*; le informazioni vengono scambiate attraverso una *LDP Session* stabilita tra i due peers.

LDP divide i messaggi usati in quattro categorie:

1. messaggi di *discovery*, usati per annunciare e mantenere la presenza del LSR nella rete;
2. messaggi di *session*, usati per mantenere, stabilire e terminare una sessione tra LDP peers;
3. messaggi di *advertisement*, usati per creare, cambiare e cancellare corrispondenze tra label e FEC;
4. messaggi di *notification*, usati per fornire informazioni di avviso o segnalare condizioni di errore.

I messaggi di discovery forniscono al LSR un meccanismo per annunciare la propria presenza nella rete, periodicamente vengono inviati dei messaggi di *Hello* come pacchetti UDP al gruppo di multicast di tutti i router della rete e alla porta UDP 646. Quando un LSR decide di stabilire una sessione con un altro LSR di cui ha appreso l'esistenza grazie ai messaggi di Hello, esso avvia la procedura LDP di *inizializzazione sessione* usando TCP come protocollo di trasporto. Se la procedura termina con successo i due LSR diventano LDP peers e può cominciare lo scambio di messaggi. L'utilizzo di una sessione TCP per lo scambio di messaggi è dovuto alla necessità di stabilire una comunicazione affidabile che garantisca la consegna del messaggio tra due peers; mentre per le operazioni di discovery è sufficiente una comunicazione basata su UDP.

2.3.1 Spazio delle label

La nozione di "spazio delle label" è utile per poter discutere di assegnamento delle label. Si distinguono due tipi di spazi delle label:

Per interface label space. Il significato di una label di un pacchetto entrante nel LSR è legata all'interfaccia da cui è arrivato. Un esempio può essere un router con interfacce ATM che usano VCI come label, o una interfaccia Frame Relay che usa DLCI come label. Questo significa che nell'ambito del LSR si possono avere label con lo stesso valore ma con destinazioni e operazioni differenti, discriminate dall'interfaccia di arrivo del pacchetto.

Per platform label space. Tutte le interfacce condividono lo stesso spazio delle label, per cui il significato di una label è univoco e indipendente dalla interfaccia di arrivo.

2.3.2 Identificatori LDP

Un identificatore LDP è un valore su 6 byte usato per identificare un LSR e il suo spazio delle label. I primi 4 byte identificano il LSR e devono avere un valore univoco nella rete, come un router-ID a 32-bit assegnato al LSR. Gli ultimi due byte identificano uno specifico spazio delle label nel LSR. Nel caso di un assegnamento delle label per platform questo campo sarà sempre zero. Un identificatore LDP viene rappresentato con la seguente notazione: $\langle LSR\_ID \rangle$: $\langle label\_space\_id \rangle$ (es. lsr1:0, lsr3:2).

2.3.3 Hello Adjacency & Session

Il meccanismo di discovery consente ad un LSR di trovare dei potenziali LDP peers. Ci sono due varianti del meccanismo di discovery:

- il meccanismo base usato per rilevare LSR vicini, direttamente connessi a livello link;
- un meccanismo aggiuntivo usato per localizzare LSR che non sono direttamente connessi a livello link.

Nel meccanismo base di discovery, il LSR invia periodicamente da ogni interfaccia un *LDP Link Hello* in un pacchetto UDP alla porta well-known di LDP e al gruppo multicast di tutti i router. Il messaggio trasporta anche l'identificatore del LSR e dello spazio delle label per quella interfaccia, in più altre informazioni per la configurazione.

Ricevere un LDP Link Hello da una interfaccia identifica una *Hello Adjacency* (HA) con un potenziale LDP peer raggiungibile a livello link da quella interfaccia. Il meccanismo per localizzare LSR non direttamente connessi esula dai fini di questo lavoro, per maggiori dettagli rimandiamo alle specifiche dello standard [1].

Stabilita una HA, bisogna avviare una sessione. Le sessioni sono uniche per spazio delle label, quindi se ho due peers con spazio delle label per platform essi stabiliranno una sola sessione, anche se tra essi vengono identificate più HA, mentre con spazio delle label per interface avrò una sessione per ogni spazio annunciato.
Stabilire una sessione è un procedimento a due fasi:

- stabilire una connessione di trasporto;

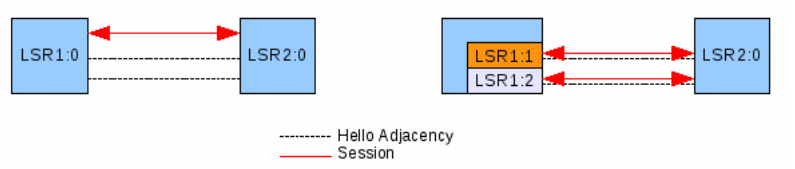


Figura 2.9: A sinistra una configurazione per platform label space, a destra una per interface label space.

- inizializzazione della sessione.

Per stabilire la connessione di trasporto i due peers si mettono d'accordo su chi deve effettuare la connessione facendo da client e chi rimane in ascolto facendo da server, rispettivamente questi due ruoli prendono il nome di *active* e *passive*. Per stabilire il ruolo i due peers confrontano l'indirizzo IP ricevuto con il proprio come se fossero due interi senza segno, chi ha l'intero maggiore svolgerà ruolo attivo e tenterà la connessione TCP alla porta LDP well-known. Stabilita la connessione di trasporto i due peers iniziano la fase di negoziazione dei parametri della sessione, scambiando un messaggio di *LDP Initialization*. Il primo messaggio di inizializzazione lo manda il LSR con ruolo attivo; il LSR con ruolo passivo processa i parametri ricevuti e invia un messaggio di inizializzazione con i propri parametri ed un messaggio di *LDP Keepalive* per segnalare che ha accettato i parametri proposti; il testimone ripassa al LSR attivo che risponderà anch'esso con un keepalive se accetterà i parametri proposti. Una volta stabiliti HA e ses-

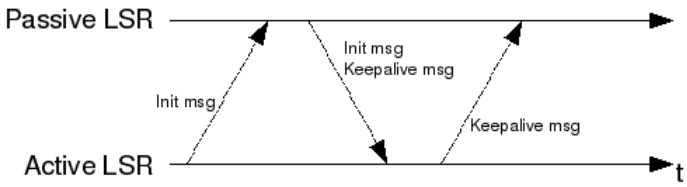


Figura 2.10: Procedura LDP di scambio di messaggi di inizializzazione.

sioni, LDP esegue delle procedure di monitoraggio su esse, basate su timeout e invio periodico di LDP Discovery Hello per ciascuna HA ed LDP Keepalive per ciascuna sessione stabilita. Di seguito è riportata la macchina a stati della fase di inizializzazione.

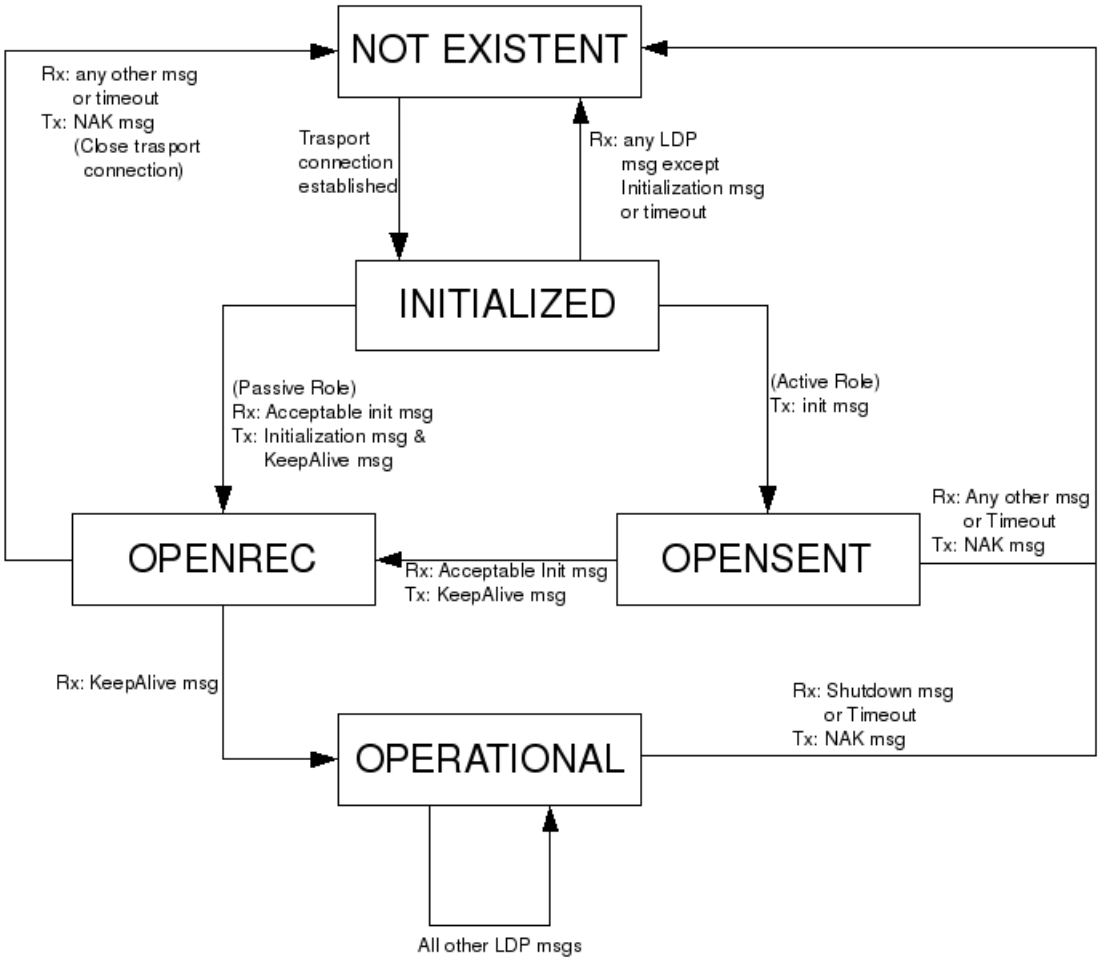


Figura 2.11: Macchina a stati della procedura di inizializzazione:

NOT EXISTENT, sessione ancora non esistente;

INITIALIZED, avvio della fase di inizializzazione;

OPENSENT, stato di un LSR attivo che ha inviato un messaggio di inizializzazione;

OPENREC, stato di un LSR passivo che ha ricevuto un messaggio di inizializzazione e uno di keepalive;

OPERATIONAL, la sessione è stata stabilita.

2.3.4 Distribuzione e gestione delle label

Per quanto riguarda la gestione delle label, LDP distingue tra metodologie di distribuzione e politiche di memorizzazione e mantenimento. Illustreremo prima le modalità di distribuzione delle label, inoltre chiameremo un LSR upstream rispetto ad un altro LSR se procedendo nella direzione del LSP questo si trova prima del nodo di riferimento, invece un LSR downstream si troverà dopo il LSR di riferimento.

Downstream-on-Demand label advertisement
La distribuzione dell'associzione FEC-label avviene solo su esplicita richiesta di un LSR. Il vantaggio della modalità DoD è che l'associazione FEC-label può essere richiesta da un LSR sulla base delle necessità, ossia solo quando l'associazione è strettamente necessaria. Ciò permette un risparmio del numero di label utilizzate e dello spazio di memoria utilizzato (questa modalità è preferita su reti ATM).

Downstream Unsolicited label advertisement
In questa modalità il downstream LSR è responsabile di annunciare un label mapping se vuole che un upstream LSR usi la label. Le specifiche di LDP permettono che un upstream LSR invii una richiesta di mapping al downstream LSR configurato in DU, ma la richiesta potrebbe essere ignorata dal downstream LSR.

Queste due modalità possono essere presenti entrambe nella configurazione dei nodi di una rete, bisogna fare attenzione a dei casi particolari che possono presentarsi con una configurazione eterogenea della rete, in cui non si ha mai scambio di label. Supponiamo di avere un LSR-upstream (Ru) in modalità DU e un LSR-downstream (Rd) in modalità DoD: Rd suppone che Ru richiederà un label mapping quando ne avrà bisogno e Ru suppone che Rd annuncerà una label se vuole che Ru la usi. In questa particolare situazione non si ha scambio di label tra i due LSR. La soluzione sta in una oculata configurazione delle modalità di funzionamento dei nodi della rete.

Ciascuna delle due modalità di distribuzione già viste opera in contemporanea con una delle due seguenti modalità riguardanti la procedura di divulgazione dei label mapping:

Independent control mapping . Se un LSR è configurato in questa modalità,

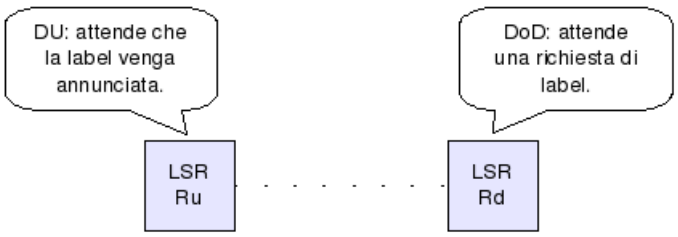


Figura 2.12: Esempio di erronea configurazione delle modalità di diffusione delle label di due LSR della rete.

trasmetterà un messaggio di mapping in seguito a una delle seguenti condizioni:

- LSR riconosce una nuova FEC nella IP routing table, e la modalità di advertisement è DU (figura 2.13a);
- Rd LSR riceve una richiesta LDP dal peer in upstream (Ru) di FEC binding e la FEC è presente nella routing table di Rd LSR (figura 2.13b);
- il next-hop per una FEC cambia a favore di un altro LDP peer ed è settata l'individuazione dei loop;
- gli attributi (come path vector e hop count) di un mapping cambiano;
- LSR riceve un mapping da un downstream next-hop e
  1. non è stato creato ancora un upstream mapping,
  2. l'identificazione dei loop è attiva,
  3. gli attributi del mapping sono cambiati.

Ordered control mapping . Un messaggio di mapping è trasmesso dal downstream LSR in seguito ad una delle seguenti condizioni:

- in modalità DU, LSR riconosce una nuova FEC nella routing table ed è l'egress-router per quella FEC(figura 2.14a);
- in modalità DoD, LSR riceve un messaggio di richiesta da un upstream peer per una FEC che è presente nella forwarding table e il LSR è l'egress per questa FEC o ha un downstream mapping per questa FEC (figura 2.14b);
- il next-hop per una FEC cambia a favore di un altro LDP peer, ed è settata l'individuazione dei loop;

- gli attributi di un mapping (come path vector e hop count) cambiano;
- LSR riceve un mapping da un downstream next-hop e
  1. non è stato creato ancora un upstream mapping,
  2. l'identificazione dei loop è attiva,
  3. gli attributi del mapping sono cambiati.

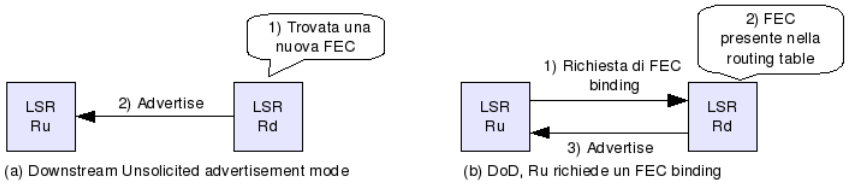


Figura 2.13: Independent control mapping.

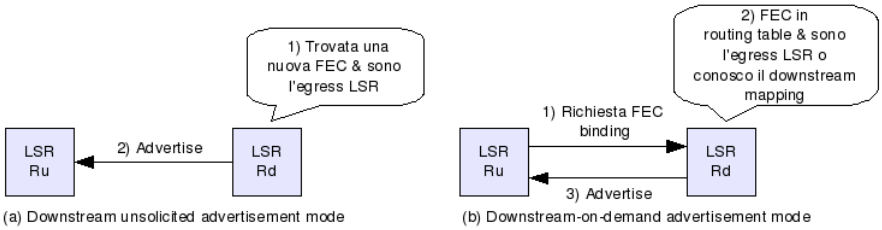


Figura 2.14: Ordered control mapping.

Viste le modalità di distribuzione delle associazioni FEC-label riportiamo le modalità di conservazione di queste informazioni nelle tabelle del LSR:

Liberal retention mode . Il LSR mantiene in memoria, nella sua FTN, tutte le associazioni FEC-label ricevute dagli altri LSR, indipendentemente se questi sono next-hop per la FEC in esame. Il vantaggio di questa modalità consiste nell'abbreviare i tempi di reazione a eventuali cambi di LSP per una data FEC, in quanto le associazioni FEC-label necessarie per stabilire il nuovo LSP sono già presenti in memoria. Per contro ha lo svantaggio di richiedere una maggiore quantità di memoria.

Conservative retention mode . Il LSR mantiene in memoria, nella FTN, solo le associazioni FEC-label necessarie all'inoltro dei pacchetti, ossia quelle ricevute da LSR che sono effettivamente i next-hop di quella FEC. Questa

modalità permette una gestione più efficiente dello spazio di memoria in dispositivi con capacità ridotte, ma necessità per funzionare o di una richiesta esplicita dell'associazione FEC-label necessaria alla costruzione del nuovo LSP, oppure l'attesa dovuta alla ricezione di una associazione da parte del nuovo next-hop.

Per divulgare le informazioni sul next-hop per un dato indirizzo, un LSR invia un messaggio di *LDP Address* contenente il proprio identificatore LDP e gli indirizzi da esso annunciati, questo messaggio attraversa la rete configurando ogni LSR con l'apposito next-hop per un dato indirizzo. Se un next-hop per un indirizzo improvvisamente non è più raggiungibile, il LSR che si accorge della situazione, propaga un messaggio di *LDP Withdrow Address* per ritirale le informazioni riguardati quell'indirizzo.

2.3.5 Struttura dei messaggi

I messaggi LDP sono stati definiti in un formato indipendente dal mezzo. L'intenzione è quella di combinare più messaggi in un singolo datagram per minimizzare l'overhead del processamento di più messaggi separati. Esaminiamo le convenzioni di codifica di un messaggio. Ogni messaggio (fisico) LDP, chiamato *protocol data unit* o PDU, comincia con un *LDP header*, seguito da uno o più messaggi LDP (logici). All'interno dei messaggi le informazioni vengono codificate in strutture *Type-Length-Value* (TLV).

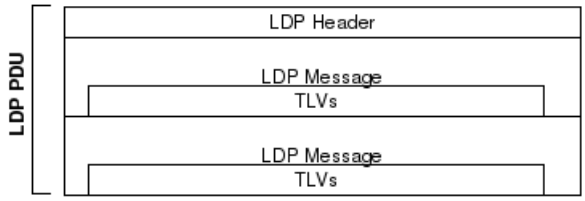


Figura 2.15: LDP PDU contenente due messaggi.

All'interno del LDP Header vengono trasportate tre semplici informazioni: versione del protocollo LDP, lunghezza in byte del PDU (esclusi il campo versione e lunghezza) e l'identificatore LDP del LSR che ha generato il messaggio.
Di seguito al LDP Header vengono codificati i messaggi, strutturati come segue:

- *U bit*, è il bit che indica messaggio sconosciuto. Se settato ad uno indica che il ricevente non è stato in grado di interpretare il messaggio;

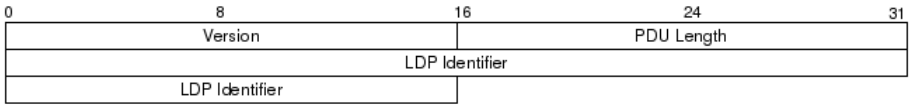


Figura 2.16: LDP Header.

- *Type*, tipo messaggio;
- *Length*, lunghezza messaggio (esclusi i campi U, tipo e lunghezza);
- *Message ID*, è un identificatore del messaggio usato solitamente per associare messaggi di risposta a messaggi di richiesta;
- *Mandatory Parameters*, insieme dei TLV obbligatori in questo messaggio;
- *Optional Parameters*, insieme dei TLV opzionali per questo messaggio.

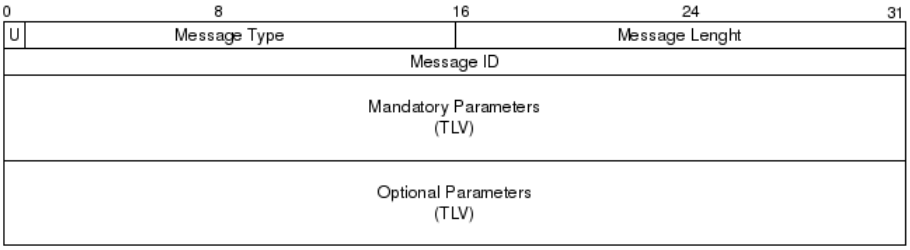


Figura 2.17: Struttura dei campi di un messaggio LDP.

La struttura TLV è usata come contenitore di dati ed è codificata come segue:

- *U bit*, in seguito alla ricezione di un messaggio con un TLV sconosciuto, nel TLV il bit U era settato ad 0, l'intero messaggio viene ignorato e il LSR deve inviare una notifica al LSR che lo ha generato; se invece U era settato ad 1 il messaggio viene processato ignorando il TLV sconosciuto;
- *F bit*, valido solo quando U è settato, indica se il TLV sconosciuto deve essere inoltrato con il messaggio (F=1) o se il TLV deve essere eliminato dal messaggio prima di procedere con l'inoltro (F=0);
- *Type*, specifica il tipo del TLV e quindi i dati trasportati;
- *Length*, lunghezza del TLV in byte;
- *Value*, campo variabile in cui vengono inseriti i dati da trasportare.

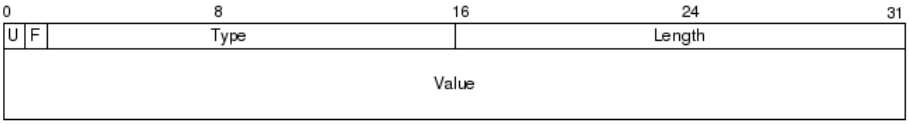


Figura 2.18: Struttura dei campi di un TLV.

Di seguito riportiamo una lista dei messaggi LDP e dei TLV, per maggiori dettagli sulle procedure con cui vengono usati rimandiamo alle specifiche dello standard [1].

Lista messaggi:
Notification Message
Hello Message
Initialization Message
KeepAlive Message
Address Message
Address Withdraw Message
Label Mapping Message
Label Request Message
Label Withdraw Message
Label Release Message
Label Abort Request Message

Lista TLV:
FEC
Address List
Hop Count
Path Vector
Generic Label
ATM Label
Frame Relay Label
Status
Extended Status
Returned PDU
Returned Message
Common Hello Parameters

IPv4 Transport Address
Configuration Sequence Number
IPv6 Transport Address
Common Session Parameters
ATM Session Parameters
Frame Relay Session Parameters
Label Request Message ID

Capitolo 3

Preparazione del testbed

In questo capitolo verrà descritto l'ambiente utilizzato per lo sviluppo e il test di questo lavoro. Il testbed è stato realizzato su base della WMN Roofnet (2.1.2), ideata a Boston dalla collaborazione tra Cambridge Massachusetts e il MIT[1] [3]. Il testbed conta 5 dispositivi collocati all'interno del RedLab nel dipartimento di Ingegneria Informatica, polo A, dell'Università di Pisa. I dispositivi, di cui verrà descritto hardware e software, sono configurati in modo da creare una rete WMN basata sull'algoritmo di routing Srcr. A questi dispositivi possono essere collegati dei client che effettuino richieste di traffico.

3.1 Hardware

Ogni dispositivo consiste di un piccolo calcolatore con architettura x86, equipaggiato con

- due interfacce wireless conformi allo standard 802.11a/b/g e munite di antenna omnidirezionale;
- due interfacce ethernet utilizzate per la connessione ai client.

Le interfacce wireless sono configurate in modo da operare con RTS/CTS disabilitato (il motivo sarà mostrato più avanti) ed utilizzano una modalità non standard "pseudo-IBSS", simile alla modalità IBSS in cui i nodi comunicano direttamente senza AP, la differenza sta nell'omettere la segnalazione con beacons del 802.11 e il meccanismo BSSID (network ID); questo risolve la tendenza della modalità

[1] Massachusetts Istitute of Technology

IBSS di formare partizioni di rete con differente BSSID malgrado avendo lo stesso network ID. Queste partizioni limitano l'affidabilità della rete [3].

3.2 Software ed auto-configurazione

Tutti i dispositivi eseguono lo stesso software:

- sistema operativo OpenWrt [15] basato su Linux;
- software di routing implementato in Click;
- un server DHCP.

Dal punto di vista del client si ha l'impressione di una infrastruttura di rete cablata, la connessione di un client come PC o laptop avviene attraverso l'interfaccia Ethernet del dispositivo, che configura opportunamente il client tramite DHCP.

3.2.1 Click Modular Router

Per la realizzazione di tutte le funzionalità riguardanti il routing, valutazione delle metriche, forwarding MPLS è stato adottato l'ambiente di sviluppo Click Modular Router. Questo software permette di definire semplici elementi per la manipolazione di pacchetti e tramite un linguaggio di scripting, si possono collegare questi elementi tra di loro per svolgere operazioni sempre più complesse. Sono disponibili anche una serie di librerie contenenti elementi che svolgono le operazioni più comuni in un router. Il software e gli elementi sono realizzati in C++ e compilati come modulo del kernel, questo permette di eseguirlo in *kernel space* sostituendosi al tradizionale meccanismo di manipolazione dei pacchetti del kernel. In modalità kernel i pacchetti sono processati molto più velocemente, l'accesso ai pacchetti avviene direttamente dal dispositivo e non è mediato da strati inferiori come avverrebbe se il software venisse eseguito in modalità *user-level*. Per maggiori dettagli sul software rimandiamo all'ampia documentazione riportata on-line [11, 12]. In questo lavoro faremo spesso riferimento alla struttura a blocchi utilizzata per rappresentare l'organizzazione degli elementi nello script di configurazione di click, per questo motivo di seguito daremo delle indicazioni su come interpretare questi grafici.

Un elemento di Click è rappresentato con un riquadro, all'interno abbiamo il nome della classe a cui questo oggetto appartiene (corrisponde al nome della classe

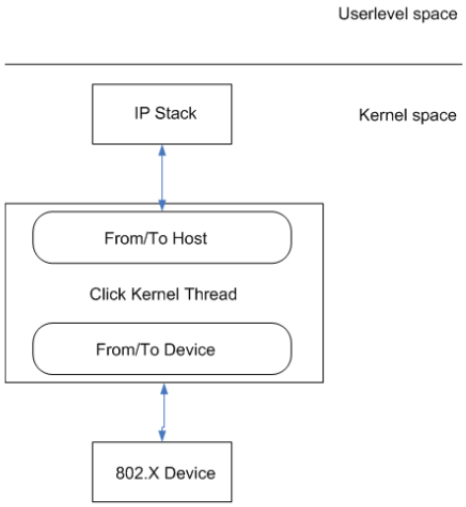


Figura 3.1: Schema generale di Click in kernel mode.

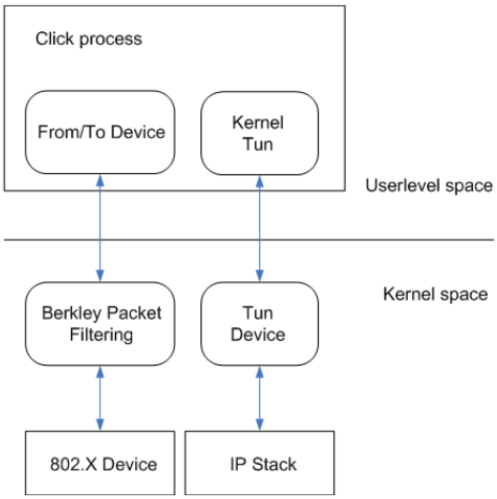


Figura 3.2: Schema generale di Click in modalià userlevel.

C++ che deriva dalla classe base Element) e tra parentesi la sua stringa di configurazione. Sul bordo del riquadro sono rappresentati dei triangoli o dei quadrati, rispettivamente corrispondono alle porte di ingresso e di uscita dell'elemento. Le connessioni tra le porte di due elementi sono raffigurate con tratto continuo e su esse transitano dei pacchetti, con tratto discontinuo si intende invece che l'oggetto da cui parte la freccia contiene un riferimento all'oggetto puntato e quindi può usare i metodi e i campi pubblici di quest'ultimo. In figura 3.4 è riportato

lo schema di uno script Click che realizza un Ethernet switch, dove sono presenti sia connessioni che riferimenti, i pacchetti vengono catturati dall'elemento FromDevice e inviati dall'elemento ToDevice.

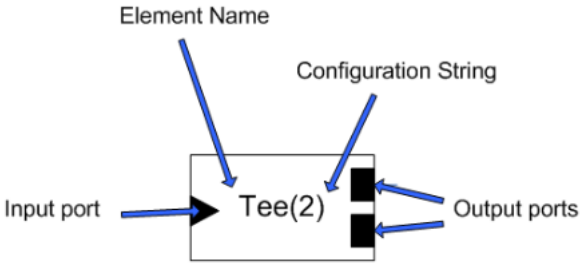


Figura 3.3: Rappresentazione di un elemento Click.

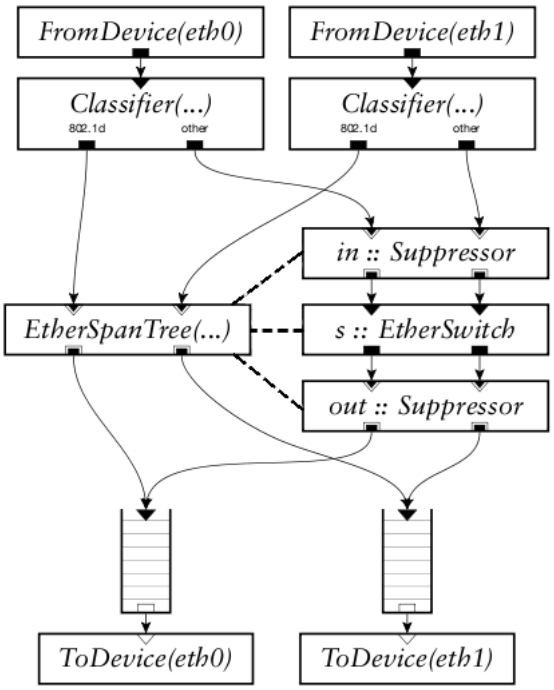


Figura 3.4: Schema a blocchi di uno script Click che realizza uno switch Ethernet.

3.2.2 Indirizzamento

Come in Roofnet i nodi trasportano i pacchetti IP dentro un proprio formato di header chiamato Roofnet layer. Ogni nodo richiede un indirizzo univoco e deve

essere in grado di configurare il proprio indirizzo senza far richieste esplicite ad altri nodi. Tutto questo viene realizzato scegliendo un indirizzo IP i cui 24 bits meno significativi corrispondono ai 24 bit meno significativi dell'indirizzo MAC della prima interfaccia Ethernet, e gli 8 bit più significativi corrispondono ad un blocco di indirizzi di classe A. Più in dettaglio se sono X.X.X i 24 bit recuperati dall'indirizzo MAC abbiamo che l'indirizzo identificativo del nodo sarà 5.X.X.X, mentre gli indirizzi delle interfacce wireless sono 3.X.X.X (ath0) e 4.X.X.X (ath1), questi due indirizzi non sono globalmente routeble, hanno senso solamente per il protocollo di routing e per il Roofnet layer per specificare il percorso del pacchetto. Gli indirizzi delle interfacce Ethernet invece sono 1.0.0.1 (eth0) e 2.0.0.1 (eth1) e rispettivamente assegnano al client che vi si collega l'indirizzo 1.X.X.X e 2.X.X.X, in questo modo l'indirizzo del client identifica anche il nodo a cui è connesso.

Interfaccia	Net	Indirizzo
eth0	1.0.0.0/8	1.0.0.1
eth1	2.0.0.0/8	2.0.0.1
ath0	3.0.0.0/8	3.X.X.X
ath1	4.0.0.0/8	4.X.X.X
srcr	5.0.0.0/8	5.X.X.X

Tabella 3.1: Indirizzamento adottato da ogni nodo della WMN.

3.3 Protocollo di routing, Srcr

Come in Roofnet, il protocollo di routing adottato è Srcr, un protocollo di tipo reactive che cerca di trovare le rotte con più alto throughput tra ogni coppia di nodi della rete. L'impiego di antenne omnidirezionali da a Srcr un'ampia scelta di link che potrebbe usare, ma la maggior parte sono di bassa qualità, Srcr deve valutare il throughpup utilizzabile di ogni link. La realizzazione di Srcr è dovuta a recenti studi e misurazioni sul comportamento delle reti wireless nel mondo reale [16, 17, 18].

Srcr è un protocollo di source-routing[2] come DSR [19, 20], per evitare la formazione di loop nel routing quando le metriche dei link cambiano. Ogni nodo

[2]I protocolli di souce-routing specificano nel pacchetto non solo l'indirizzo del destinatario ma anche la lista dei nodi da attraversare.

Srcr mantiene un database parziale con le metriche dei link tra coppie di nodi, e usa l'algoritmo di Dijkstra su questo database per calcolare le rotte. Un nodo apprende nuove metriche su un link in tre modi:

1. un nodo che invia un pacchetto include la metrica corrente nel source-route del pacchetto, in questo modo tutti i nodi lungo la rotta possono acquisire informazioni sulla metrica;

2. se un nodo deve inviare un pacchetto per una destinazione di cui non ha la rotta nel suo database corrente, invia una richiesta nello stile DSR cioè attraverso flooding e aggiunge le metriche apprese da tutte le risposte nel proprio database;

3. un nodo può ascoltare la richiesta e la risposta di un'altra coppia di nodi e acquisire informazioni sulla metrica di quel link ed aggiungerla al proprio database.

Questa combinazione di link-state e sistema di query come DSR è stata ispirata da MCL [18].

Un problema di questa tecnica è che l'inondazione di richieste spesso non segue il percorso migliore. Srcr risolve questo problema operando come segue: quando un nodo riceve una nuova richiesta, aggiunge l'informazione sulla metrica del link, trasportata nella source-route del pacchetto, al proprio link database; poi il nodo calcola la rotta migliore dalla sorgente della richiesta e sostituisce la source-route del pacchetto con la rotta migliore; infine il nodo ripropaga in broadcast la richiesta. Se il nodo in futuro riceverà un'altra copia della richiesta ed è in grado di calcolare nuovamente una rotta migliore, con una metrica più bassa, reinoltrerà la richiesta con la nuova rotta. In pratica, molti nodi propagano una richiesta solo una o due volte. Dagli studi su Roofnet si è visto che in una rete più grande e densa, dove i nodi mandano dati a molte destinazioni, è necessario un meccanismo di query più scalabile.

Le condizioni di un link possono cambiare ottenendo che una rotta attiva non è più la rotta migliore. Se un link di una rotta attiva interrompe del tutto il forwarding dei pacchetti, il nodo upstream invierà una notifica alla sorgente di ogni pacchetto di cui è fallito l'inoltro. Se invece un link di una rotta attiva semplicemente degrada le prestazioni, la sorgente apprenderà la nuova metrica del link dai pacchetti che lo attraversano. Infine la sorgente rieseguirà l'algoritmo di Dijkstra sul proprio database dei link, ogniqualvolta apprende il cambiamento

di una metrica e il router cambierà rotta se è stato in grado di calcolarne una migliore di quelle adoperate.

3.4 Metriche

Srcr sceglie le rotte in base alla metrica *Estimated Trasmission Time* (ETT), che deriva dalla metrica *Estimated Trasmission count* (ETX) [17]. ETT predice l'ammontare di tempo che occorre per inviare un pacchetto su una rotta, tenendo conto del massimo throughput di ogni link e della probabilità di consegna del pacchetto a quel bit-rate. Srcr sceglie il percorso con il più basso valore di ETT, affinché questa rotta sia in grado di consegnare più pacchetti nell'unità di tempo.

Ogni nodo invia periodicamente in broadcast 1500 byte ad ogni bit-rate disponibile in 802.11 e un pacchetto di dimensione minima, 60 byte, a un Mbit/s. I nodi tengono traccia della frazione di questi broadcast ricevuti, e rispondono con i relativi valori statistici calcolati. Srcr deduce che il più alto throughput di un link è il bit-rate con il più elevato prodotto tra probabilità di consegna e bit-rate. La probabilità di consegna ad un certo bit-rate è il prodotto tra la frazione di 1500-byte ricevuti e la frazione di 60-byte a un Mbit/s in ricevuti.

La metrica ETT per un dato link è il tempo previsto per l'invio con successo di un pacchetto di 1500 byte al massimo throughput del link, compreso il tempo di eventuali ritrasmissioni predette dalla probabilità di consegna. La metrica ETT di una rotta è la somma degli ETT di ogni link del percorso. Srcr assume la seguente relazione tra throughput t_i con un hop della rotta e il throughput t end-to-end della rotta:

$$t = \frac{1}{\sum_i \frac{1}{t_i}}$$

Hops	Campioni	Throughput con RTS/CTS	Throughput senza RTS/CTS
1	3	2094	1735
2	5	836	725
3	6	314	312

Tabella 3.2: Throughput in Kbits/s sperimentato con e senza RTS/CTS.

Questa relazione è abbastanza accurata per percorsi corti, dove al massimo trasmette un nodo per volta [21]. Invece rappresenta un throughput sottostimato

per rotte più lunghe nelle quali hop distanti possono inviare simultaneamente senza interferire, ed è sovrastimato quando le trasmissioni su due nodi differenti collidono e vengono perse. Questo tipologia di problemi si risolverebbe attivando RTS/CTS che preverrebbe le collisioni. Dalle misure effettuate su Roofnet, l'utilizzo di RTS/CTS però non migliora le performance, in tabella 3.2 è mostrato il throughput sperimentato con e senza RTS/CTS su un numero casuale di percorsi a 1, 2 e 3 hops.

Capitolo 4

Fase di progettazione

In questo capitolo verranno discusse le scelte fatte per il raggiungimento degli obiettivi proposti. Cominceremo con le modifiche da apportare al sistema di label switching (4.1) in modo da supportare le funzionalità di PHP e FRR (fast reroute), che verranno opportunamente configurate dal sistema di distribuzione delle label. Di seguito verranno proposte delle soluzioni per la realizzazione del software (LDP Light, LDPL) che si occuperà dello scambio e della configurazione delle label (4.2), quindi motiveremo la propensione per LDP e analizzeremo le modalità di funzionamento con riferimento al contesto WMN. Infine verrà descritta la tecnica di individuazione e installazione di detour e come poterla integrare come estensione del software LDPL (4.3).

4.1 Modifiche al label switching

Come detto precedentemente, tutte le funzionalità dei nostri dispositivi sono state realizzate in Click: dall'algoritmo di routing e la misura delle metriche fino al label switching MPLS. Descriveremo la struttura dello script di configurazione di Click per meglio comprendere dove bisognerà agire per aggiungere le funzionalità di PHP e FRR da noi richieste.

In figura 4.1 è rappresentata il comportamento generale dello script Click che governa il dispositivo: in alto si trovano i blocchi che si occupano di catturare e inviare i pacchetti dalle interfacce fisiche, EthDevice corrispondono alle rispettive interfacce Ethernet, WirelessDev corrisponde alle interfacce wireless, quella più interessante, LinuxHost, non corrisponde a nessuna interfaccia fisica: dato che

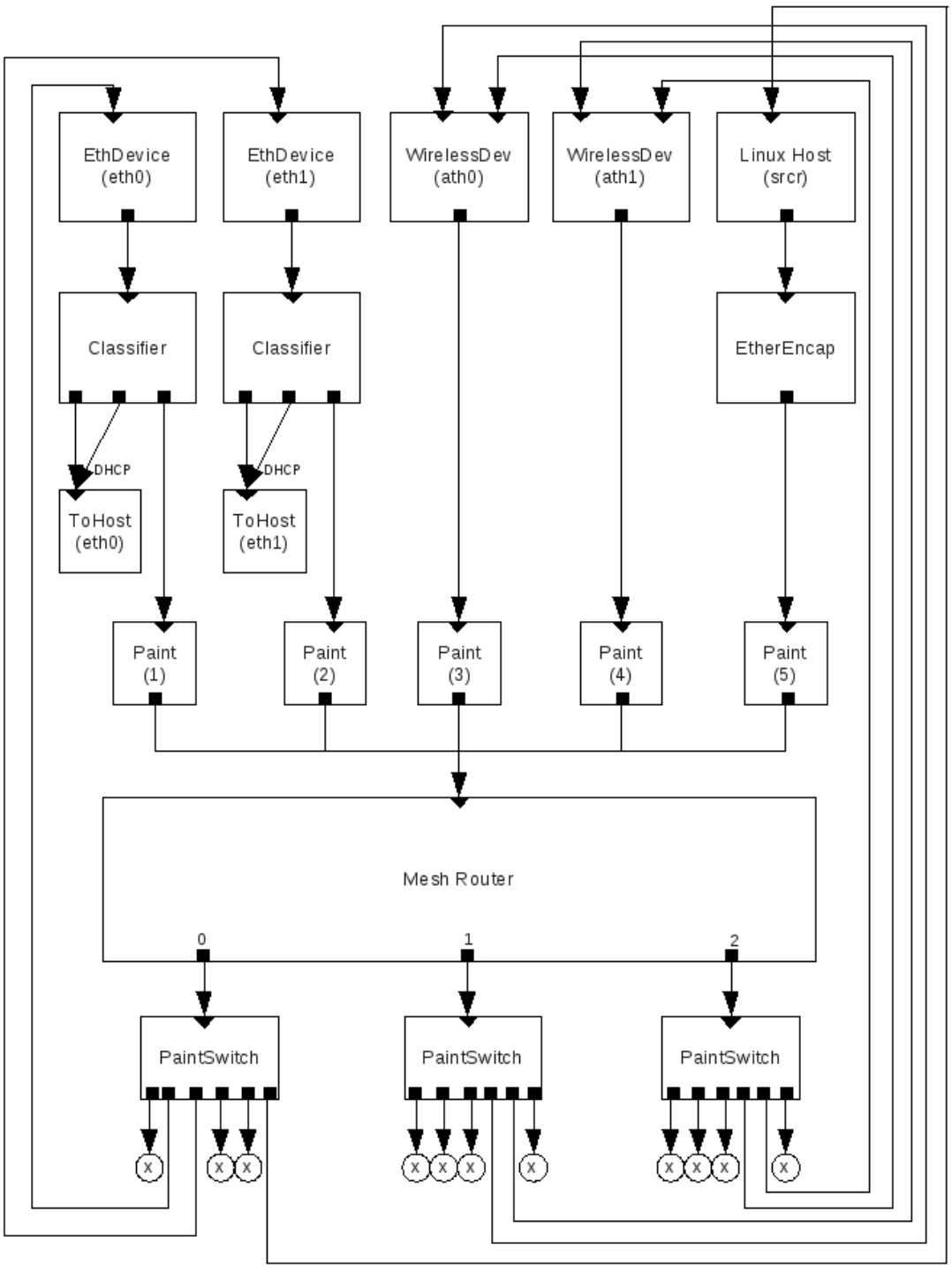


Figura 4.1: Struttura generale dello script Click dei router usati per il testbed.

Click governa la gestione delle interfacce non è più possibile operare direttamente su esse da parte del sistema; vengono quindi create delle interfacce virtuali, che

gestiscono il traffico generato e ricevuto da i processi in esecuzione sul Sistema Operativo; questa interfaccia quindi rappresenta l'unica via di comunicazione per i processi interni. Seguendo la strada di un pacchetto ricevuto da uno dei blocchi interfaccia, vediamo che il pacchetto viene marcato per specificare da quale interfaccia è arrivato (elementi Paint) e successivamente passato al blocco Mesh Router in cui sono implementate tutte le funzionalità del router. Processato dal blocco Mesh Router, il pacchetto viene immesso su uno di tre possibili output: sulla porta 0 escono i pacchetti diretti verso le interfacce Ethernet e verso LinuxHost, sulla porta 1 e 2 escono i pacchetti diretti alle interfacce wireless, la differenza tra i due percorsi è che su uno transitano i pacchetti dati diretti al mezzo wireless e sull'altro i pacchetti per la gestione della rete mesh (messaggi del routing protocol e della misura delle metriche); questi pacchetti entrando su due porte differenti del blocco WirelessDev, vengono inseriti in una coda prioritaria che favorisce prima l'invio dei messaggi di gestione e controllo della rete.

Risulta evidente che le modifiche da apportare a questa struttura si concentrano all'interno del blocco Mesh Router. In figura 4.2 è rappresentata la struttura interna del blocco Mesh Router: i pacchetti entrano dall'unica porta di input del blocco e vengono immessi in un classificatore che distingue sul valore del campo EtherType: valori come 0x0943/0x0945 sono riservati ai messaggi di routing e ai pacchetti dati che transitano sul mezzo wireless inoltrati dal routing tradizionale, il valore 0x0941 è riservato al sistema di misurazione delle metriche basato su messaggi di probe. I rami che ci interessano principalmente sono quello dedicato al traffico IP (0x0800) e quello dedicato al traffico MPLS (0x8847).

In azzurro, sono evidenziati gli elementi che detengono le tre tabelle necessarie per il forwarding MPLS: l'elemento FTN si trova all'inizio del ramo relativo ai pacchetti IP, confrontandoli al loro arrivo con le FEC dei LSP installati. L'elemento ILM si trova all'inizio del ramo MPLS e per ogni pacchetto in ingresso cerca una corrispondenza con le label in tabella. Trovata una corrispondenza, sia FTN che ILM, associano al pacchetto in indice della tabella NHLFE e lo inviano in ingresso all'elemento NHLFE; quest'ultimo elemento assocerà in base all'indice assegnato un'azione (di PUSH dato che arriva dalla FTN) e un next-hop.

L'elemento NHLFE viene configurato con i MAC address delle interfacce riconosciute per il forwarding e setta il numero di porte di uscita di conseguenza: sulla porta 0 escono i pacchetti che in seguito ad una operazione di POP, contengono ancora un livello di shim header e quindi la label verrà riprocessata dalla

ILM, dalla porta 1 escono i pacchetti che in seguito ad una operazione di POP, non hanno più uno shim header e vengono affidati al routing tradizionale. Si noti che in entrambi i due casi non si fa uso del next-hop, in seguito ad una POP non viene considerato questo campo e il pacchetto viene riprocessato dallo stesso LSR. Quando un pacchetto viene proposto per il forwarding dalla NHLFE, questo viene fatto uscire da una porta con indice maggiore a 1; queste porte corrispondono alle interfacce con cui viene configurata NHLFE, nello stesso ordine crescente. Il pacchetto a questo punto avrà source-MAC quello dell'interfaccia di uscita e destination-MAC il next-hop, ma prima di essere inviato deve passare dall'elemento SetPaintInterface che in base al source-MAC marca il pacchetto per l'invio dalla interfaccia opportuna (per maggiori dettagli sugli elementi MPLS [2]).

Abbiamo identificato dove intervenire per aggiungere la funzionalità di PHP, modificando i sorgenti dell'elemento NHLFE possiamo far in modo che anche in seguito ad una operazione di POP consideri il campo next-hop. Dato che il next-hop adesso sarà sempre valido, si distingueranno i pacchetti destinati allo stesso LSR da quelli da inoltrare perchè contrassegnati con uno dei MAC address usati per la configurazione di NHLFE (una sorta di "next-hop-self"). Questa modifica da sola non è sufficiente, infatti nel caso in cui ho PHP con un solo shim header, otterremo l'inoltro di un *pacchetto MPLS senza shim header*; l'ultimo hop, identifica il pacchetto MPLS dal EtherType, ma finendo in ingresso all'elemento ILM, verrà scartato generando come errore "shim header non valido". Per risolvere questo problema, abbiamo aggiunto un elemento che riconosce se siamo in presenza di una header IP o di uno shim header, il pacchetto esce dalla porta 0 primo caso altrimenti dalla porta 1. In figura 4.3 sono state aggiunte le modifiche appena proposte.

Per quanto riguarda la funzionalità di FRR, occorre un sistema che associa ad un evento, identificato come l'arrivo di un pacchetto dalla FTN o dalla ILM, non più una sola azione rappresentata da una entry della tabella NHLFE, ma almeno due azioni con la possibilità di commutare da una all'altra e viceversa, mantenendo consistenti le tabelle FTN, ILM e NHLFE. Inoltre occorre un'ulteriore rettifica alla NHLFE, attualmente questo elemento applica una azione di PUSH, POP e REPLACE per volta, ma come già visto nel paragrafo 2.2.2, occorre poter effettuare combinazioni di azioni in sequenza, questo è stato ottenuto modificando la struttura della tabella NHLFE: ogni entry della tabella è identificata da un indice,

lo stesso a cui fanno riferimento FTN e ILM, aggiungendo un campo che contiene l'indice dell'azione successiva si ottiene un sistema versatile per la realizzazione di operazioni complesse come una REPLACE seguita da una PUSH o due POP in cascata; basterà poi prestabilire un valore per l'indice che identifichi l'ultima operazione della sequenza. Tornando al FRR, otteniamo l'effetto voluto aggiungendo un nuovo elemento che mantiene una struttura che, come già detto, associa ad un evento due azioni (due indici di ingresso nella tabella NHLFE), questa struttura prende il nome di *Fast Reroute Database* (FRRdb). Questo nuovo elemento, oltre a mantenere le informazioni sul FRR, dovrà operare sui campi delle tabelle FTN e ILM, tramite un riferimento al rispettivo elemento; inoltre dovrà fornire un'interfaccia che permetta di aggiungere, togliere informazioni dal FRRdb e attivare, disattivare la commutazione per un data entry della FTN e ILM. In figura 4.4 è riportato l'elemento MeshRouter con le modifiche per supportare FRR.

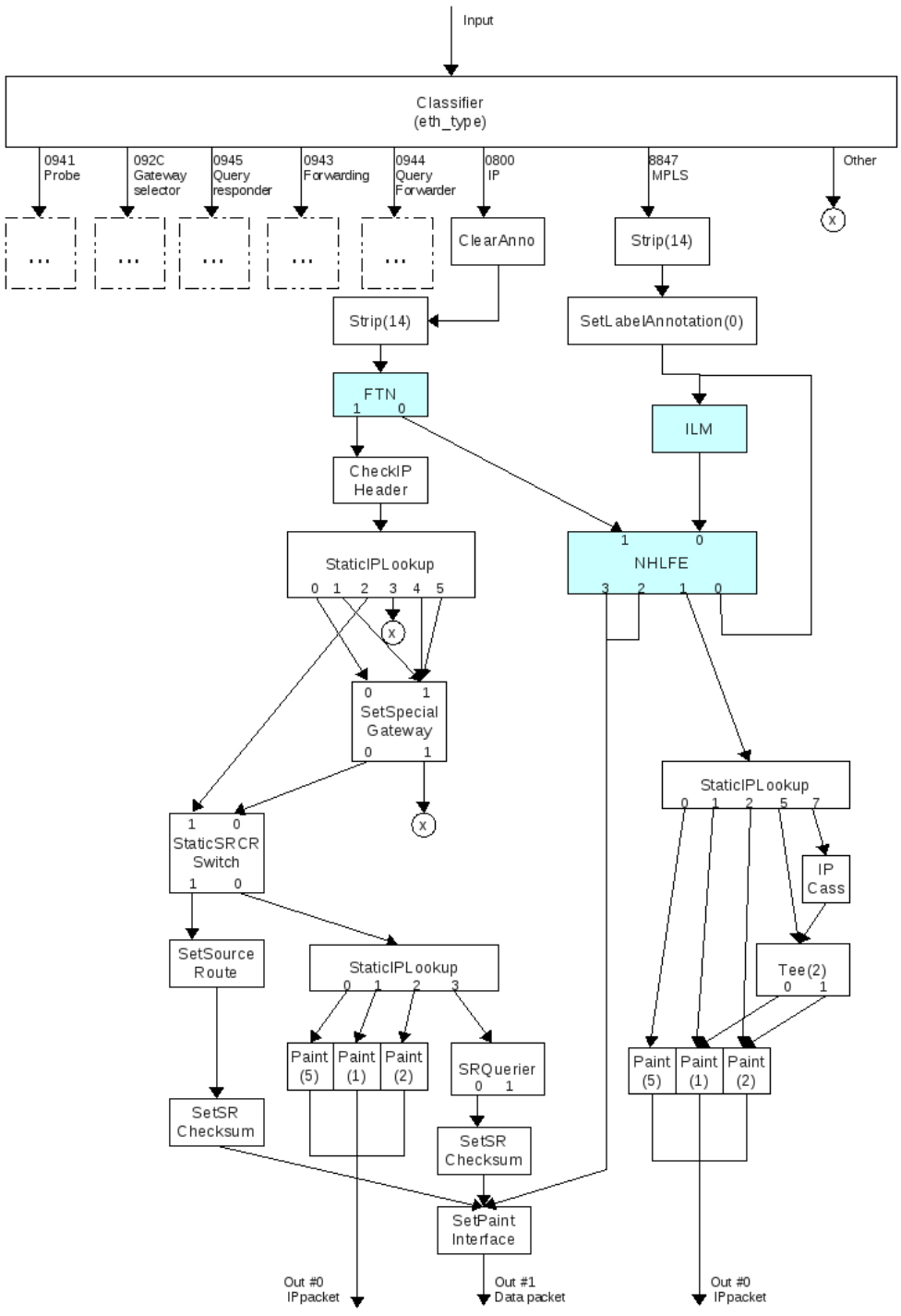


Figura 4.2: Struttura interna dell'elemento MeshRouter. Per una migliore lettura i triangoli e i quadrati che identificavano una porta sono stati omessi. Nel caso di più porte in ingresso o uscita, queste sono state contrassegnate dall'indice che le identifica; nel caso di una sola porta di ingresso o uscita è da considerarsi come porta 0.

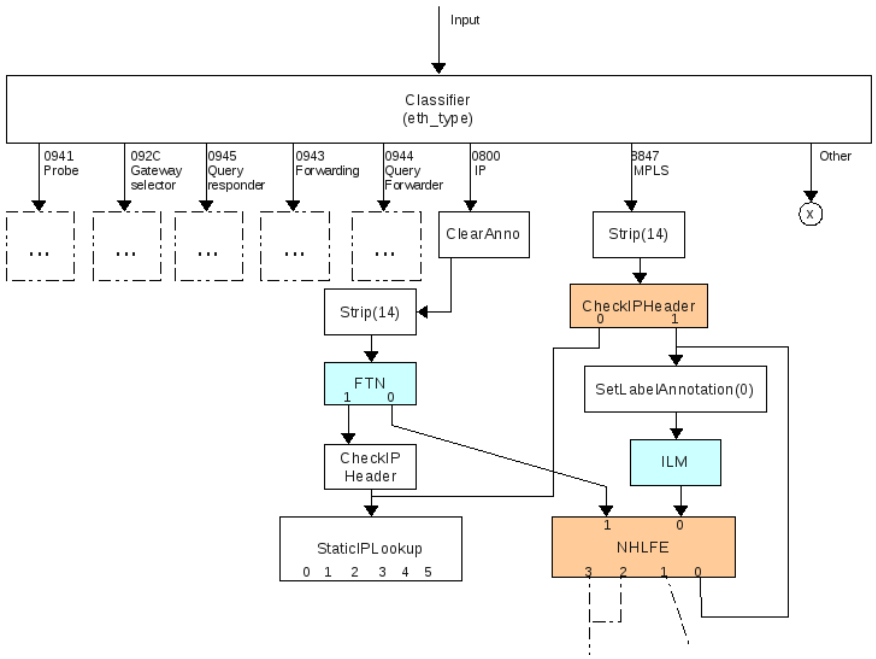


Figura 4.3: Struttura elemento MeshRouter con aggiunta della funzionalità di PHP. In Arancio gli elementi aggiunti e modificati.

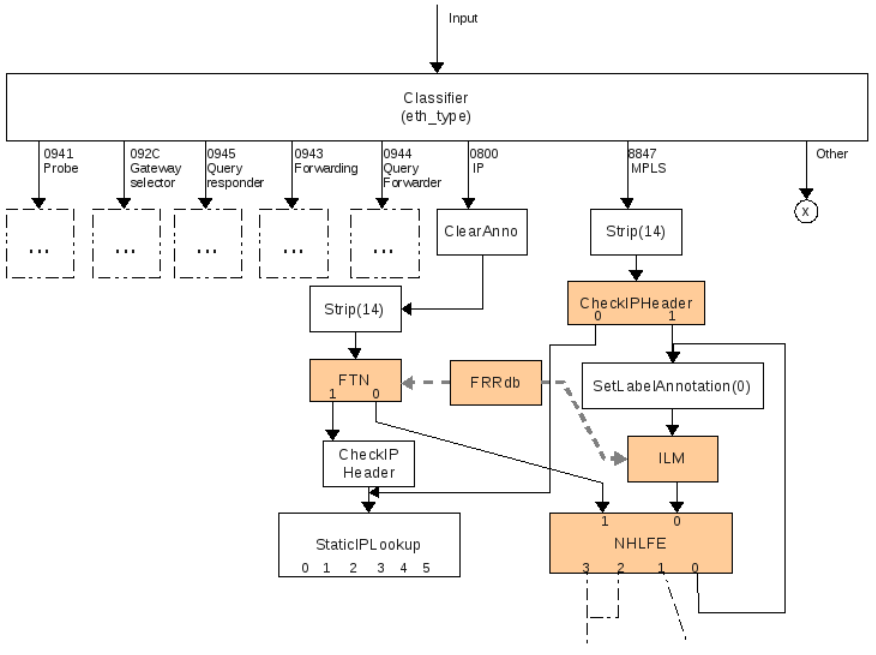


Figura 4.4: Struttura elemento MeshRouter con aggiunta della funzionalità di PHP e FRR. In Arancione gli elementi aggiunti e modificati.

4.2 Protocollo di distribuzione delle label

Per sfruttare le potenzialità di un sistema di label switching occorre un protocollo di configurazione e scambio delle label. Gli elementi Click che detengono la gestione delle tabelle FTN, ILM e NHLFE forniscono una interfaccia[1] che permette la loro configurazione. L'obiettivo che si vuole raggiungere con un sistema di distribuzione delle label è la configurazione di queste tabelle MPLS al fine di installare e rimuovere un LSP. Le soluzioni adottate generalmente da MPLS sono due: *Label Distribution Protocol* (LDP) e *Resorce Reservation Protocol* (RSVP). In questo lavoro si è optato per un approccio del tipo LDP per i seguenti motivi:

1. i messaggi LDP viaggiano su protocollo di trasporto TCP che fornisce delle garanzie sulla consegna del pacchetto, i messaggi RSVP sono inseriti nel payload di un pacchetto IP;
2. RSVP è un sistema di gestione più complesso di LDP, in quanto non si occupa soltanto dello scambio delle label ma anche della prenotazione delle risorse a scopo di traffic engineering, funzioni che esulano dallo scopo di questo lavoro.

Fissato il sistema di riferimento è stata fatta un'analisi per stabilire quali componenti di LDP sfruttare, quali omettere e quali modificare ai nostri scopi.

Come descritto nella sezione 2.3.3, il sistema di discovery e sessioni usato da LDP è molto versatile e si presta bene a un contesto cablato quanto ad un contesto di rete WMN soggetto a repentini cambiamenti; per questo motivo si è deciso di adottarlo senza nessuna modifica. Lo stesso vale per il sistema di identificazione LDP, si è deciso di lasciare il supporto per entrambe le due metodologie, per platform e per interface, anche se in ambito di questa ricerca verrà adottata solamente la metodologia per platform; il supporto al per interface label space è stato mantenuto nell'ipotesi che possa essere utile per lavori futuri assegnare uno spazio delle label per ogni interfaccia wireless.

Le implementazioni parziali riguardano invece la parte sulla distribuzione e conservazione delle label, il set di messaggi usati e di conseguenza le procedure implementate. Tra le diverse metodologie di distribuzione delle label è stata implementata solamente *Downstream on Demand con Ordered Control mapping* per le seguenti ragioni:

[1] L'interfaccia è realizzata tramite handlers [13, 11, 14].

1. la modalità DoD fornisce diversi vantaggi rispetto alla UD in un contesto WMN: il traffico MPLS non rappresenta l'unico tipo di traffico anzi il label switching è considerato come un mezzo privilegiato da assegnare ad una categoria di flussi con vincoli temporali stringenti o che richiedano un certo livello di robustezza nei confronti di interruzioni improvvise della rete, di conseguenza un modalità DoD che si attiva solamente in caso di necessità di un LSP è più indicata rispetto ad una modalità UD che mira ad installare anticipatamente potenziali LSP. Inoltre, in presenza di un mezzo condiviso, quale quello wireless, è importante introdurre il minor overhead possibile sotto forma di messaggi di controllo; la modalità DoD rappresenta la soluzione migliore in questo contesto; senza trascurare che questa modalità riduce il set di label utilizzate unicamente a quelle necessarie per il forwarding.

2. La modalità Ordered Control mapping prevede che una richiesta di label mapping venga propagata fino all'egress-LSR, anche se un LSR intermedio già possiede un label mapping per la FEC annunciata. Questo forza la creazione di LSP indipendenti dato che non viene fatta aggregazione sui LSR intermedi. Questa soluzione è stata preferita perchè permette un controllo maggiore sul singolo flusso e semplifica le operazioni di ritiro di un LSP.

Per quanto riguarda la conservazione delle label è stata adottata la modalità *Conservative* dato che vengono conservate soltanto le label assegnate dai LSR downstream. Non è stato implementato invece il sistema di divulgazione degli indirizzi di un LSR tramite messaggi di LDP address perchè avrebbe aggiunto overhead sulla rete WMN, e perchè le informazioni divulgate da questo sistema sono superflue nella struttura del nostro testbed. Il sistema di divulgazione degli indirizzi di un LSR serve ad informare gli altri LSR della rete dei possibili next-hop per un certo indirizzo, questa informazione può essere recuperata facilmente dalla tabella di routing di Srcr; essendo un protocollo source-routing esso associa ad un indirizzo da raggiungere l'intero percorso lungo la rete, in questo modo il nostro sistema di distribuzione delle label può risalire al next-hop per una data FEC; inoltre eseguendo l'installazione di un LSP sulla base delle informazioni calcolate da Srcr, si potrebbero monitorare i cambiamenti di rotta di Srcr per una certa destinazione (dovuti all'interruzione di un link o alla variazione delle metriche) e decidere se installare un nuovo LSP per la nuova rotta calcolata. Un ulteriore vantaggio di mappare un LSP con la rispettiva rotta Srcr è la sicurezza

(entro i limiti dell'algoritmo di routing, vedi sezioni 3.3 e 3.4) di sfruttare il percorso con throughput maggiore.

Operando delle riduzioni alle procedure LDP implementate, di conseguenza anche il set di messaggi utilizzati si riduce. Sono stati implementati tutti i messaggi di Discovery e inizializzazione sessione, quindi messaggi *Hello*, *Initialization*, *KeepAlive* e i messaggi di installazione e ritiro di un LSP: *Label Mapping*, *Label Request* e *Label Release*; tutti questi messaggi sono stati accompagnati dall'implementazione dei loro relativi TLV.

Il software di gestione delle label, che per comodità chiameremo LDPL (Label Distribution Protocol Light), abbiamo deciso di realizzarlo come processo utente eseguito dal sistema OpenWrt di ogni LSR, al posto di una implementazione ad elementi integrata in Click; questo perchè a livello kernel alcune operazioni non sono possibili, come la gestione dei timers e le sessioni TCP. L'intero software è stato realizzato con un approccio parametrico, con la possibilità di variare modalità di funzionamento del protocollo, variabili temporali e opzioni di configurazione. Abbiamo realizzato anche un software di controllo remoto, in grado di comunicare con ciascuna instanza di LDPL in esecuzione sulla rete ed acquisire informazioni sullo stato e inviare i comandi di installazione e rimozione di LSP.

Il software LDPL è stato realizzato seguendo la struttura in figura 4.5, i blocchi all'interno di LDPL rappresentano le unità operative che svolgono i seguenti compiti: scoperta di altri LSR adiacenti con instanze LDPL in esecuzione (Discovery System), l'inizializzazione e mantenimenti delle sessioni LDP (Session System), lo scambio di messaggi per l'installazione e rimozione LSP (Core System), l'interazione con il pannello di controllo remoto (Control System) e infine l'analisi delle rotte calcolate dal protocollo di routing per verificare la corrispondenza tra LSP installati e rotte correnti. Come si vede in figura 4.5 il software LDPL, in particolare il Core System e il Path Analizer System, interagisce con Click per il recupero di informazioni sul routing e per la configurazione delle tabelle MPLS, grazie alle interfacce messe a disposizione dagli elementi Click corrispondenti (handler Click [13, 11, 14]).

4.3 Individuazione e installazione di Detour

Un *Detour* è un backup-LSP con delle particolari caratteristiche. Lo scopo è quello di deviare dal LSP principale in modo da saltare il punto di interruzione

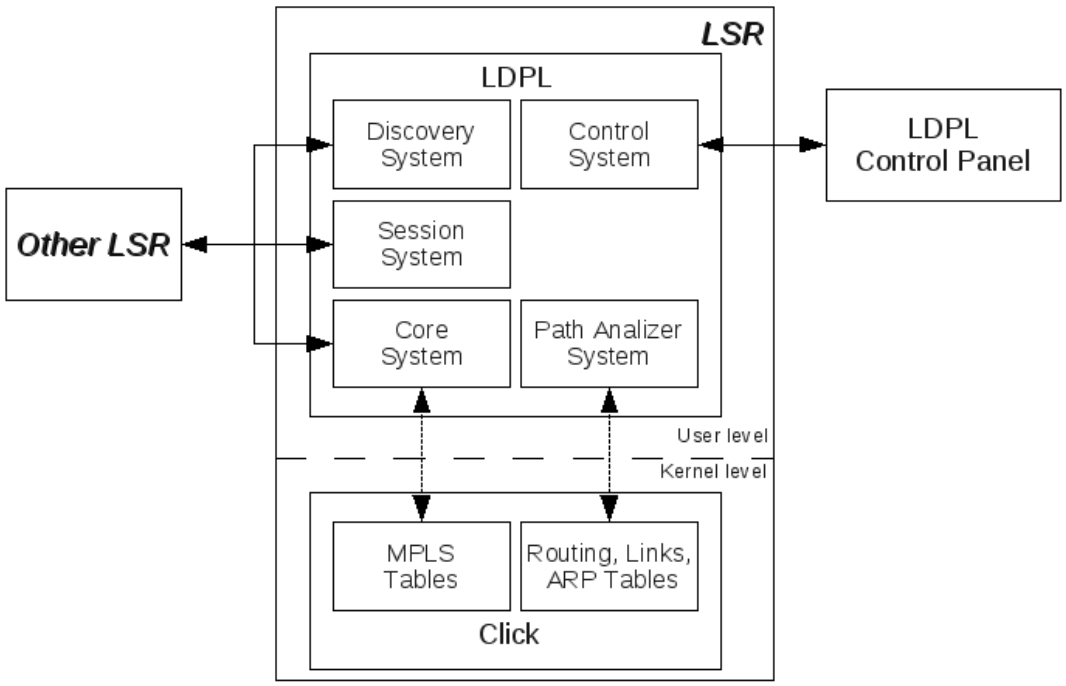


Figura 4.5: Struttura a blocchi e interazioni di LDPL .

o guasto e ritornare sul LSP nel nodo successivo al guasto; questo comporta che il LSP con detour manterrà la stessa lunghezza del LSP principale. A causa di questa particolare formulazione, un detour è sempre di lunghezza pari a due nodi e il nodo intermedi è in connessione diretta con i due LSR del LSP principale da collegare; quest'ultima affermazione anche se ovvia non è da sottovalutare e vedremo più avanti le motivazioni.

La procedura d'individuazione e installazione di un detour è stata realizzata come estensione del software LDPL. Abbiamo deciso di creare i percorsi di detour in fase di installazione del LSP, perchè durante questa procedura abbiamo lo scambio di informazioni riguardanti le label e sarà sufficiente operare su queste informazioni per aggiungere la funzionalità da noi richiesta con il minor impatto possibile sulla struttura del processo LDPL. Analizzeremo le procedure standard di installazione e rilascio di un LSP per poter individuare dove e come agire.

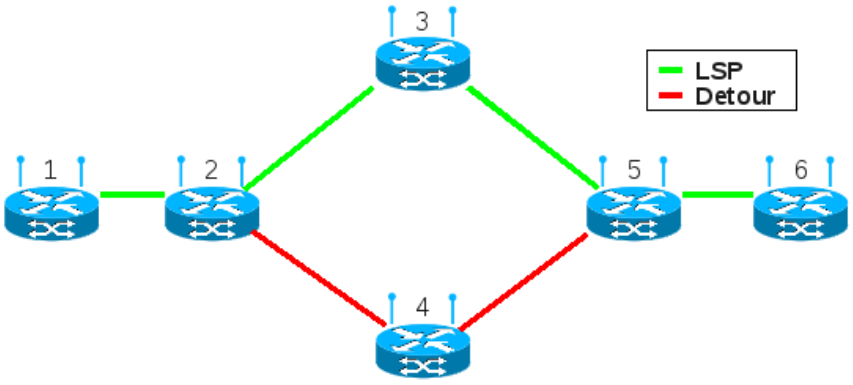


Figura 4.6: Un esempio di detour. A tratto discontinuo le connessioni wireless. In verde il percorso principale e in rosso il detour che protegge sia da interruzione del link tra LSR2 e LSR3 che da guasti di LSR3.

4.3.1 Procedura standard di installazione di un LSP

La procedura standard di installazione di un LSP prevede due fasi: la prima di propagazione del messaggio *Label Request* dal ingress-LSR fino al egress-LSR, seguendo la rotta stabilita dal protocollo di routing per l'indirizzo specificato nella FEC; la seconda fase parte dal egress-LSR, questo dopo aver ricevuto il messaggio di richiesta, propaga a ritroso, lungo lo stesso percorso, un messaggio di *Label Mapping* contenente una label per quel LSP. In questo modo ogni LSR informa della propria label il LSR precedente che aggiornerà di conseguenza le proprie tabelle ILM e NHLFE fino a giungere al punto di origine dell'LSP che installando una entry nella sua FTN e NHLFE termina la procedura.

In figura 4.7 è riportata graficamente la procedura appena descritta. La richiesta parte dal LSR1 e termina nel LSR6.

4.3.2 Procedura standard di ritiro di un LSP

La procedura standard di ritiro di un LSP è più semplice di quella di installazione, prevede solamente la propagazione lungo il LSP di un messaggio di *Label Release* contenente la label da eliminare nel LSR successivo. Il LSR aggiorna le tabelle FTN, ILM ed NHLFE cancellando l'entrata corrispondente. L'unica particolarità si presenta nell'evenienza che il LSP sia interrotto (dovuta al guasto di un LSR o interruzione di un link): la procedura prosegue normalmente fino al LSR antecedente l'interruzione, per i LSR successivi non c'è modo di eliminare le

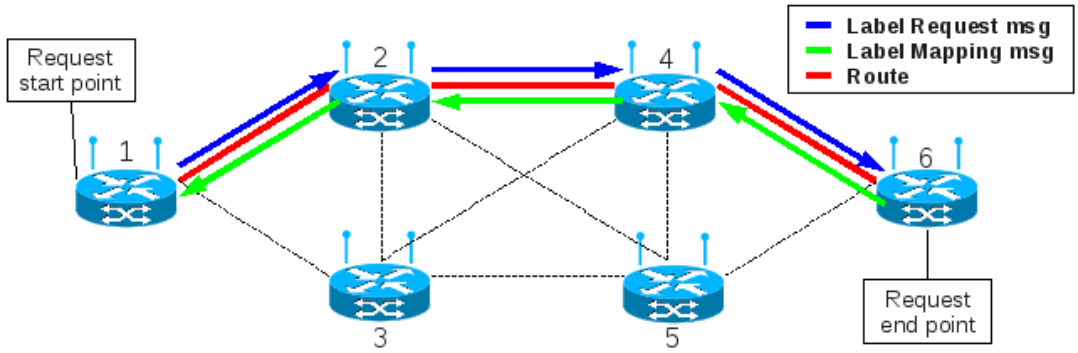


Figura 4.7: Procedura di installazione di un LSP. A tratto discontinuo le sessioni LDP stabilite tra i LSR. In rosso la rotta stabilita dal protocollo di routing, in blue la propagazione dei messaggi di Label Request e in verde la propagazione dei messaggi di Label Mapping.

entry relative al LSP cancellato salvo che intervenendo manualmente sulle tabelle MPLS. In questo lavoro non è stato previsto per semplicità un sistema di *Label Garbage Collection*, in questo caso bisognerebbe monitorare il traffico su ogni tratto di un LSP e stabilire un insieme di condizioni che permettano di considerare quel tratto non più utilizzato per poi procedere al ritiro della label. Un primo approccio si potrebbe basare su dei timers, ma bisogna fare attenzione al fatto che un LSP potrebbe restare inattivo per molto tempo facendo scattare inopportunamente il meccanismo di label garbage collection. In figura 4.8 è riportata graficamente la procedura di rilascio di un LSP.

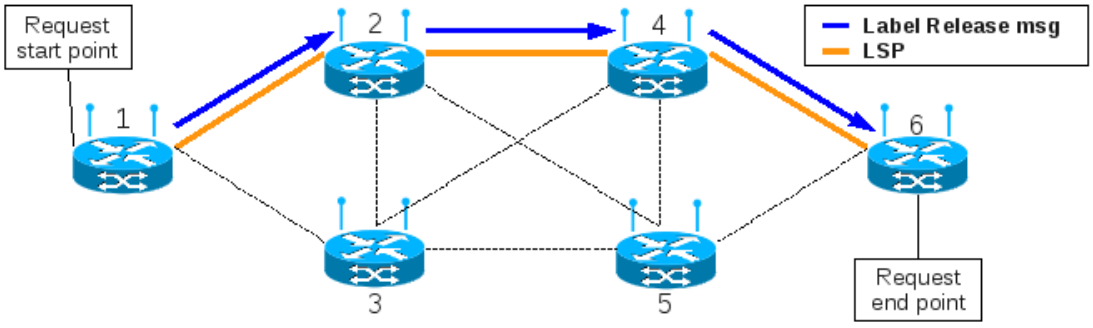


Figura 4.8: Procedura di ritiro di un LSP. A tratto discontinuo le sessioni LDP stabilite tra i LSR. In arancio il percorso del LSP e in blue la propagazione dei messaggi di Label Release.

4.3.3 Determinazione del Next-Next-Hop

Il problema principali di una applicazione che sfrutta FRR consiste nel recuperare una label che abbia significato per un LSR che non è direttamente connesso. Con riferimento alla figura 4.6, LSR2 se vuole usare il detour deve prima conoscere la label che adopera LSR3 per inoltrare il traffico verso LSR5; questo perchè uscendo dal detour il pacchetto deve giungere a LSR5 come se fosse stato inviato da LSR3. La soluzione adottata consiste nel mantenere la label del LSR5 nel messaggio di label mapping non solo fino a LSR3 ma anche nel messaggio da LSR3 a LSR2. Questa label prende il nome di *Next-Next-Hop Label* (NNHL) [22], viene creato così un nuovo TLV contenente la NNHL e aggiunto al messaggio di label mapping. Quindi la procedura di divulgazione della NNHL opera come segue: riferendoci alla figura 4.9, LSR6 inserisce nel messaggio di label mapping il proprio NNHL-TLV e lo invia a LSR4, questo non interpreterà il NNHL-TLV e lo propagherà aggiungendo un ulteriore NNHL-TLV (il proprio). I NNHL-TLV cominceranno ad essere interpretati da un LSR quando si trovano in numero uguale a due, e vengono gestiti come in una coda FIFO[2]; LSR2 troverà due NNHL-TLV nel messaggio di label mapping ricevuto, il primo conterrà la NNHL di LSR6. A sua volta LSR2 estrarrà la label valida ed inserirà il proprio NNHL-TLV, proseguendo allo stesso modo in LSR1.

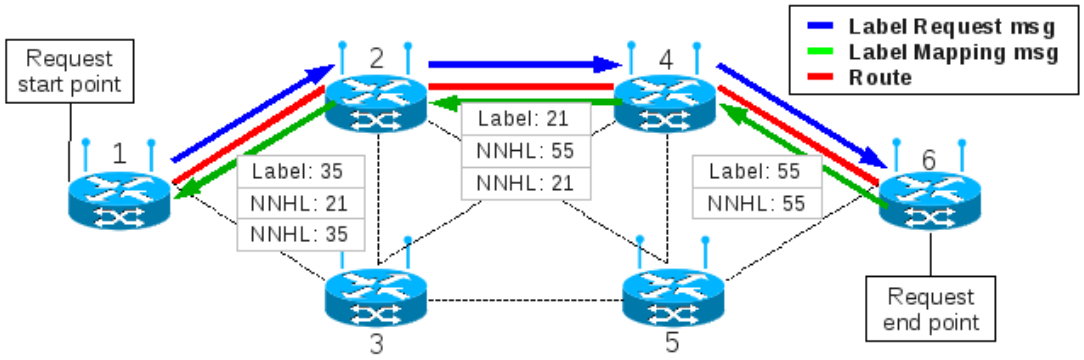


Figura 4.9: Esempio di diffusione della NNHL nel messaggio di label mapping (in verde).

Siamo riusciti a far ottenere ad un LSR la NNHL; il problema successivo sarà stabilire verso quale LSR proseguire per l'installazione del detour, o meglio trovare un LSR in comune tra i due LSR ai capi del detour. Per ottenere questa informazione è sufficiente aggiungere nel NNHL-TLV la lista degli identifi-

[2]First In - First Out

catori dei neighbor-LSR[3] (tranne quello su cui passa il LSP). Il LSR che interpreterà il NNHL-TLV calcolerà l'intersezione dei due insiemi rappresentati dai propri neighbor-LRS e da quelli ricevuti, i LSR ottenuti sono possibili candidati a formare un detour.

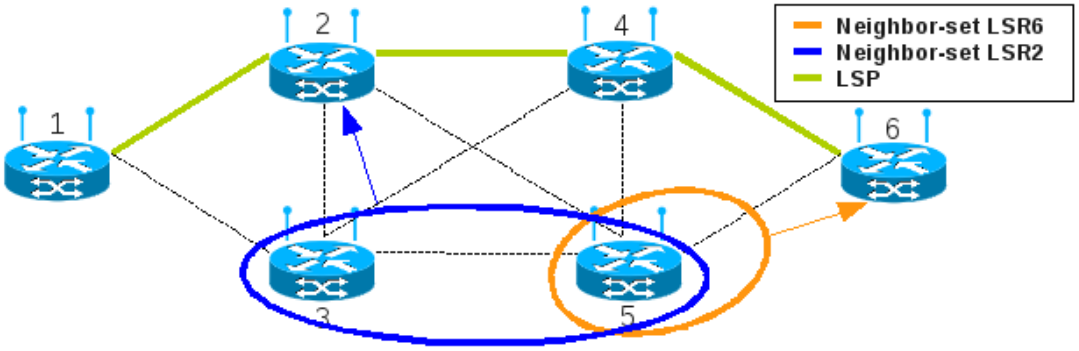


Figura 4.10: L'intersezione dei due insiemi neighbor-LSR rappresenta il LSR candidato per un detour da LSR2 a LSR6.

Le politiche che stabiliscono quale dei LSR candidati per il detour sarà effettivamente quello adottato possono essere diverse, ma uno studio di questo tipo esula dagli obiettivi di questo lavoro; noi ci siamo limitati a scegliere il primo LSR riscontrato nell'intersezione dei due insiemi.

A questo punto diamo definitivamente la struttura del NNHL-TLV: questo conterrà la NNHL, un campo con l'identificatore del LSR che lo ha generato e il neighbor-set. In figura 4.12 è rappresentata graficamente la procedura con lo scambio delle informazioni per l'installazione di un detour.

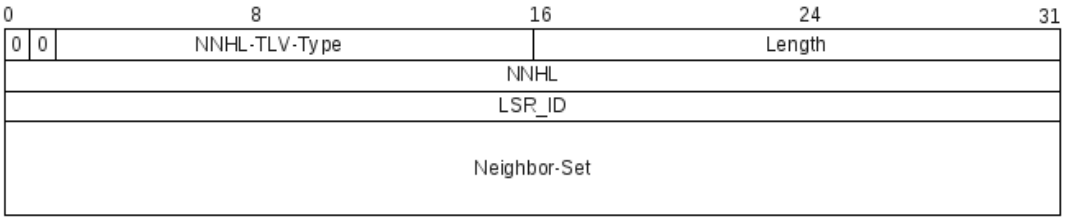


Figura 4.11: Struttura di un NNHL-TLV.

Recuperate tutte le informazioni per l'installazione del detour, il LSR avvia una nuova procedura di installazione di LSP leggermente modificata. Si definisco-

[3]LSR direttamente connessi.

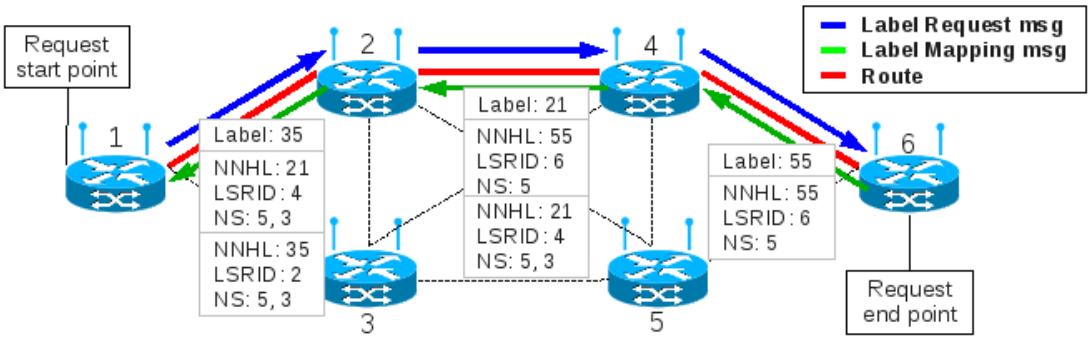


Figura 4.12: Procedura di installazione di un LSP con l'aggiunta del NNHL-TLV contenente le informazioni per l'installazione del detour (NS: neighbor-set).

no due nuovi messaggi di richiesta e mapping per differire dall'installazione del LSP standard. Questi due messaggi, che chiameremo *Label Detour Request* e *Label Detour Mapping*, differiscono dai corrispondenti Label Request e Label Mapping perchè nel primo la FEC richiesta non corrisponde più ad un indirizzo ma all'identificatore LDP del LSR destinazione, mentre nel secondo non viene trasportato il NNHL-TLV. Il tipo diverso dei messaggi serve anche a distinguere su quali tabelle MPLS andare ad operare, in questo caso FRRdb e NHLFE per il primo LSR del detour, ILM e NHLFE per il secondo LSR del detour. Si noti inoltre che si può effettuare l'installazione di un detour senza avere scambio di messaggi con l'ultimo LSR del detour, sfruttando la funzione di PHP e la conoscenza a priori della lunghezza del detour: il secondo LSR del detour sa già di essere il penultimo hop, quindi può installare subito l'operazione di POP e assegnare al primo LSR una label. Questa ottimizzazione permette di ridurre parecchio il numero di messaggi scambiati in fase di installazione di un LSP, specialmente in un contesto di rete fortemente magliata dove il numero di detour installabili lungo un LSP è elevata.

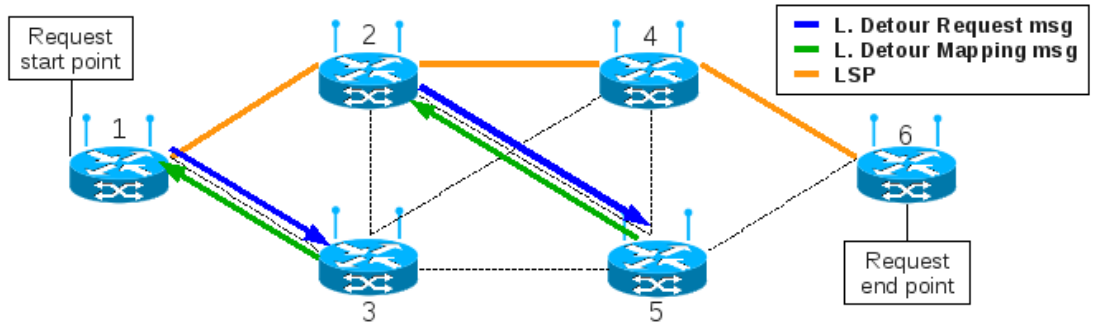


Figura 4.13: Fase di installazione dei detour. In blue i messaggi di Label Detour Request e in verde i messaggi di Label Detour Mapping.

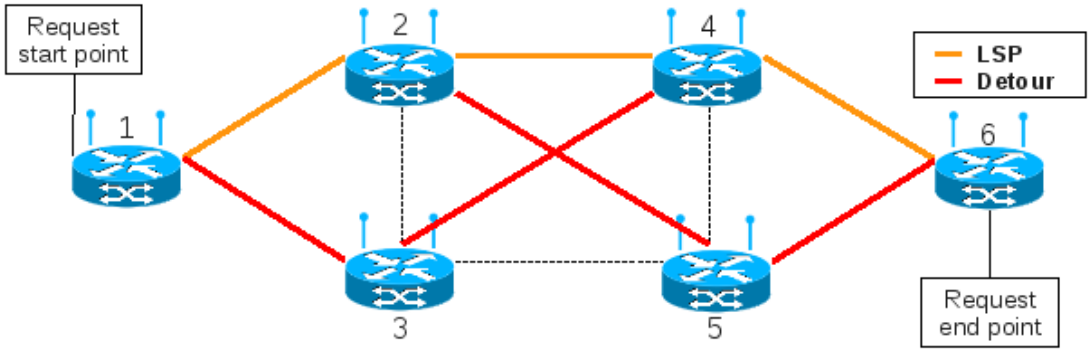


Figura 4.14: Scenario dopo l'installazione del LSP con detour. In arancio LSP principale, in rosso il detour a protezione di LSR4 e in viola il detour a protezione di LSR2.

Capitolo 5

Implementazione

In questo capitolo descriveremo la fase implementativa di questo lavoro. Cominceremo esponendo le ulteriori modifiche da applicare allo script Click per supportare completamente le operazioni di LDPL (5.1), successivamente descriveremo la realizzazione del software di gestione delle label LDPL (sezione 5.2), proseguendo con l'implementazione dell'elemento FRRdb di Click adibito all'attivazione del fast reroute (sezione 5.3). Infine mostreremo il software di gestione della rete che permette il controllo da remoto dei singoli LDPL peer per l'installazione e rimozione di LSP e la visualizzazione del loro stato 5.4.

5.1 Modifiche dello script Click a supporto di LDPL

Prima di cominciare con l'implementazione del protocollo di scambio delle label abbiamo dovuto affrontare diversi problemi legati alla struttura del router realizzata in Click e alla modalità in cui quest'ultimo gestisce le interfacce di comunicazione. Il primo problema che ci si è posto davanti riguarda il sistema di diffusione in multicast dei messaggi di Hello che naturalmente dovrebbe sfruttare il broadcast naturale del mezzo wireless; ma come descritto precedentemente, ad un processo utente non è permesso l'utilizzo diretto delle interfacce wireless, esso può soltanto comunicare attraverso l'interfaccia virtuale fornita da Click, è lo script di Click che infine deciderà su quale interfaccia indirizzare il pacchetto. Inoltre analizzando il ramo IP dello script Click dell'elemento meshrouter, abbiamo appurato che il traffico in broadcast/multicast veniva comunque scartato. L'elemento responsabile corrisponde al primo elemento StaticIPLookup incontrato sul

ramo IP, il quale riconosce i pacchetti con indirizzo IP multicast/broadcast. La soluzione a questo problema prevede una ulteriore modifica allo script Click che permette di riconoscere i pacchetti IP multicast a noi interessati e li immette direttamente nel mezzo wireless da entrambe le interfacce; inoltre la modifica dovrà alterare l'header IP, cambiando l'indirizzo sorgente con il rispettivo indirizzo utilizzato per identificare la interfaccia wireless di uscita. Il motivo di questa operazione è legato alla necessità di risalire all'informazione riguardante l'interfaccia di arrivo di un pacchetto Hello: senza questa modifica, il pacchetto giungerebbe con una coppia di indirizzi sorgente-destinazione del tipo 5.x.x.x-224.0.0.3 (l'indirizzo usato per l'invio multicast dei messaggi Hello Discovery) e non ci sarebbe alcun modo per sapere da quale interfaccia è stato ricevuto, ricevendo un pacchetto del tipo 3.x.x.x-224.0.0.3 posso ricercare nella tabella dei link, in cui sono mantenute tutte le coppie di indirizzi che identificano un link, una entry con sorgente 3.x.x.x e destinazione il router in questione [3|4].y.y.y, trovando in questo modo l'informazione cercata. Infine questa informazione costituirà una entry della nostra tabella delle Hello Adjacency (vedi più avanti). Le modifiche che hanno permesso questo sono riportate in fig. 5.1, la parte di script da noi aggiunta preleva i pacchetti in multicast/broadcast, li classifica a seconda se contengono l'indirizzo di multicast 224.0.0.3 e se sono pacchetti in ingresso o in uscita. I pacchetti ricevuti dall'esterno vengono marcati con destinazione l'interfaccia virtuale (Paint(5)), i pacchetti destinati alla trasmissione vengono sdoppiati (Tee(2)) e per ciascuno viene assegnato l'indirizzo sorgente dell'interfaccia di uscita, ricalcolato il checksum IP (FixIPChecsums), incapsulato in un frame ethernet con MAC sorgente quello dell'interfaccia di uscita e destinazione quello di broadcast, infine il pacchetto viene marcato con destinazione la rispettiva interfaccia wireless specificata nel MAC sorgente e fornito in ingresso alla porta per il traffico prioritario dell'elemento WirelessDev. In questo modo anche se usiamo un indirizzo di multicast questo verrà trattato come broadcast, sarà poi il ricevitore a riconoscere l'indirizzo multicast IP e far emergere il pacchetto dalla propria interfaccia virtuale.

Di seguito riportiamo il codice di configurazione dell'elemento IPClassifier:

```
IPClassifier(
  (src host 5.20.186.236) and (dst host 224.0.0.3 or 5.255.255.255),
  src host 3.20.186.236 or 4.20.186.236,
  src net 3.0.0.0/8 or 4.0.0.0/8,
  -
);
```

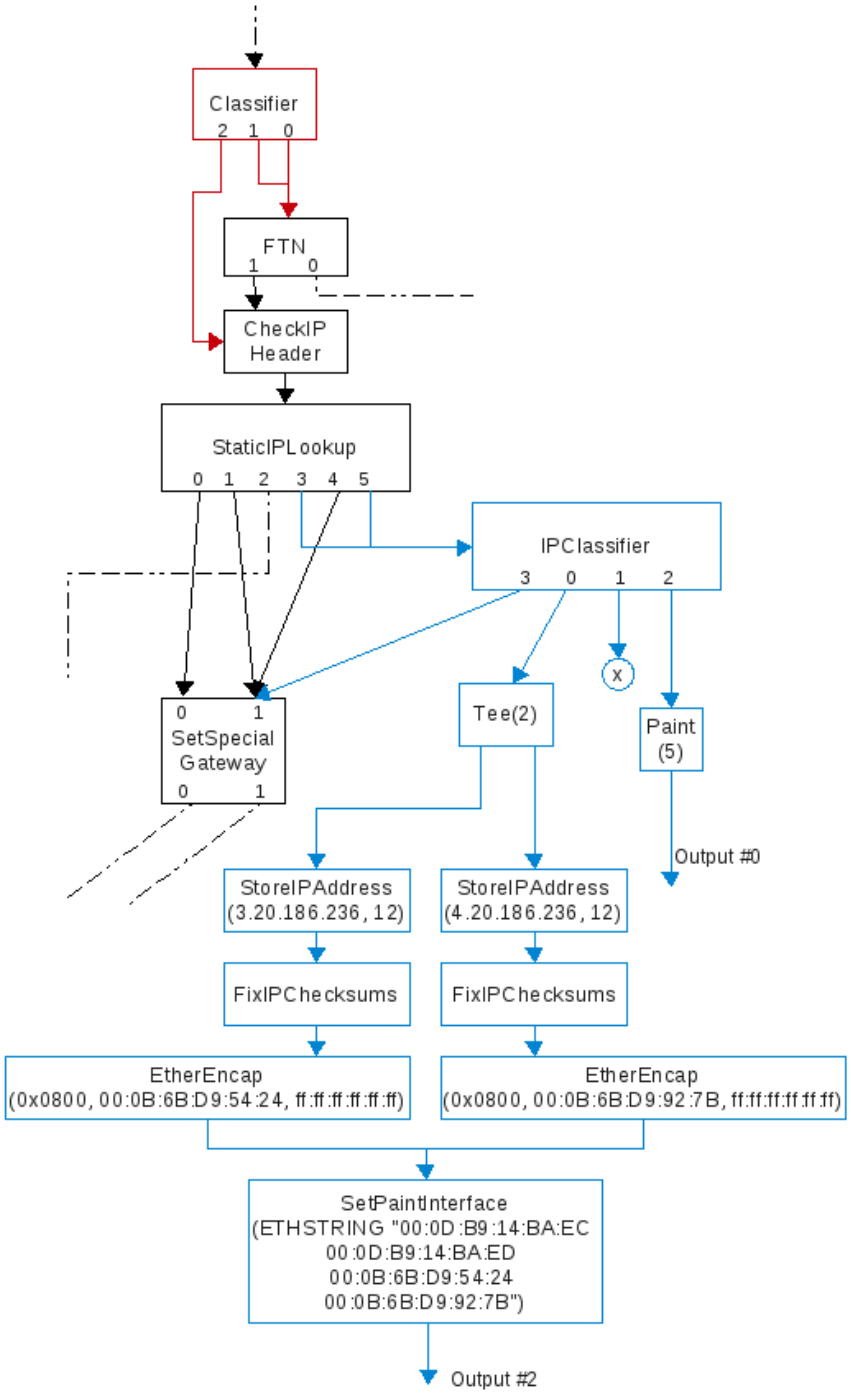


Figura 5.1: In azzurro i componenti aggiunti per supportare l'invio e ricezione dei messaggi di Hello Discovery in broadcast dalle interfacce wireless; in rosso la modifica adottata per non far transitare i messaggi di controllo sul LSP.

Il codice appartiene alla macchina identificata dall'indirizzo 5.20.186.236; la prima riga della stringa di configurazione identifica i pacchetti broadcast e multicast LDPL discovery in uscita (porta 0), la seconda i pacchetti in multicast trasmessi e ricevuti dallo stesso dispositivo (porta 1), questo si verifica nel caso eccezionale in cui entrambe le interfacce wireless sono configurate sullo sesso canale fisico, i pacchetti in uscita da questa porta vengono scartati. La terza riga della stringa di configurazione identifica i pacchetti ricevuti (porta 2), l'ultima, dato che queste regole sono processate sequenzialmente, combacia con tutti gli altri casi (porta 3). L'altro elemento che abbiamo realizzato è quello che si occupa del ricalcolo del checksum, ne riportiamo il codice:

```
elementclass FixIPChecksums {
      input -> SetIPChecksum -> ipc :: IPClassifier(tcp, udp, -)
          -> SetTCPChecksum -> output;
      ipc[1] -> SetUDPChecksum -> output;
      ipc[2] -> output
}
```

Un'altra condizione indesiderata è la seguente, supponiamo di avere un LSP per il LSRy, la FEC corrisponderà all'indirizzo host del LSR (5.x.x.x) e tutto il traffico che arriverà all'ingress-LSR destinato a LSRy verrà dirottato sul LSP, ciò che non vogliamo è che anche i messaggi inviati dal pannello di controllo di LDPL viaggino sul LSP, questo perchè se c'è una inconsistenza nel LSP il LSRy potrebbe risultare isolato dal pannello di controllo remoto. La soluzione consiste nell'aggiungere un elemento che anticipa la FTN e se rileva un messaggio con la porta specificata per i messaggi di controllo, oltrepassa l'elemento FTN. Questa modifica è inserita in cima al ramo IP come si può vedere in fig. 5.1 e riportiamo il codice di configurazione dell'elemento:

```
Classifier(
  23/11  34/0290, //UDP 656 in
  23/11  36/0290, //UDP 656 out
  -
);
```

5.2 LDPL

Il software LDPL è stato implementato con il linguaggio di programmazione C++ come processo utente; si è fatto uso delle librerie *pthread* per il supporto alla programmazione multithread e le librerie *socket* per le comunicazioni UDP

e TCP. Il paradigma di programmazione seguito consiste nella suddivisione delle operazioni in unità indipendenti a cui viene assegnato un thread; queste unità di esecuzione sono supportate da delle strutture dati condivise con accesso in mutua esclusione dove occorre garantire la consistenza dei dati. Queste unità di esecuzione appartengono ad uno dei blocchi operazionali identificati in fig. 4.5.

Di seguito descriveremo le strutture dati utilizzate, le funzioni svolte dai singoli thread e loro interrelazioni.

5.2.1 Strutture dati

In questa sezione descriveremo le strutture dati e spiegheremo il loro utilizzo e le motivazioni che hanno spinto alla loro realizzazione.

Interfaces

Questa struttura dati gestisce l'utilizzo delle interfacce che vengono usate nel label switching, è stata pensata per una questione di compatibilità con sistemi in cui è possibile l'accesso alle interfacce fisiche, nel nostro sistema questa struttura si occuperà solamente dell'interfaccia virtuale fornita da Click.

La struttura consiste in un vettore di elementi *IntEntry*, che identificano un'interfaccia tramite i valori di indirizzo, socket UDP in lettura sull'indirizzo di multicast, socket UDP in scrittura sull'indirizzo di multicast e socket TCP in ascolto per le richieste di apertura sessione. Aggiunte tutte le interfacce interessate alla comunicazione LDPL, sarà sufficiente interagire con questa struttura per inviare o ricevere qualsiasi messaggio di discovery e le richieste di apertura sessione.

L'oggetto andrà inizializzato con alcuni parametri comuni a tutte le interfacce:

- porta adottata dal protocollo (646);
- TTL per i messaggi di multicast (1);
- opzione di loopback dei messaggi di multicast;
- indirizzo multicast usato;
- lunghezza della coda di backlog per le richieste TCP.

Inizializzato l'oggetto, sarà possibile aggiungere e togliere le interfacce tramite i metodi *addint* e *delint* specificando il loro indirizzo. Settate le interfacce potrò inviare i messaggi di multicast usando il metodo *sendtoall* specificando il buffer

e la dimensione, mentre per ricevere un messaggio di multicast utilizzeremo il metodo *receivefromall* che restituisce il buffer con il messaggio, informazioni sulla sorgente e l'interfaccia di ingresso[1]. Mentre il metodo *acceptfromall* si mette in ascolto su tutti i socket TCP per l'apertura di una sessione, restituendo il socket della sessione TCP creata.

Activity

Questa è una struttura dati di tipo template per gestire interazioni tra thread. I thread vengono registrati in questa struttura dati con un chiave che ne permette l'identificazione, ad ogni thread viene assegnata una coda su cui possono mettersi in attesa altri thread sempre registrati. La struttura gestisce anche alcune informazioni riguardanti l'esecuzione del thread come tempo trascorso dall'avvio, tempo di inattività dall'ultimo checkpoint segnalato dal thread; queste informazioni possono essere usate per un sistema di ripristino di thread bloccati. I metodi principali sono la registrazione di un thread con *init* che richiede un valore per la chiave e una variabile di tipo pthread_t, il metodo *touch* che segna il cheackpoint per un thread e richiede la chiave del thread; il metodo *timedwait* invece blocca un thread sulla coda di un altro e specificando un valore temporale in secondi maggiore di zero, il thread resterà bloccato al massimo per quel intervallo, distinguendo se il risveglio è stato dovuto ad una signal o al timeout e infine il metodo *timedsignal* che effettua la signal risvegliando tutti i thread bloccati su una coda, specificando un intervallo temporale maggiore di zero secondi, l'operazione verrà effettuata dopo l'attesa dell'intervallo. Per recuperare le informazioni sui thread registrati possiamo usare i metodi *inactivefrom* e *timefromstart* che specificando un thread restituiscono l'informazione omonima.

Main_procs

Questa classe deriva dalla precedente, aggiungendo due metodi e definendo il tipo della chiave (enum T_name). I due metodi sono *printstatus* che, specificato un thread, ne stampa lo stato completo e *name* che traduce un valore chiave in una stringa corrispondente al nome del thread.

[1] Bisogna fare attenzione perchè si riferisce quelle inserite nella struttura, per noi sempre l'interfaccia virtuale, l'interfaccia fisica di ingresso verrà ricavata ad-hoc dall'indirizzo sorgente.

HA

Rappresenta la struttura dati che contiene le informazioni sulle Hello Adjacency. Abbiamo usato una struttura di tipo *map* con accesso di tipo hash sull'indirizzo che identifica l'altro capo dell'Hello Adjacency. Ogni elemento di questa struttura contiene le seguenti informazioni su una Hello Adjacency:

- LSR_ID del dispositivo remoto che ha inviato il messaggio di Hello Adjacency;
- tempo di interarrivo di un messaggio di Hello Adjacency;
- un numero sequenziale che cambia in caso di modifiche nella configurazione del LSR remoto;
- istante di arrivo dell'ultimo messaggio Hello Adjacency;
- ruolo che avrà il dispositivo corrente nella fase di inizializzazione della sessione, (PASSIVE, ACTIVE, NONE);
- stato della Hello Adjacency:

  NEW, indica che è una nuova Hello Adjacency;

  INIT, indica che è in esecuzione la procedura di inizializzazione sessione su questa Adjacency;

  ESTABLISHED, significa che la sessione stabilita con il LSR remoto utilizza il link identificato dalla Adjacency;

  BACKUP, indica che ci sono almeno due Adjacency con lo stesso LSR, di cui una già è usata per la sessione stabilita, l'Adjacency marcata come backup non viene usata per l'invio di messaggi di sessione;

  ERROR, si è verificato un errore durante la fase di inizializzazione, occorre riportare la Hello Adjacency allo stato di NEW per poi riavviare una nuova procedura di inizializzazione sessione.

In fig. 5.2 è rappresentato il diagramma a stati di una Hello Adjacency, le transizioni di stato verranno illustrate meglio durante la descrizione dei thread che compongono LDPL.

L'accesso ai metodi di questa classe è garantito in mutua esclusione grazie ad un semaforo interno; i metodi principali sono *add, del, search, exist, refresh, checktimeout*, che rispettivamente permettono di aggiungere, rimuovere una Hello

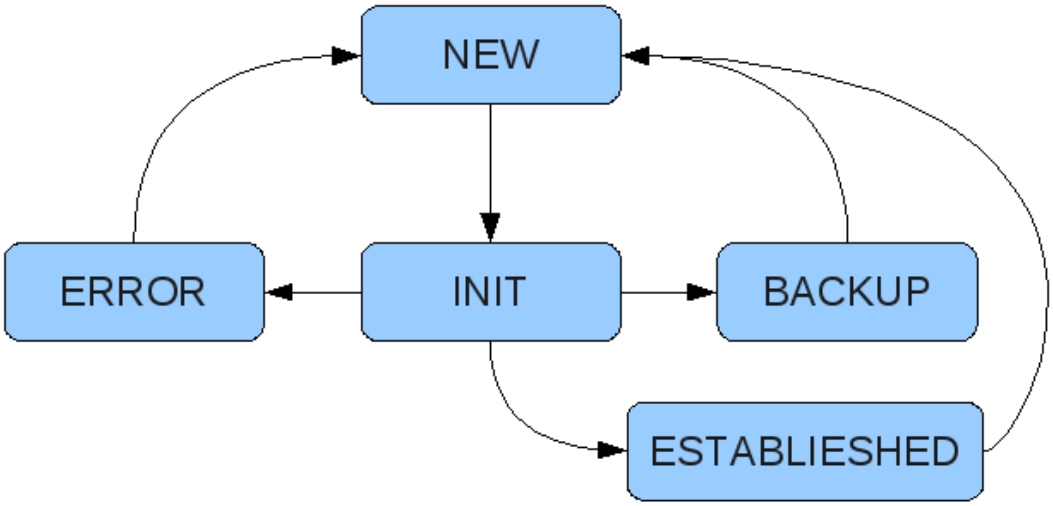


Figura 5.2: Diagramma degli stati di una Hello Adjacency.

Adjacency, cercare e restituire una Hello Adjacency della tabella, verificare l'esistenza di una Hello Adjacency, rinfrescare il timestamp e verificare se il timeout dall'ultimo messaggio è scaduto. Il metodo *setDelay* permette di settare il fattore di attesa, richiede un valore intero e imposta il timeout per una Adjacency pari al suo tempo di interarrivo per questo fattore; solitamente questo valore è 3. Altri tre metodi interessanti sono *searchbck* che restituisce l'indice di una Hello Adjacency di backup specificando il LSD_ID del LSR interessato, *fixerror* riporta tutte le Adjacency in stato di ERROR allo stato di NEW e infine *init* avvia una procedura di inizializzazione sessione per le Adjacency in stato NEW con ruolo ACTIVE.

Ci sono anche altri metodi per il settaggio e per il recupero dei valore dei campi di una Hello Adjacency, per maggiori dettagli rimandiamo alla documentazione Doxygen del software.

LDPS

Questa struttura dati contiene le informazioni relative alle sessioni LDP aperte, è realizzata con un vettore dinamico di elementi *LDPS_entry*, che contengono i seguenti dati di una sessione:

- LSR_ID del LSR remoto;
- i parametri della sessione scambiati in fase di inizializzazione;

- il socket della sessione;
- un campo booleano che che specifica se devo mandare, per il round[2] seguente, il messaggio di KeepAlive della sessione oppure no; perchè è già stato scambiato un altro messaggio che attesta comunque l'attività della sessione.
- timestamp riguardante l'ultimo KeepAlive o messaggio ricevuto sulla sessione;
- indirizzo del LSR remoto.

Una entry in questa struttura viene aggiunta in seguito ad una procedura di inizializzazione sessione andata a buon fine. L'accesso ai metodi avviene in mutua esclusione gestita da un semaforo interno. I metodi principali sono *newsession, delsession, refreshpar* per aggiungere, rimuovere una sessione e aggiornarne i parametri. Altri metodi come *touch* e *mark* agiscono rispettivamente aggiornando il timestamp e marcando una sessione per non inviare il KeepAlive nel round successivo. Il metodo *checktimeout* verifica l'esistenza di sessioni scadute e richiede in ingresso un riferimento all'oggetto HA per poter aggiornare opportunamente lo stato della rispettiva Hello Adjacency (da ESTABLISHED a NEW). Il metodo *sendka* effettua l'invio dei messaggi KeepAlive a tutte le sessioni non marcate e resetta la marcatura sulle sessioni già marcate, inoltre prendendo in ingresso un riferimento all'oggetto HA cambia lo stato della relativa Hello Adjacency in NEW se l'invio del messaggio non va a buon fine. Il metodo *receivefromallsession* permette di mettersi in ascolto su tutti i socket delle sessioni aperte in attesa di un messaggio, è possibile specificare un intervallo di tempo per rendere l'attesa di un messaggio finita; su questo metodo si baserà il thread *core* (5.2.2). Sono presenti ulteriori metodi per recuperare il valore dei campi delle singole sessioni. Per maggiori dettagli rimandiamo alla documentazione Doxygen del software.

MsgQueue

Questa struttura mantiene una lista di messaggi già inviati, in attesa di una risposta (come nel caso di Label Request e Label Mapping). I messaggi vengono inseriti specificando il loro message id come chiave per la loro identificazione; all'immissione viene assegnato un timestamp ad ogni messaggio in modo da poter eliminare i messaggi più vecchi per i quali non è arrivata alcuna risposta. I metodi

[2]I round sono scanditi dall'intervallo temporale di interinvio dei messaggi di KeepAlive di un dato LSR.

principali sono *addmsg, delmsg, getmsg* per aggiungere, rimuovere o recuperare i messaggi dalla lista, per tutti questi metodi bisogna specificare un message id. Infine il metodo *delholdest* elimina i messaggi più vecchi di un certo valore temporale specificato.

LabelBroker

Questa struttura gestisce le label tenendo conto di quelle libere e quelle occupate. È usata oltre che per le label anche per gestire gli indici della tabella NHLFE. La struttura è composta da due valori che indicano gli estremi superiore e inferiore dello spazio delle label utilizzato e un campo *current* che contiene il valore della prima label libera, quindi viene inizializzato col valore dell'estremo inferiore dello spazio delle label. Fa parte della struttura anche un vettore dinamico in cui vengono inserite le label non più utilizzate. Quando viene assegnata una label se il vettore delle label restituite è vuoto, assegno la label in current e lo incremento, se il vettore non è vuoto, assegno una di queste label e la elimino dal vettore. L'oggetto LabelBroker va inizializzato con i valori estremi dello spazio delle label, poi fornisce i seguenti metodi *getLabel* e *releaseLabel* che svolgono il compito omonimo; il metodo *check* che verifica la consistenza dei parametri di inizializzazione e *clear* che riporta allo stato iniziale l'oggetto.

In fase di inizializzazione, a quest'oggetto verrà assegnato un valore per il limite inferiore dello spazio delle label pari a 16, questo perchè nello standard MPLS le label inferiori a 16 hanno un significato particolare, ad esempio la label 3 è usata anche per indicare l'azione di POP.

Routes, LocalHosts, Arp e Nodes

Queste strutture dati si occupano dell'interfacciamento con gli handler Click per recuperare informazioni come le rotte del protocollo di routing, gli indirizzi ospitati, tabella ARP e la lista dei router della WMN. Ognuna di queste strutture va inizializzata con i relativi handler e fornisce il metodo *import* che effettua l'acquisizione delle informazioni. L'accesso ai dati viene fatto direttamente perchè dichiarati nella sezione pubblica della classe.

NHLFE_hash, FTN_hash, ILM_hash e FRRdb

Queste strutture mantengono, localmente al processo LDPL, le informazioni relative alle tabelle MPLS. NHLFE_hash, FTN_hash e ILM_hash sono realizzate con una struttura *map* con chiave rispettivamente l'indice, FEC e label di ingresso, questo per ottenere un accesso più veloce alle informazioni. FRRdb invece è realizzato con una struttura *vector* i cui elementi sono le entry della tabella fast reroute database. L'interfacciamento con le tabelle MPLS che risiedono in Click avviene nel seguente modo: quando cambia un valore di NHLFE_hash, FTN_hash e ILM_hash procedo con lo svuotamento delle tabelle MPLS in Click (metodo *clear*) e la loro totale riscrittura (metodo *sync*), questo perchè, dato che LDPL è l'unico processo autorizzato alla modifica delle label, non c'è alcun interesse ad effettuare operazioni di lettura o acquisizione di informazioni dalle tabelle MPLS in Click; per FRRdb, invece, prima di applicare delle modifiche si deve acquisire il contenuto della tabella in Click, questo perchè essenzialmente non sarà LDPL ad occuparsi dell'attivazione o disattivazione del fast reroute, questo compito verrà demandato a terze parti, come sistemi di link management. Inoltre devo seguire un certo ordine per la chiamata dei metodo sync delle tabelle MPLS, ogni volta che si aggiorna un valore della FTN e ILM con il metodo sync, devo successivamente richiamare in sequenza i metodi *import* e *sync* di FRRdb per caricare eventuali interventi di terze parti e riapplicarli ai valori di FTN e ILM che ho aggiornato. Le modifiche apportate da FRRdb non agiscono sulle tabelle FTN e ILM locali a LDPL ma solo su quelle in Click, questo perchè in fase di rilascio di un LSP è utile che le entry FTN e ILM siano impostate con i valori del LSP principale.

Scendendo più nel dettaglio delle strutture, FTN_hash è stata realizzata come una map di vector di elementi FTN_entry: la map è scandita sul valore del campo FEC, poi per la stessa FEC posso avere più entry della tabella FTN di cui soltanto uno attivo; in fase di sincronizzazione con la relativa tabella MPLS, vengono trasferite solo le entry attive e quelle non attive rimandano nella tabella locale ad LDPL, in questo modo non inseriamo informazioni superflue ai fini del forwarding nelle tabella FTN in Click. Un'altra particolarità la riscontriamo nel FRRdb, questo è realizzato con due strutture, una vector di elementi FRRdb_entry dove risiedono le informazioni definitive e una map di elementi FRRdb_entry dove risiedono le informazioni temporanee sulla creazione di una entry del database; questo perchè le informazioni per installare una entry non sono disponibili tutte

nello stesso momento, ma vengono ricostruite in fasi differenti del processo di installazione. Al fine della procedura di installazione le informazioni della tabella temporanea vengono recuperate in base all'identificativo assegnato ai messaggi di richiesta di installazione di un detour, completate e inserite nella tabella definitiva.

Tutte le strutture mettono a disposizione diversi metodi per l'accesso e la modifica delle informazioni nelle tabelle, per il loro uso rimandiamo alla relativa documentazione Doxygen. Di seguito riportiamo i campi degli elementi entry delle diverse tabelle:

NHLFE_entry: indice, azione, next-hop, interfaccia di uscita, un valore che indica la bontà di questa entry (fissato sempre al valore massimo, 10), flag che indica se la entry è attiva, prossimo indice (si veda sezione 4.1) e il path Srcr corrispondente.

FTN_entry: indice corrispondente all'azione NHLFE, flag di validità dell'entry. Il campo FEC è specificato come chiave della struttura map.

ILM_entry: indice corrispondente all'azione NHLFE, FEC di appartenenza. Il campo "label in ingresso" è specificato come chiave della struttura map.

FRRdb_entry: LSR_ID next-hop, LSR_ID next-next-hop, LSR_ID next-hop alternativo (detour), indice NHLFE quando il FRR non è attivo, indice NHLFE quando il FRR è attivo, flag che indica lo stato attivo o non attivo della entry.

ldppdu_t, ldpmsg_t e ldptlv_t

Queste strutture sono adibite alla formattazione e alla creazione di un messaggio LDPL. La struttura ldppdu_t corrisponde ad un contenitore di elementi ldpmsg_t che a loro volta sono un contenitore di elementi ldptlv_t. Nella struttura ldppdu_t vengono codificati i campi dell'header PDU, in più abbiamo un vettore dinamico di elementi ldpmsg_t. Nella struttura msgldp_t viene codificato l'header del messaggio e in un vettore dinamico vengono mantenuti i TLV del messaggio. La struttura ldptlv_t, che corrisponde la base di questa impostazione gerarchica, contiene i campi dell'header TLV e un buffer contenente i dati.

L'accesso ai campi di una struttura viene fatto direttamente. Ogni struttura ldppdu_t e ldpmsg_t mettono a disposizione i metodi *push_msg, pop_msg, push_TLV* e *pop_TLV* per l'aggiunta e la rimozione di messaggi e TLV. Le tre strutture mettono a disposizione anche il metodo *import* che, fornito un buffer,

ricostruisce la struttura dati: quando arriva un PDU sotto forma di buffer viene chiamato il metodo import di un oggetto ldppdu_t passandogli il buffer, questo identificherà le porzioni di buffer relative ai messaggi e chiamerà il metodo import per un oggetto ldpmsg_t passandogli la relativa porzione di buffer, procedendo allo stesso modo anche con i TLV. L'operazione inversa viene svolta dai metodi *pdubuff* e *msgbuff* che dalla struttura costruiscono il buffer che verrà inviato: l'operazione comincia chiamando il metodo pdubuff di un oggetto ldppdu_t questo creerà il buffer delle dimensioni specificate dal suo campo *lenght* e comincerà con il riempimento del buffer chiamando i metodi msgbuff dei propri oggetti ldpmsg_t.

Per la formulazione dei TLV vengono messe a disposizione una serie di funzioni, una per ogni tipo di TLV usato, che, fornendo i parametri opportuni, generano il TLV da inserire in un messaggio. Per maggiori dettagli rimandiamo alla documentazione Doxygen del software.

LDPConfig

questa è una semplice struttura su cui vendono trasferiti i dati acquisiti dal file di configurazione LDPL.conf. Posso accedere direttamente ai campi relativi ai parametri di configurazione, la struttura possiede un solo metodo, *parse*, che effettua l'acquisizione dei dati specificando il nome del file di configurazione.

5.2.2 Struttura dei thread

In questa sezione verrà descritto il comportamento dei singoli thread che compongono il software LSDL. I thread oltre ad appartenere ad una delle unità operazionali mostrate in fig. 4.5 , sono dinvisi in due categorie: *ciclici* e a *singola esecuzione*.

discover

È un thread ciclico appartenente al Discovery System; opera sulla struttura condivisa HA ed effettua le seguenti operazioni: all'avvio attende che il thread *init_passive_ruole_server* sia avviato, in seguito effettua l'invio di un messaggio di Hello su tutte le interfacce utilizzate dal sistema LSPL e l'operazione di verifica di Adjacency scadute. Infine il processo si mette in attesa per un tempo pari all'intervallo temporale di interinvio dei messaggi di discovey. L'attesa viene effettuata chiamando il metodo *timedwait* della struttura dati Main_procs specificando se

stesso come thread, in questo modo un altro thread può stimolare una esecuzione del thread discover fuori dall'intervallo temporale. Questo viene fatto di solito quando ricevo il primo messaggio di Hello da un altro LSR e voglio assicurarmi anche il LSR remoto avverta la mia presenza prima di cominciare la fase di inizializzazione della sessione.

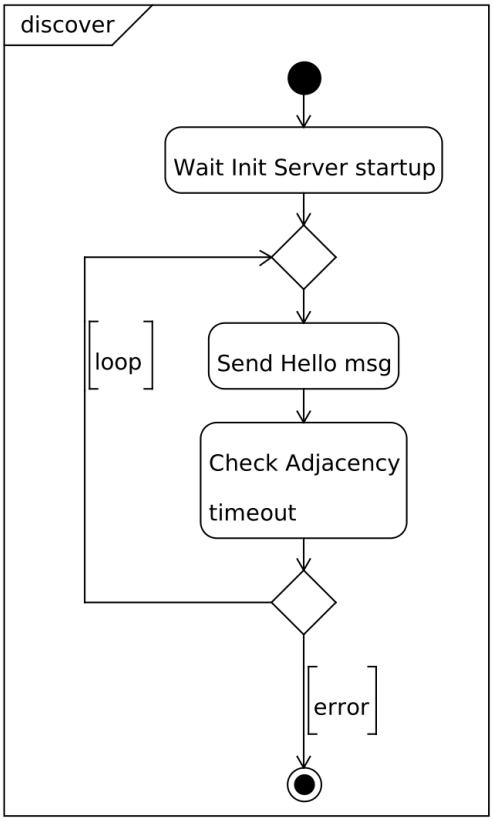


Figura 5.3: Diagramma di attività del thread discover.

helloprocess

È un thread ciclico appartenente al Discovery System che si occupa del processamento dei messaggi Hello ricevuti; opera sulla struttura condivisa HA e compie le seguenti operazioni: all'avvio attende che il thread *init_passive_ruole_server* sia in esecuzione, poi comincia il ciclo mettendosi in attesa dell'arrivo di un messaggio di Hello. Ricevuto un messaggio, verifica la consistenza del messaggio, se ben formato controlla se appartiene ad una nuova Adjacency (aggiunge la voce nella

HA) o se l'Adjacency era già esistente (aggiorna il timestamp), infine si rimette in attesa di un nuovo messaggio.

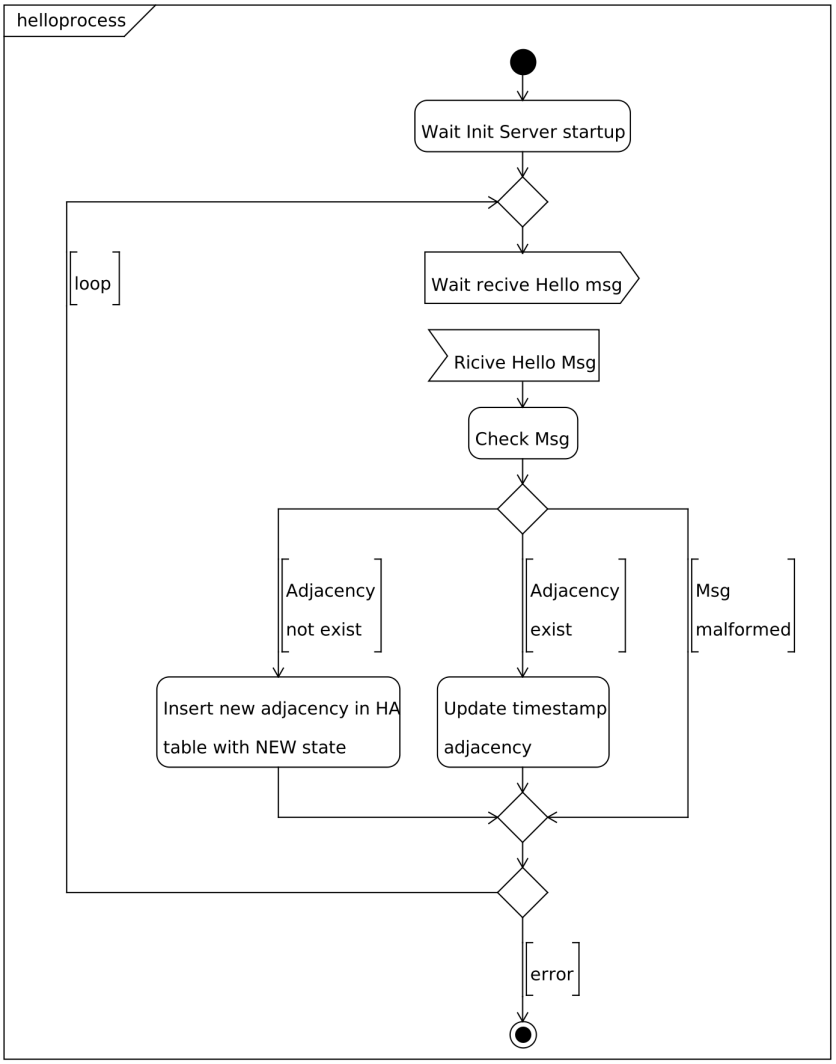


Figura 5.4: Diagramma di attività del thread helloprocess.

init_passive_role_server

È un thread ciclico appartenente al Session System che, opera da server per l'apertura di connessioni TCP. Interagisce con la struttura HA per verificare se la richiesta di creazione di una nuova sessione è in contrapposizione con lo stato dell'Adjacency. Il thread compie le seguenti operazioni, prima di tutto verifica se la richiesta di apertura di un connessione TCP avvenga su un Adjacency esistente

e con stato NEW o BACKUP e ruolo PASSIVE, altrimenti scarta la richiesta; se può accettare la richiesta aggiorna lo stati dell'Adjacency a INIT ad indicare che è in corso la fase di inizializzazione, infine avvia il thread *init_passive_role_proc* passandogli il socket stabilito per la connessione TCP.

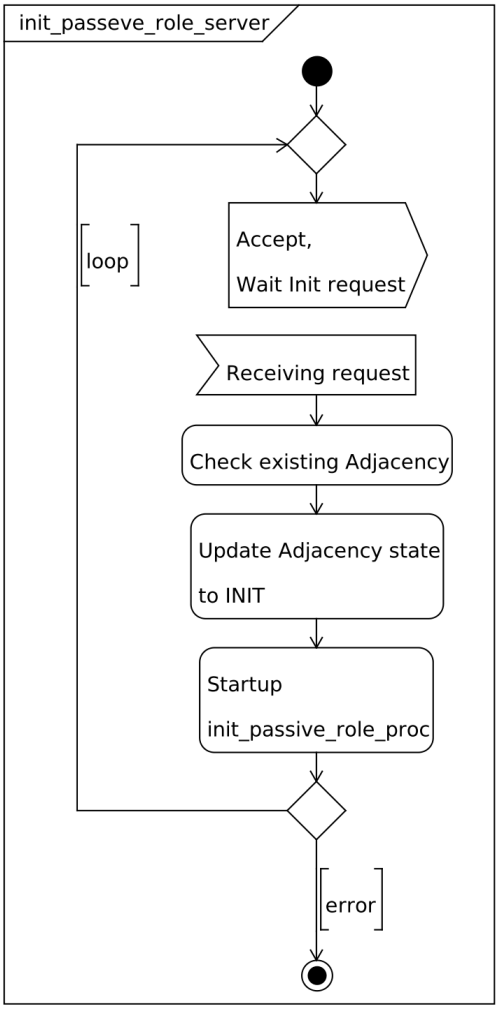


Figura 5.5: Diagramma di attività del thread init_passive_role_server.

init_passive_role_proc

È un thread a singola esecuzione appartenente al Session System; svolge le operazioni di un peer LDP con ruolo passivo durante la fase di inizializzazione. Agisce sulle strutture dati HA e LDPS e compie le seguenti operazioni: si mette in attesa del messaggio di apertura della fase di inizializzazione (LDP Initializa-

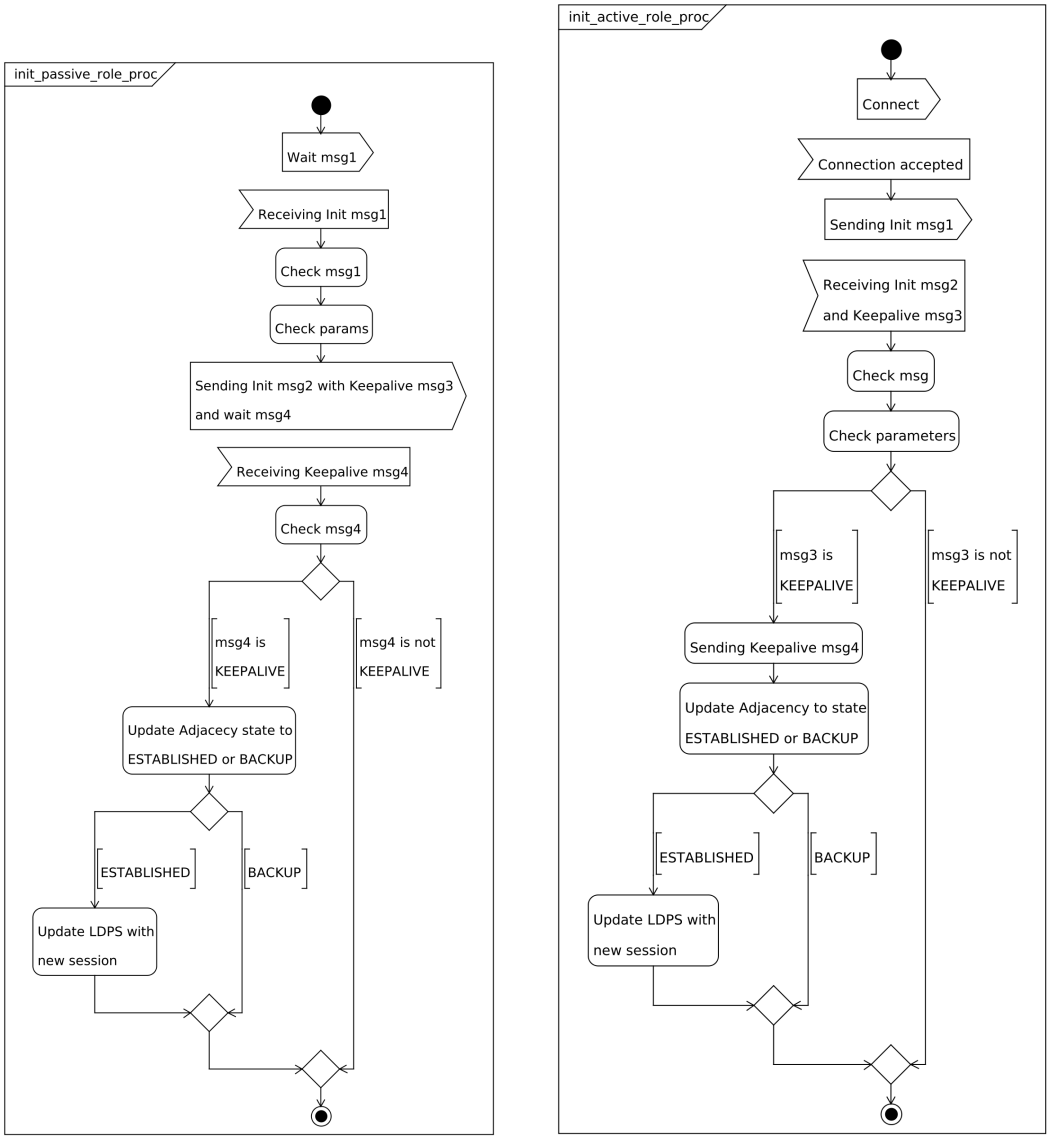
tion) sul socket ricevuto da init_passive_role_server, ricevuto il messaggio verifica che sia conforme e controlla i parametri trasportati; a questo punto risponde inviando un PDU contenente i messaggi LDP Initialization con i propri parametri e il messaggio di KeepAlive per notificare l'accettazione dei parametri ricevuti. Il thread attende l'arrivo del messaggio di KeepAlive del peer remoto prima di proseguire con l'aggiornamento dello stato dell'Adjacency a ESTABLISHED o BACKUP e l'aggiornamento della tabella LDPS con una nuova sessione se non è una Adjacency di backup. Tutte le operazioni riguardanti la ricezione di messaggi sono temporizzate, se allo scadere del timer il messaggio non è arrivato, lo stato dell'Adjacency viene settato su ERROR e la procedura di inizializzazione si interrompe.

init_active_role_proc

È un thread a singola esecuzione appartenente al Session System; svolge le operazioni di un LDP peer con ruolo attivo durante la fase di inizializzazione di una sessione. Agisce sulle strutture dati HA e LDPS e compie le seguenti operazioni: invia una richiesta di apertura di una connessione TCP. Se l'apertura della connessione va a buon fine prosegue con la procedura di inizializzazione inviando il primo messaggio di LDP Initialization. Poi attende l'arrivo dei messaggi di LDP Initialization e KeepAlive del peer con ruolo passivo e verifica la struttura dei messaggi, i parametri ricevuti e la conferma di accettazione dei parametri precedentemente inviati (KeepAlive). Se i parametri ricevuti sono accettabili il thread risponde inviando un messaggio KeepAlive, sancendo la fine della procedura di inizializzazione. Prima di terminare aggiorna lo stato della Adjacency a ESTABLISHED o BACKUP e nel caso in cui non sono su una Adjacency di backup aggiorno la tabella LDPS con la nuova sessione creata. Tutte le operazioni riguardanti la ricezione di messaggi sono temporizzate, se allo scadere del timer il messaggio non è arrivato, lo stato dell'Adjacency viene settato su ERROR e la procedura di inizializzazione si interrompe.

ka

È un thread ciclico appartenente al Session System. Opera sulle strutture dati condivise HA e LDPS. Si attiva allo scadere di un timer che scandisce i round di invio dei KeepAlive sulle sessioni. Opera sulle strutture condivise HA e LDPS



(a) (b)

Figura 5.6: Diagramma di attività dei thread init_passive_role_proc e init_active_role_proc.

e compie le seguenti azioni: verifica se ci sono sessioni scadute ed eventualmente le rimuove; invia un messaggio KeepAlive su tutte le sessioni marcate per l'invio di KeepAlive (vedi struttura LDPS); corregge le inconsistenze tra HA e LDPS e riporta le Adjacency in stato di errore allo stato NEW. Infine il thread avvia il thread init_active_role_proc per ogni Adjacency in stato NEW e ruolo ACTIVE fornendo l'indirizzo del peer con cui deve aprire la connessione TCP.

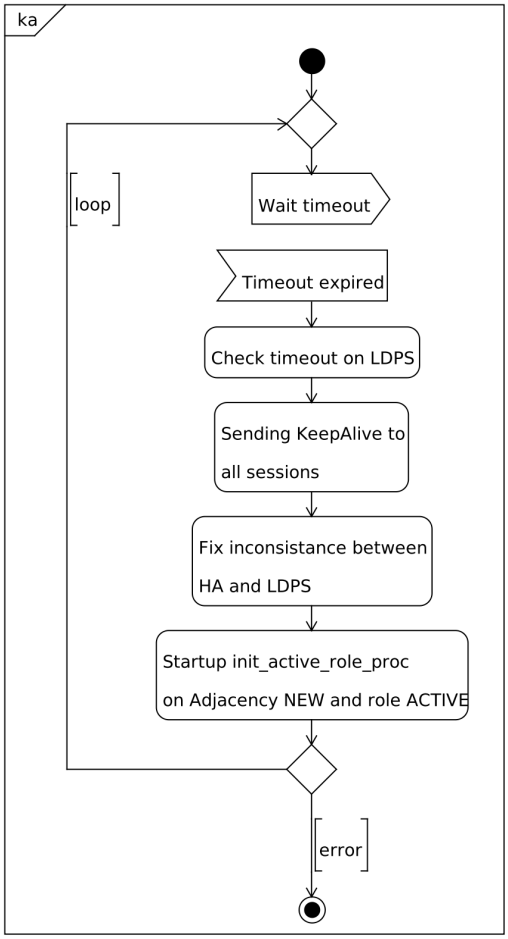


Figura 5.7: Diagramma di attività del thread ka.

controller

È un thread ciclico appartenente al Control System. Opera sulle strutture dati condivise HA, LDPS, FTN_hash, ILM_hash, NHLFE_hash e FRRdb. Si attiva all'arrivo di un messaggio di controllo dal pannello di controllo LDPL. All'arrivo

di un messaggio il thread controlla che sia ben formattato, infine lo processa basandosi sul tipo del messaggio, di seguito sono riportati i messaggi di controllo e l'azione che generano:

STATUS, il LSR risponde al pannello di controllo inviando la versione del software e la lista dei nodi della rete WMN;

TABLEHA, il LSR risponde al pannello di controllo inviando il contenuto della tabella HA;

TABLELDPS, il LSR risponde al pannello di controllo inviando il contenuto della tabella LDPS;

LABELREQUEST, avvia la procedura di installazione di un nuovo LSP dopo aver controllato se la FEC specificata nel messaggio sia raggiungibile;

LSPON/LSPOFF, attiva o disattiva al forwarding un LSP;

TABLEMPLS, il LSR risponde al pannello di controllo inviando il contenuto delle tabelle MPLS;

CLEARTABLEMPLS, il LSR cancella il contenuto delle tabelle MPLS;

LABELRELEASE, il LSR avvia la procedura di cancellazione di un LSP;

FRRON/FRROFF, il LSR attiva o disattiva una entry della tabella FRRdb;

ECHO, il LSR invia un messaggio di risposta al LSR che ha effettuato la richiesta di ECHO;

core

È un thread ciclico appartenente al Core System. Opera sulle strutture dati condivise LDPS, FTN_hash, ILM_hash, NHLFE_hash e FRRdb. Si attiva all'arrivo di un messaggio LDP da una delle sessioni attive. All'arrivo di un messaggio, viene verificata la corretta formattazione e infine viene avviata la procedura richiesta. Il thread non rimane permanentemente in attesa di un messaggio, se non arriva nessun messaggio entro un certo intervallo di tempo (configurabile da file di configurazione), questo riprende la sua esecuzione principalmente per poter aggiornare la lista di socket su cui deve ascoltare l'arrivo di un messaggio. I messaggi gestiti dal thread sono:

_KEEPALIVE, effettua l'aggiornamento del timestamp di una sessione;

_LREQ, ricevuto un messaggio di LDP Label Request, se il LSR non è l'egress-LSR per la FEC specificata, continua la propagazione della richiesta lungo la rotta stabilita dall'algoritmo di routing;

_LMAP, ricevuto un messaggio di LDP Label Mapping, il LSR aggiorna le tabelle MPLS, richiede una nuova label e, se non è l'ingress-LSR, la propaga al LSR precedente in un altro messaggio LDP Label Mapping. La ricezione di questo messaggio avvia anche la procedura di installazione di un detour se l'insieme dei neighbor in comune con il LSR e il next-next-hop non è vuoto.

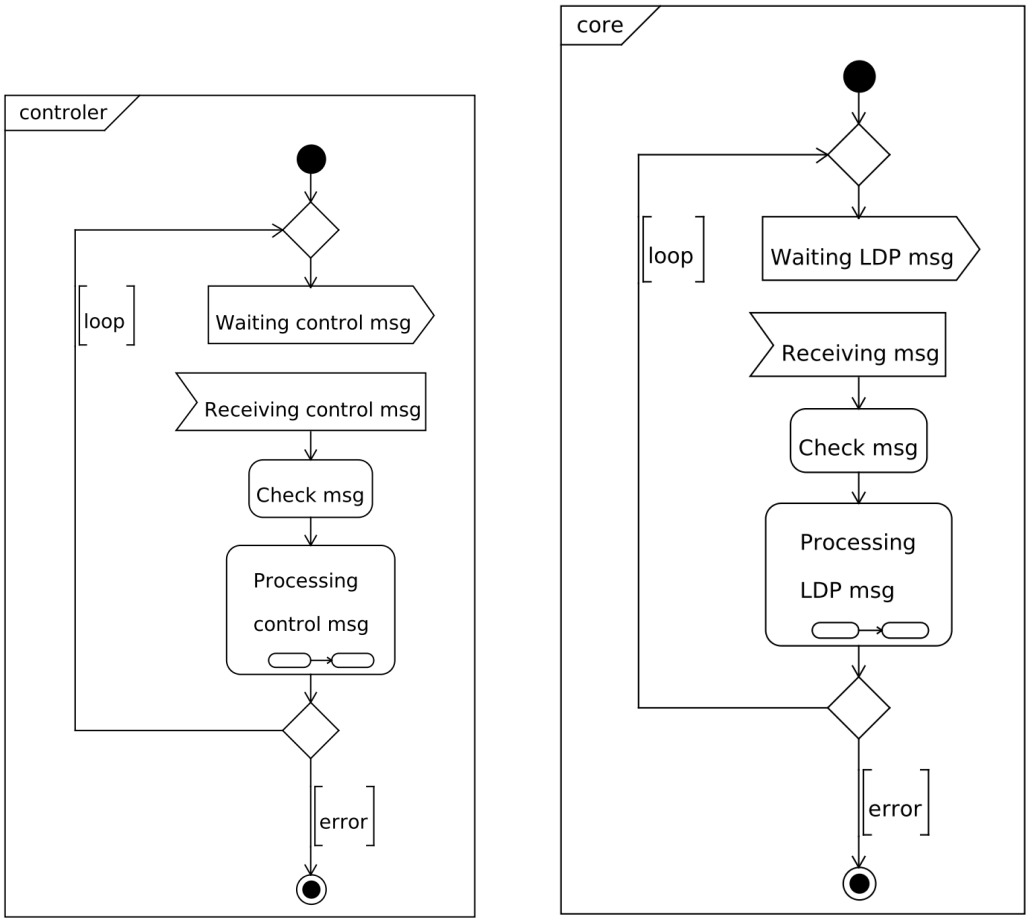
_LRELEASE, ricevuto il messaggio LDP Label Release, il LSR aggiorna le tabelle MPLS e propaga il messaggio fin quando il LSR non corrisponde all'egress-LSR;

_LDREQ, la ricezione del messaggio Label Detour Request causa l'aggiornamento delle tabelle MPLS e l'invio come risposta del messaggio di Label Detour Mapping;

_LDMAP, la ricezione del messaggio Label Detour Mapping completa l'aggiornamento delle tabelle MPLS cominciato con la ricezione del messaggio LDP Label Mapping che ha avviato la procedura di installazione di questo detour.

pathguardinan

È un thread ciclico appartenente al Path Analizer System. Opera sulle strutture dati condivise FTN_hash, NHLFE_hash. il thread compie ciclicamente le seguenti operazioni e ogni esecuzione è scandita da un time configurabile da file di configurazione: invia una richiesta di ECHO al Control System dell'egress-LSR di ogni LSP che parte dal LSR in questione. La richiesta di ECHO (destinata alla porta 656) non viaggia sul LSP ma sul tradizionale routing IP, stimolando il calcolo della rotta da parte del protocollo di routing. Il thread acquisisce le nuove informazioni sulla rotta e le confronta con quelle associate al LSP in fase di installazione; se le rotte differiscono allora avvia una nuova procedura di installazione di un LSP per la medesima FEC. Istallato il nuovo LSP setta opportunamente i flag di validità nelle rispettive entry della FTN_hash. Se dovesse verificarsi un



(a) (b)

Figura 5.8: Diagramma di attività dei thread controller e core.

nuovo cambiamento del percorso di routing che corrisponde ad un LSP già installato, il thread si limita ad operare soltanto sui flag di validità delle rispettive entry FTN_hash.

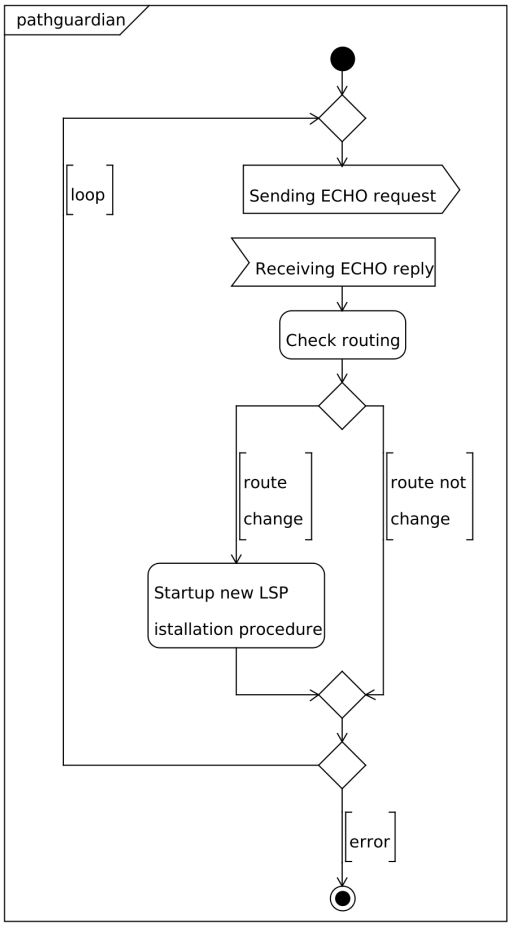


Figura 5.9: Diagramma di attività del thread pathguardian.

5.2.3 Struttura del Main

Il Main del software LDPL compie tutte le operazioni di inizializzazione, cominciando dall'acquisizione dei dati di configurazione dal file LDPL.conf; sulla base di questi dati effettua l'inizializzazione delle strutture dati condivise, resetta lo stato delle tabelle MPLS contenute in Click in modo da poter partire da uno stato consistente. Dopo aver inizializzato le strutture dati procede con l'avvio dei thread ciclici e li registra alla struttura Main_procs che ne gestisce la sincronizzazione e lo stato. Alla fine l'esecuzione del Main entra in pausa attendendo la terminazione forzata dal sistema operativo.

Per avviare il software è sufficiente eseguire il seguente comando dal dispositivo interessato:

```
LDPL [file configurazione]
```

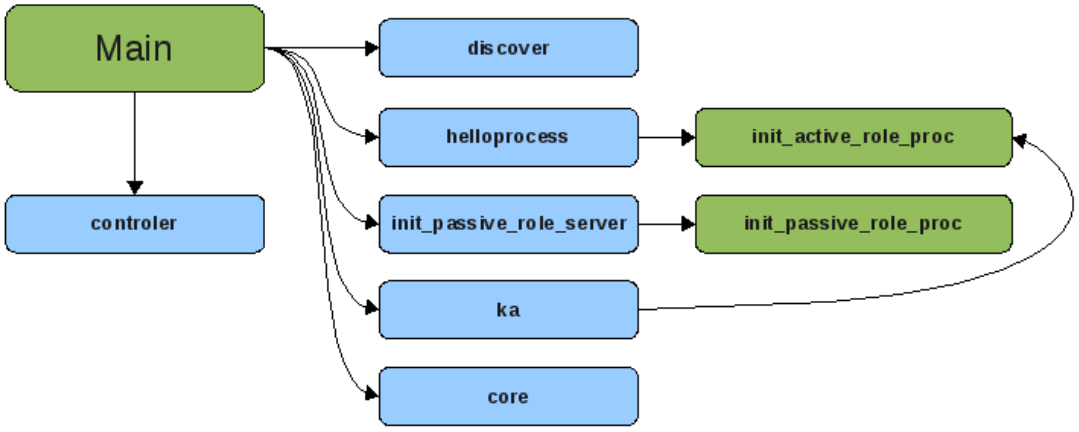


Figura 5.10: Grafico che rappresenta la gerarchia di avvio dei thread partendo dal processo Main: in azzurro in thread ciclici e in verde i thread a singola esecuzione.

5.2.4 File di configurazione LDPL.conf

Questo file contiene tutti i campi configurabili del sistema LDPL; segue la classica sintassi dei file conf unix dove basta specificare campo e valore; è possibile introdurre dei commenti facendo precedere la riga corrispondente dal carattere #. I campi configurabili sono i seguenti:

LSR_ID: valore intero che identifica l'id del LSR;

LABEL_ID: valore intero che identifica l'id dello spazio delle label (valore default 0);

MULTI_ADDR: indirizzo di multicast dei messaggi di discovery;

PORT: porta UDP/TCP utilizzata per i messaggi LDP (default 646);

MAXPDU: dimensione massima in byte di un PDU;

HELLOTIME: valore intero che indica l'intervallo temporale di interinvio dei messaggi LDP Hello (in secondi);

TTL: valore del campo TTL dei messaggi UDP di discovery;

INADDR: lista delle interfacce utilizzate dal protocollo, la lista va specificata come una lista di indirizzi separati da una spaziatura;

MULTI_LOOP: valore booleano (0/1) per attivare la modalità di loopback dei messaggi di multicast, utile in alcune condizioni di debug;

KEEPALIVETIME: valore intero che indica i secondi dell'intervallo di interinvio dei messaggi LDP KeepAlive;

BACKLOG: valore intero che indica la lunghezza della coda di backlog del server init_passive_role_server;

PVL: campo che specifica la lunghezza massima di un TLV path vector, questo parametro non viene usato nella nostra implementazione ma fa parte dei parametri scambiati dalla procedura LDP standard di inizializzazione sessione; è stato comunque mantenuto per possibili impieghi futuri;

CONTROLLER_PORT: valore intero che rappresenta la porta di ascolto del thread controller;

DELAYFACTOR: valore intero che corrisponde al fattore moltiplicativo per calcolare il timeout di una sessione o di una Adjacency;

CORETIME: valore intero che indica i secondi di un intervallo di inattività del thread core;

VERSION: versione del software LDPL;

PHP: valore booleano (0/1) che indica se la funzione di PHP è attiva;

DETOUR: valore booleano (0/1) che indica se durante l'installazione di un LSP devo avviare anche la procedura di installazione di un detour;

CHASE_ROUTING_CHANGE: valore booleano (0/1) che indica se avviare o non avviare il sistema di "inseguimento" delle rotte del protocollo di routing (quindi se avviare il thread pathguardian);

Di seguito un esempio di file di configurazione LDPL.conf:

```
LSR_ID: 1
LABEL_ID: 0
ULTI_ADDR: 224.0.0.3
PORT: 646
MAXPDU: 512
HELLOTIME: 5
TTL: 1
INADDR: 5.20.186.236
MULTI_LOOP: 0
KEEPALIVETIME: 15
BACKLOG: 10
PVL: 5
CONTROLLER_PORT: 656
DELAYFACTOR: 3
CORETIME: 2
VERSION: 1
PHP: 1
DETOUR: 1
CHASE_ROUTING_CHANGE: 1
```

5.3 Elemento FRRdb

Per supportare la funzione di FRR abbiamo dovuto realizzare un nuovo elemento che mantenesse le informazioni del fast reroute database e che fornisse dei metodi per applicare le commutazioni di rotte. L'elemento che abbiamo realizzato si chiama *FRRdb* e non prevede porte di ingresso e/o uscita di pacchetti,

infatti questo elemento non processa pacchetti ma agisce sulle tabelle mantenute dagli elementi FTN e ILM. L'elemento, al suo interno, mantiene due riferimenti all'istanza dell'elemento FTN e ILM e quando viene attivato un detour agisce sull'entry relativa modificando l'indice dell'azione corrispondente, cioè cambia l'azione svolta per lo stesso evento. Questa azione, che corrisponde ad una entry della tabella nell'elemento NHLFE, dovrà già essere stata opportunamente installata dal sistema di distribuzione delle label. Abbiamo realizzato la tabella usando un vettore dinamico di elementi FRRdbentry, ognuno dei quali contiene le seguenti informazioni: id del next-hop, id del next-next-hop, id del next-hop del detour, indice della NHLFE relativo all'azione base, indice della NHLFE relativo all'azione di immissione nel detour e flag di stato dell'entry (attiva/disattiva).

L'elemento mette a disposizione gli appositi handler che permettono dall'esterno di installare, cancellare e attivare delle entry:

entry_add_handler , effettuando una scrittura sull'handler e specificando i valori della entry separati da uno spazio, permette di aggiungerla;

entry_remove_handler , effettuando una scrittura sull'handler e specificando i valori della entry separati da uno spazio, permette di rimuoverla;

table_clear_handler , effettuando una scrittura sull'handler cancella la tabella FRRdb;

entry_activate_handler , questo handler permette di svolgere due azioni specificando le informazioni di input in modo diverso: posso attivare una singola entry specificando tutti i campi della entry stessa oppure attivare tutte le entry che hanno lo stesso valore di next-hop specificando soltanto l'id del next-hop; questo perchè potrei voler attivare tutti i detour che proteggono un certo next-hop dopo aver rilevato la sua irragiungibilità;

entry_deactivate_handler , funziona allo stesso modo dell'handler precedente solo che disattiva l'utilizzo del detour;

entries_dump_handler , questo handler viene acceduto in lettura e fornisce il contenuto della tabella FRRdb.

5.4 Pannello di controllo

Abbiamo realizzato un software che permetta il controllo dei singoli LSR da remoto. Il software si basa su un sistema di scambio di messaggi su protocollo UDP, permette di interfacciarci con ogni singolo LSR usando un paradigma di tipo "domanda e risposta". Le operazioni che si possono effettuare da questo software sono:

1. la visualizzazione dello stato del processo LDPL;
2. la visualizzazione della tabella delle Hello Adjacency;
3. la visualizzazione della tabella delle sessioni LDP attive;
4. l'installazione di un nuovo LSP specificando un'appropriata FEC;
5. la possibilità di specificare se usare o non un LSP installato per il forwarding di pacchetti;
6. il ritiro delle label di un LSP installato;
7. la visualizzazione delle tabelle MPLS del LSR;
8. la cancellazione forzata delle tabelle MPLS del LSR;
9. attivazione e disattivazione di un determinato detour per un LSP specificato.

Tutti questi comandi vanno eseguiti dopo aver selezionato un LSR target a cui verrà inviata la richiesta; questo risponderà, in base al comando selezionato, con un messaggio di risposta appropriato.

Abbiamo fatto uso delle librerie FLTK v.1.1.9 [23, 24] per la costruzione della GUI [3].

I comandi sono comunicati attraverso un messaggio UDP alla porta 656, che corrisponde alla porta di default su cui i processi LSPL si pongono in ascolto (questo valore è configurabile dal file LDPL.conf e dalla GUI stessa). La struttura del messaggio è molto semplice: i primi 4 byte corispondono al comando che si vuole eseguire e di seguito sono codificati i parametri, per i comandi che li richiedono.

I comandi che richiedono degli ulteriori parametri visualizzano, prima dell'invio del messaggio, un pannello che ne semplifica l'immissione. Questi comandi sono

[3] Grafic User Interface.

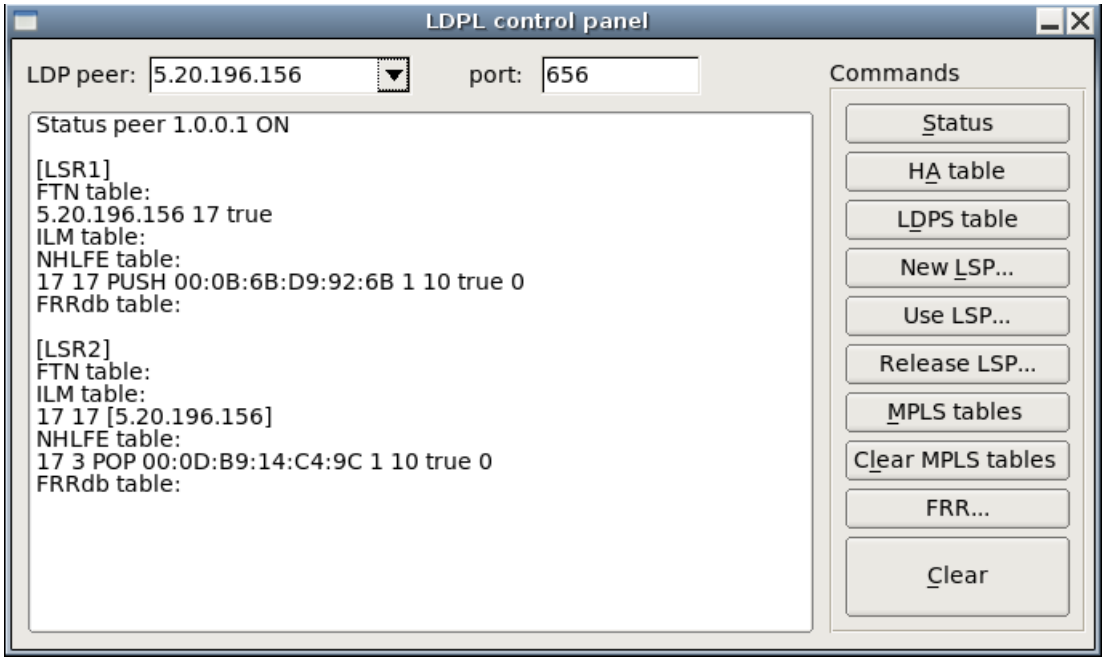


Figura 5.11: Lo screenshot del software LDPL control panel con l'output di uno dei test effettuati nel cap. 6.

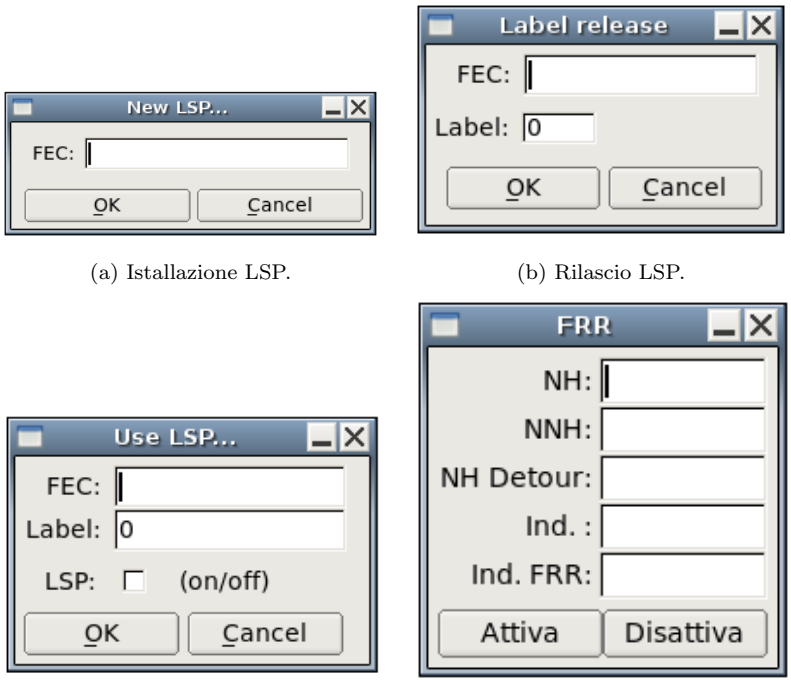
contrassegnati con la classica notazione dei punti di sospensione. I pannelli per l'immissione dei parametri verranno descritti dettagliatamente in seguito.

I messaggi di risposta, inviati dai LSR, contengono semplicemente una stringa da visualizzare.

Il pannello principale mostrato in fig. 5.11 ci permette di specificare il *LDP peer target* attraverso il suo indirizzo e la porta da utilizzare, sulla destra sono riportati i comandi possibili, al centro invece abbiamo messo un buffer display che visualizza le risposte ai comandi e permette la selezione e la copia del testo ottenuto; il comando di visualizzazione delle tabelle MPLS effettua una visualizzazione incrementale dei messaggi di risposta in modo da poter confrontare le tabelle di più LSR; il pulsante *Clear* non corrisponde a nessun comando ma effettua la cancellazione del buffer display.

Il comando *Status* oltre a visualizzare lo stato del LSR target, effettua l'acquisizione automatica degli indirizzi dei nodi che compongono la WMN, questa lista è accedibile dal menù a tendina accanto al campo LDP peer; questa soluzione agevola le operazioni di comando, bisogna far attenzione però che in questa lista sono anche presenti gli host attivi che non hanno in esecuzione un'istanza di

LDPL.



(a) Istallazione LSP.

(b) Rilascio LSP.

(c) Utilizzo LSP.

(d) Attivazione/disattivazione FRR.

Figura 5.12: Pannelli di inserimento parametri per i relativi comandi.

I comandi *HA table*, *LDPS table* e *MPLS table* effettuano la visualizzazione delle rispettive tabelle. Il comando *New LSP...* avvia la procedura di installazione di un nuovo LSP dal LSR target, come parametri richiede l'indirizzo di destinazione, corrispondente con la FEC, immesso nel relativo pannello (fig. 5.12(a)). Il comando *Release LSP...* avvia la procedura di rilascio di un LSP precedentemente creato, il comando ha efficacia solamente sull'ingress-LSR del LSP, e bisogna specificare la entry corrispondente al LSP nella tabella FTN specificando FEC e indice (fig. 5.12(b)). Il comando *Use LSP...* attiva o disattiva un LSP al forwarding, opera sul flag di validità delle entry della FTN quindi occorre specificare FEC, indice e stato del flag (fig. 5.12(c)). Il comando *FRR...* attiva o disattiva una entry del fast reroute database, occorre specificare i campi della relativa entry: next-hop, next-next-hop, detour next-hop, indice NHLFE base e indice NHLFE FRR, infine attivarla o disattivarla attraverso l'apposito pulsante (fig. 5.12(d)).

Capitolo 6

Test

In questo capitolo verrà presentato prima lo scenario adottato per i test seguito dalla spiegazione di come sono stati condotti i test (6.1) e infine verranno illustrati i risultati (6.2 e 6.3).

6.1 Scenario dei test

Per i test sono stati adottati i dispositivi descritti nella sezione 3.1; in ambito di laboratorio si è preferito configurare le interfacce wireless in modalità 802.11a, per risentire il meno possibile delle interferenze causate dall'elevato numero di reti wireless rilevate in modalità 802.11b/g. I dispositivi sono stati disposti all'interno del RedLab del dipartimento di ingegneria informatica dell'Università di Pisa.

Attraverso un'oculata assegnazione dei canali, abbiamo realizzato la topologia in figura 6.1, dove i link con lo stesso colore indicano lo stesso canale di funzionamento. Nella rappresentazione grafica dei LSR (figura 6.1) è stata inserita anche l'interfaccia usata da un link, distinguendo tra interfaccia wireless (cerchio con il numero dell'interfaccia) e interfaccia ethernet (quadrato con il numero dell'interfaccia).

Sono stati inseriti due client collegati ad una interfaccia ethernet di LSR1 e di LSR4, il client 1.20.186.236 è anche l'host su cui opera il software di controllo LDPL.

Sono stati condotti una serie di test di validazione del protocollo e dei test per la misurazione di prestazioni. I test si concentrano sui casi che abbiamo ritenuto

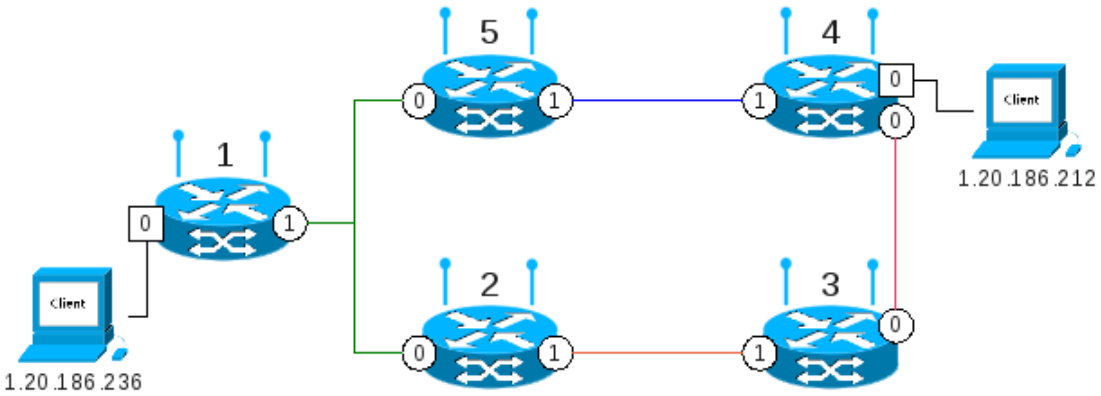


Figura 6.1: Topologia del testbed.

più significativi entro i limiti dell'ambiente di test; a causa delle numero ridotto di nodi a disposizione non è stato possibile eseguire test più complessi.

6.2 Test di validazione

La serie di test di validazione consiste in:

1. installazione di un LSP
   (a) senza detour,
   (b) con detour;
2. disinstallazione di un LSP precedentemente creato
   (a) senza detour,

	IP host	ath0 MAC	ath1 MAC
LSR1	5.20.186.236	00:0B:6B:D9:54:24	00:0B:6B:D9:92:7B
LSR2	5.20.196.156	00:0B:6B:D9:92:6B	00:0B:6B:D9:92:6C
LSR3	5.21.208.48	00:0B:6B:D9:92:6F	00:0B:6B:D9:92:17
LSR4	5.20.186.212	00:0B:6B:D9:92:0A	00:0B:6B:D9:92:80
LSR5	5.20.186.132	00:0B:6B:D9:92:7A	00:0B:6B:D9:92:8F

Tabella 6.1: Tabella contenente gli indirizzi IP identificativi dei LSR e i relativi indirizzi MAC delle interfacce wireless per poter avere un riscontro sui valori dei campi next-hop.

(b) con detour;

3. disattivazione/attivazione al transito di un LSP precedentemente creato;
4. variazione del percorso di un LSP precedentemente creato, dovuta alla variazione di rotta del protocollo di routing:
   (a) nel caso in cui un LSR del LSP non sia più raggiungibile causa guasto,
   (b) nel caso in cui la variazione delle metriche causa il cambiamento della rotta assegnata da Srcr;
5. abilitazione/disabilitazione all'utilizzo di un detour;

La serie di test condotti sulle prestazioni sono:

1. tempo medio di installazione di un LSP;
2. tempo medio di aggiornamento di un LSP in seguito al guasto di un LSR lungo il LSP stesso;

Da questo punto in poi i test di validazione saranno indicati dalla lettera *V* seguita dal numero del test (es. V.1a) e i test di prestazioni dalla lettera *P* seguita dal numero del test; inoltre in tutti i test va considerata la topologia in figura 6.1 tranne nei casi in cui vengono esplicitamente dichiarate delle variazioni allo scenario all'inizio del test.

La risposta alle azioni richieste nei test corrisponde al output visualizzato dal programma di controllo LDPL; le righe delle tabelle MPLS sono da interpretare come segue:

```
FTN table:
 [FEC] [NHLFE_index] [validity_flag]
ILM table:
 [label_in] [NHLFE_index] [FEC]
NHLFE table:
 [NHLFE_index] [label_out] [action] [next-hop] [out_interface]
                           [validity_flag] [next_NHLFE_index]
FRRdb:
 [LSRid_next-hop] [LSRid_next-next-hop] [LSRid_detour-next-hop]
                  [NHLFE_index] [detour_NHLFE_index] [active]
```

Test V.1a - Per testare la validità nell'installazione di un LSP verranno installati dei LSP e volta per volta inserite le tabelle MPLS ottenute sui LSR interessati.

Istallazione di un LSP da LSP1 con FEC 5.20.196.156:

```
[LSR1]
FTN table:
5.20.196.156 16 true
ILM table:
NHLFE table:
16 16 PUSH 00:0B:6B:D9:92:6B 1 10 true 0
FRRdb table:

[LSR2]
FTN table:
ILM table:
16 16 [5.20.196.156]
NHLFE table:
16 3 POP 00:0D:B9:14:C4:9C 1 10 true 0
FRRdb table:
```

Installazione di un LSP da LSR1 con FEC 5.21.208.48:

```
[LSR1]
FTN table:
5.21.208.48 17 true
5.20.196.156 16 true
ILM table:
NHLFE table:
16 16 PUSH 00:0B:6B:D9:92:6B 1 10 true 0
17 17 PUSH 00:0B:6B:D9:92:6B 1 10 true 0
FRRdb table:

[LSR2]
FTN table:
ILM table:
16 16 [5.20.196.156]
17 17 [5.21.208.48]
NHLFE table:
16 3 POP 00:0D:B9:14:C4:9C 1 10 true 0
17 3 POP 00:0B:6B:D9:92:17 1 10 true 0
FRRdb table:

[LSR3]
FTN table:
ILM table:
NHLFE table:
FRRdb table:
```

Installazione di un LSP da LSR1 con FEC 1.20.186.212:

```
[LSR1]
FTN table:
5.21.208.48 17 true
5.20.196.156 16 true
1.20.186.212 18 true
ILM table:
NHLFE table:
```

```
16 16 PUSH 00:0B:6B:D9:92:6B 1 10 true 0
17 17 PUSH 00:0B:6B:D9:92:6B 1 10 true 0
18 16 PUSH 00:0B:6B:D9:92:7A 1 10 true 0
FRRdb table:

[LSR2]
FTN table:
ILM table:
16 16 [5.20.196.156]
17 17 [5.21.208.48]
NHLFE table:
16 3 POP 00:0D:B9:14:C4:9C 1 10 true 0
17 3 POP 00:0B:6B:D9:92:17 1 10 true 0
FRRdb table:

[LSR3]
FTN table:
ILM table:
NHLFE table:
FRRdb table:

[LSR4]
FTN table:
ILM table:
NHLFE table:
FRRdb table:

[LSR5]
FTN table:
ILM table:
16 16 [1.20.186.212]
NHLFE table:
16 3 POP 00:0B:6B:D9:92:80 1 10 true 0
FRRdb table:
```

Tutti quanti i test hanno restituito i valori attesi. In particolare si noti che nella prima installazione la configurazione del PHP non avviene perchè priva di senso, il next-hop di LSR2 viene configurato con un MAC address appartenente ad una propria interfaccia ethernet ad indicare se stesso come destinazione. L'assegnazione delle label e degli indici come si può notare comincia dal valore 16, questo perchè, come spiegato già nella descrizione della struttura LabelBroker, i valori inferiori a 16 sono stati riservati a particolari scopi. Nella seconda installazione viene configurato opportunamente il forwarding sfruttando la funzione di PHP e il LSR presenta nella relativa entrata della NHLFE l'azione di POP con un valore di next-hop corrispondente a LSR3 (tab. 6.1).

Test V.1b - In questo test verrà validata la capacità di installare un LSP con

relativi detour. A causa dei limiti dovuti all'esiguo numero di dispositivi, il test può essere effettuato solamente installando un LSP che congiunge i vertici opposti del quadrato formato da LSR2, LSR3, LSR4 e LSR5. Installeremo un LSP con FEC 1.20.186.212 da LSR2 e visualizzeremo il contenuto delle tabelle MPLS dei LSR interessati, ci aspettiamo l'installazione del LSP lungo una delle due possibili rotte e l'installazione del detour per l'altra rotta:

```
[LSR2]
FTN table:
1.20.186.212 16 true
ILM table:
NHLFE table:
16 16 PUSH 00:0B:6B:D9:92:7A 0 10 true 0
17 16 PUSH 00:0B:6B:D9:92:17 1 10 true 0
FRRdb table:
5 4 3 16 17 - false

[LSR5]
FTN table:
ILM table:
16 16 [1.20.186.212]
NHLFE table:
16 3 POP 00:0B:6B:D9:92:80 1 10 true 0
FRRdb table:

[LSR3]
FTN table:
ILM table:
16 16 [4]
NHLFE table:
16 3 POP 00:0B:6B:D9:92:0A 0 10 true 0
FRRdb table:

[LSR4]
FTN table:
ILM table:
NHLFE table:
FRRdb table:
```

Il test ha restituito i valori attesi. Sia che in LSR5 che in LSR3 opera la funzione di PHP solo che per il primo riguarda un LSP con FEC 1.20.186.212 e il secondo un detour con destinazione LSR4.

Test V.2a - In questo test verrà effettuato il rilascio delle label degli LSP creati nel test V.1a, aspettandoci la cancellazione delle entry relative dalle tabelle MPLS.

Rilascio LSP da LSR1 con FEC 5.20.196.156:

```
[LSR1]
FTN table:
5.21.208.48 17 true
1.20.186.212 18 true
ILM table:
NHLFE table:
17 17 PUSH 00:0B:6B:D9:92:6B 1 10 true 0
18 16 PUSH 00:0B:6B:D9:92:7A 1 10 true 0
FRRdb table:

[LSR2]
FTN table:
ILM table:
17 17 [5.21.208.48]
NHLFE table:
17 3 POP 00:0B:6B:D9:92:17 1 10 true 0
FRRdb table:
```

Rilascio LSP da LSR1 con FEC 5.21.208.48:

```
[LSR1]
FTN table:
1.20.186.212 18 true
ILM table:
NHLFE table:
18 16 PUSH 00:0B:6B:D9:92:7A 1 10 true 0
FRRdb table:

[LSR2]
FTN table:
ILM table:
NHLFE table:
FRRdb table:

[LSR3]
FTN table:
ILM table:
NHLFE table:
FRRdb table:
```

Rilascio LSP da LSR1 con FEC 1.20.186.212:

```
[LSR1]
FTN table:
ILM table:
NHLFE table:
FRRdb table:

[LSR2]
FTN table:
ILM table:
NHLFE table:
FRRdb table:
```

```
[LSR3]
FTN table:
ILM table:
NHLFE table:
FRRdb table:

[LSR4]
FTN table:
ILM table:
NHLFE table:
FRRdb table:

[LSR5]
FTN table:
ILM table:
NHLFE table:
FRRdb table:
```

Tutti i test hanno restituito i valori attesi, cioè la cancellazione delle rispettive entrate nelle tabelle MPLS.

Test V.2b - In questo test verrà effettuato il rilascio del LSP con detour, installato nel test V.1b:

```
[LSR2]
FTN table:
ILM table:
NHLFE table:
FRRdb table:

[LSR5]
FTN table:
ILM table:
NHLFE table:
FRRdb table:

[LSR3]
FTN table:
ILM table:
NHLFE table:
FRRdb table:

[LSR4]
FTN table:
ILM table:
NHLFE table:
FRRdb table:
```

Il rilascio del LSP e del detour è avvenuto correttamente, come mostrato dai risultati che riportano la cancellazione di tutte le entry delle tabelle MPLS.

Test V.3 - In questo test verranno disattivati al transito gli LSP creati nel test V.1a e successivamente riattivati.

Disattivazione LSP da LSR1 con FEC 5.20.196.156:

```
[LSR1]
FTN table:
5.21.208.48 17 true
5.20.196.156 16 false
1.20.186.212 18 true
ILM table:
NHLFE table:
16 16 PUSH 00:0B:6B:D9:92:6B 1 10 true 0
17 17 PUSH 00:0B:6B:D9:92:6B 1 10 true 0
18 16 PUSH 00:0B:6B:D9:92:7A 1 10 true 0
FRRdb table:
```

Disattivazione LSP da LSR1 con FEC 5.21.208.48:

```
[LSR1]
FTN table:
5.21.208.48 17 false
5.20.196.156 16 false
1.20.186.212 18 true
ILM table:
NHLFE table:
16 16 PUSH 00:0B:6B:D9:92:6B 1 10 true 0
17 17 PUSH 00:0B:6B:D9:92:6B 1 10 true 0
18 16 PUSH 00:0B:6B:D9:92:7A 1 10 true 0
FRRdb table:
```

Disattivazione LSP da LSR1 con FEC 1.20.186.212:

```
[LSR1]
FTN table:
5.21.208.48 17 false
5.20.196.156 16 false
1.20.186.212 18 false
ILM table:
NHLFE table:
16 16 PUSH 00:0B:6B:D9:92:6B 1 10 true 0
17 17 PUSH 00:0B:6B:D9:92:6B 1 10 true 0
18 16 PUSH 00:0B:6B:D9:92:7A 1 10 true 0
FRRdb table:
```

Come si vede dai risultati dei test la disattivazione al transito si ottiene con il settaggio a `false` del flag di validità del rispettivo LSP; ciò avviene solamente nel LSR di ingresso del LSP.

Riattivazione LSP da LSR1 con FEC 5.20.196.156:

```
[LSR1]
```

```
FTN table:
5.21.208.48 17 false
5.20.196.156 16 true
1.20.186.212 18 false
ILM table:
NHLFE table:
16 16 PUSH 00:0B:6B:D9:92:6B 1 10 true 0
17 17 PUSH 00:0B:6B:D9:92:6B 1 10 true 0
18 16 PUSH 00:0B:6B:D9:92:7A 1 10 true 0
FRRdb table:
```

Riattivazione LSP da LSR1 con FEC 5.21.208.48:

```
[LSR1]
FTN table:
5.21.208.48 17 true
5.20.196.156 16 true
1.20.186.212 18 false
ILM table:
NHLFE table:
16 16 PUSH 00:0B:6B:D9:92:6B 1 10 true 0
17 17 PUSH 00:0B:6B:D9:92:6B 1 10 true 0
18 16 PUSH 00:0B:6B:D9:92:7A 1 10 true 0
FRRdb table:
```

Riattivazione LSP da LSR1 con FEC 1.20.186.212:

```
[LSR1]
FTN table:
5.21.208.48 17 true
5.20.196.156 16 true
1.20.186.212 18 true
ILM table:
NHLFE table:
16 16 PUSH 00:0B:6B:D9:92:6B 1 10 true 0
17 17 PUSH 00:0B:6B:D9:92:6B 1 10 true 0
18 16 PUSH 00:0B:6B:D9:92:7A 1 10 true 0
FRRdb table:
```

I risultati ottenuti coincidono con i risultati attesi, i relativi flag di attività sono stati reimpostati a `true` riattivando il LSP al transito.

Test V.4a - In questo test abbiamo sperimentato la capacità del sistema di installare un nuovo LSP in caso di interruzione del percorso. Con riferimento alla topologia utilizzata è evidente che un test di questo tipo può essere effettuato creando un LSP tra due nodi sul loop formato da LSR2, LSR3, LSR4 e LSR5, in modo da poter consentire due possibili rotte per la stessa destinazione. L'installazione del LSP verrà effettuata da LSR5 con FEC 1.20.186.212 e simuleremo l'interruzione del link LSR5/LSR4 in modo da

forzare la creazione del LSP sul percorso LSR5/LSR2/LSR3/LSR4. L'interruzione del link è ottenuta disattivando l'interfaccia wireless 1 di LSR4. Verranno mostrate le tabelle dei LSR interessati prima e dopo avvenuta installazione del nuovo LSP.

```
[LSR5]
FTN table:
1.20.186.212 16 true
ILM table:
NHLFE table:
16 16 PUSH 00:0B:6B:D9:92:80 1 10 true 0
FRRdb table:

[LSR4]
FTN table:
ILM table:
16 16 [1.20.186.212]
NHLFE table:
16 3 POP 00:0D:B9:14:BA:D4 1 10 true 0
FRRdb table:
```

Dopo il rilevamento del nuovo percorso dal protocollo di routing:

```
[LSR5]
FTN table:
1.20.186.212 16 false
1.20.186.212 17 true
ILM table:
NHLFE table:
16 16 PUSH 00:0B:6B:D9:92:80 1 10 true 0
17 16 PUSH 00:0B:6B:D9:92:6B 0 10 true 0
FRRdb table:

[LSR2]
FTN table:
ILM table:
16 16 [1.20.186.212]
NHLFE table:
16 16 REPLACE 00:0B:6B:D9:92:17 1 10 true 0
FRRdb table:

[LSR4]
FTN table:
ILM table:
16 16 [1.20.186.212]
NHLFE table:
16 3 POP 00:0D:B9:14:BA:D4 1 10 true 0
FRRdb table:

[LSR4]
FTN table:
ILM table:
16 16 [1.20.186.212]
```

```
NHLFE table:
16 3 POP 00:0D:B9:14:BA:D4 1 10 true 0
FRRdb table:
```

Il test ha confermato i risultati attesi, si vede come dopo l'intervento sul link avviene l'installazione automatica di un nuovo LSP con la stessa FEC che coinvolge i LSR sulla seconda rotta possibile per raggiungere la destinazione.

Test V.4b - Questo test mira a validare la capacità del software LDPL di "inseguire" le variazioni dei percorsi di routing. Il test può essere effettuato installando un LSP che colleghi gli estremi della diagonale del quadrato formato da LSR2, LSR3, LSR4 e LSR5; grazie al fatto che i due percorsi possibili sono della stessa lunghezza e i dispositivi si trovano nello stesso ambiente, i due percorsi hanno delle metriche essenzialmente simili; sono sufficienti quindi delle leggere fluttuazioni di questi valori per prediligere un percorso rispetto ad un altro. In ambito di laboratorio infatti abbiamo sperimentato delle frequenti commutazioni da un percorso ad un altro, avvolte dovute anche alla semplice interposizione temporanea di ostacoli fisici tra i dispositivi. Il test parte in seguito all'installazione da LSR2 di un LSP con FEC 1.20.186.212:

```
[LSR2]
FTN table:
1.20.186.212 16 true
ILM table:
NHLFE table:
16 16 PUSH 00:0B:6B:D9:92:7A 0 10 true 0
FRRdb table:

[LSR5]
FTN table:
ILM table:
16 16 [1.20.186.212]
NHLFE table:
16 3 POP 00:0B:6B:D9:92:80 1 10 true 0
FRRdb table:

[LSR3]
FTN table:
ILM table:
NHLFE table:
FRRdb table:

[LSR4]
FTN table:
ILM table:
NHLFE table:
```

```
FRRdb table:
```

Subito dopo la commutazione della rotta Srcr:

```
[LSR2]
FTN table:
1.20.186.212 16 false
1.20.186.212 17 true
ILM table:
NHLFE table:
16 16 PUSH 00:0B:6B:D9:92:7A 0 10 true 0
17 16 PUSH 00:0B:6B:D9:92:17 1 10 true 0
FRRdb table:

[LSR5]
FTN table:
ILM table:
16 16 [1.20.186.212]
NHLFE table:
16 3 POP 00:0B:6B:D9:92:80 1 10 true 0
FRRdb table:

[LSR3]
FTN table:
ILM table:
16 16 [1.20.186.212]
NHLFE table:
16 3 POP 00:0B:6B:D9:92:0A 0 10 true 0
FRRdb table:

[LSR4]
FTN table:
ILM table:
NHLFE table:
FRRdb table:
```

Ritorno al LSP originario:

```
[LSR2]
FTN table:
1.20.186.212 16 true
1.20.186.212 17 false
ILM table:
NHLFE table:
16 16 PUSH 00:0B:6B:D9:92:7A 0 10 true 0
17 16 PUSH 00:0B:6B:D9:92:17 1 10 true 0
FRRdb table:

[LSR5]
FTN table:
ILM table:
16 16 [1.20.186.212]
NHLFE table:
16 3 POP 00:0B:6B:D9:92:80 1 10 true 0
```

```
FRRdb table:

[LSR3]
FTN table:
ILM table:
16 16 [1.20.186.212]
NHLFE table:
16 3 POP 00:0B:6B:D9:92:0A 0 10 true 0
FRRdb table:

[LSR4]
FTN table:
ILM table:
NHLFE table:
FRRdb table:
```

Il test ha soddisfatto le aspettative, come si vede dal contenuto delle tabelle l'installazione del nuovo LSP non prevede la cancellazione del vecchio ma semplicemente il suo blocco al transito. Questo comportamento è utile in condizioni di frequente commutazione tra due percorsi, tenendo gli LSP già pronti in memoria come in una cache, è sufficiente agire sui flag del FTN che ne determinano l'utilizzo.

Test V.5 - In questo test valideremo l'attivazione e la disattivazione all'utilizzo di un detour. Con riferimento al LSP installato nel test V.1b, prima attiveremo l'utilizzo del detour a protezione di LSR5 e successivamente lo disattiveremo. Attivazione del detour da LSR2, a protezione di LSR5:

```
[LSR2]
FTN table:
1.20.186.212 16 true
ILM table:
NHLFE table:
16 16 PUSH 00:0B:6B:D9:92:7A 0 10 true 0
17 16 PUSH 00:0B:6B:D9:92:17 1 10 true 0
FRRdb table:
5 4 3 16 17 - true
```

Disattivazione del detour da LSR2, a protezione di LSR5:

```
[LSR2]
FTN table:
1.20.186.212 16 true
ILM table:
NHLFE table:
16 16 PUSH 00:0B:6B:D9:92:7A 0 10 true 0
17 16 PUSH 00:0B:6B:D9:92:17 1 10 true 0
FRRdb table:
5 4 3 16 17 - false
```

L'esito dei test corrisponde con le aspettative, lo stato del detour è indicato dal flag della relativa entry del FRRdb. Si noti che l'indice della entry FTN (16) non è cambiato anche quando il detour era attivo, in realtà l'indice effettivo nelle tabelle MPLS che risiedono nell'elemento Click è 17, nelle tabelle locali LDPL viene lasciato l'indice del LSP originario per poter lanciare in qualsiasi momento la procedura corretta di release del LSP, anche con detour attivo (sezione 5.2.1).

6.3 Test di prestazioni

Test P.1 - I questo test sono stati effettuati venti installazioni campione per ogni LSP della stessa lunghezza. La topologia del testbed è stata riconfigurata in modo da permettere l'installazione di LSP di lunghezza massima sempre nei limiti del numero di dispositivi a nostra disposizione. La riconfigurazione è stata ottenuta disattivando l'interfaccia wireless 0 di LSR5 ottenendo la topologia in fig. 6.2.

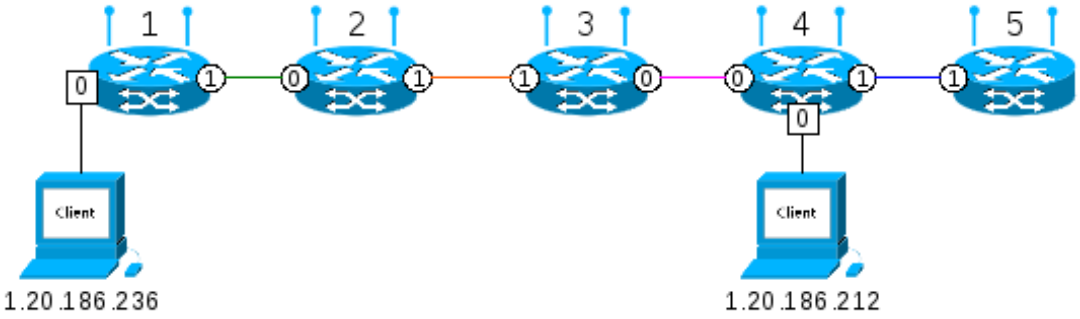


Figura 6.2: Topologia del testbed per l'installazione di LSP di lunchezza massima.

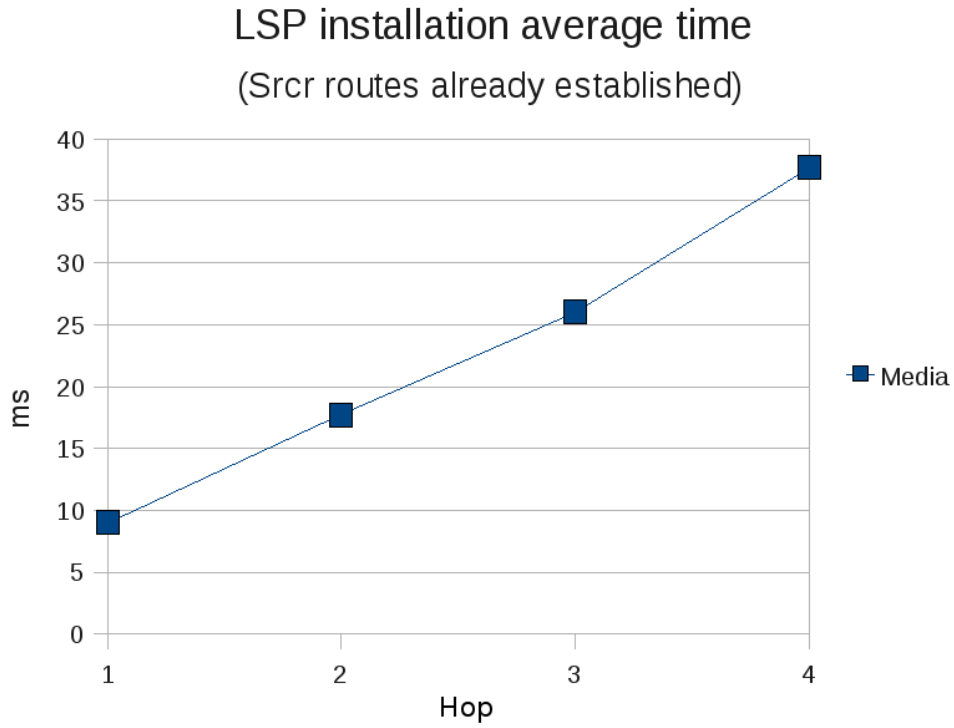
L'intervallo di tempo considerato per l'installazione di un LSP parte dall'istante di ricezione di un comando di installazione del LSP dal pannello di controllo fino alla definitiva configurazione delle tabelle MPLS dell'ultimo LSR coinvolto nella procedura.

Le misurazioni sono state condotte in due situazioni differenti, la prima, riportata in fig. 6.3(a), corrisponde all'installazione di LSP con la presenza di una rotta Srcr per la destinazione specificata come FEC, la seconda, riportata in fig. 6.3(b), corrisponde all'installazione di LSP senza la presen-

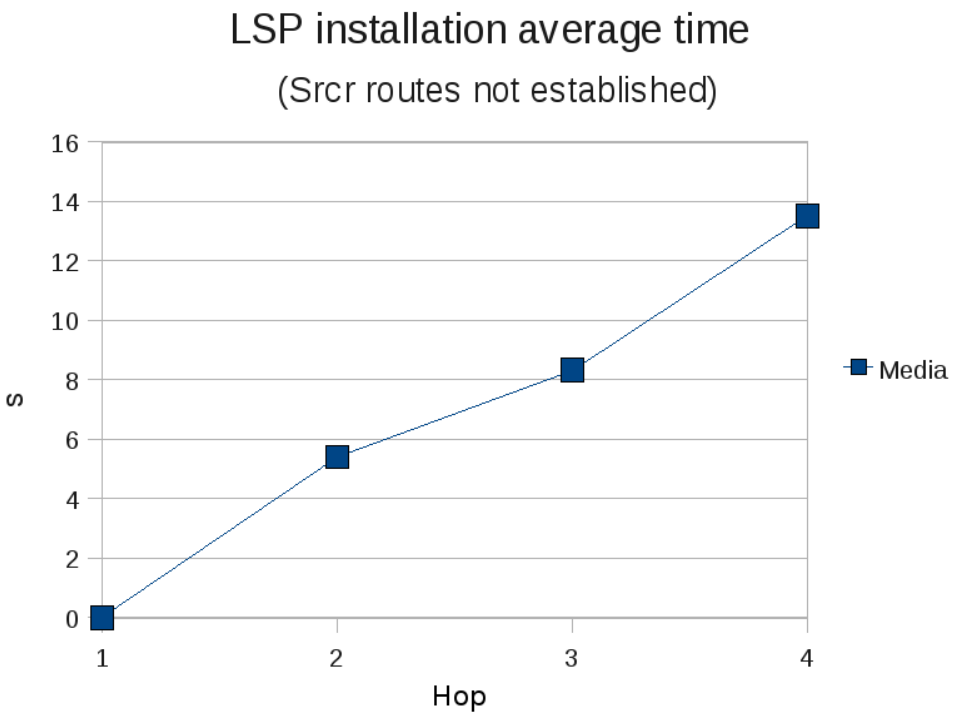
za di percorsi Srcr esistenti per la FEC specificata: i tempi si discostano notevolmente a partire da LSP con più di un nodo, su cui incidono i tempi di flooding del protocollo di routing per il calcolo della rotta non ancora esistente. Al contrario i tempi per LSP di 1 nodo coincidono perchè Srcr mantiene in memoria le rotte per i nodi direttamente connessi.

In fig. 6.3(a), è evidenziato l'andamento lineare dei tempi medi in proporzione al numero dei nodi coinvolti nell'installazione, questo risultato era stato previsto perchè legato intrinsecamente alla procedura sequenziale di installazione di un LSP.

Test P.2 - Questi test sono stati condotti per misurare il tempo medio impiegato dal protocollo LDPL ad accorgersi del cambiamento di una rotta a seguito di un guasto di un LSR intermedio o all'interruzione di un link del LSP. Abbiamo effettuato venti misurazioni campione simulando casualmente sia l'interruzione del link (disattivazione della relativa interfaccia wireless) che il guasto del dispositivo (interruzione di corrente). I risultati di questo test riportano un intervallo medio di 230 secondi che corrisponde con i tempi sperimentati dal protocollo di routing per il ricalcolo di una rotta nelle medesime condizioni.



(a)



(b)

Figura 6.3: Tempi medi d'installazione di un LSP.

Capitolo 7

Conclusioni

In questo lavoro di tesi è stato realizzato un sistema di distribuzione delle label conforme con LDP che è stato reso operativo su un testbed di rete wireless-mesh con tecnologia 802.11 e funzionalità di label switching conforme con MPLS. In seguito il sistema di distribuzione delle label è stato esteso aggiungendo la possibilità di installare dei percorsi di backup (detour) a protezione di un nodo o link del percorso principale (LSP). Il lavoro è cominciato con le opportune modifiche al testbed realizzato con riferimento al testbed Roofnet (MIT e dall'Università Cambridge Massachusetts). In questo sistema le funzioni di routing e forwarding MPLS erano implementate come script di Click Modular Router ed è stato necessario modificarle in modo da renderle compatibile con le operazioni svolte dal sistema di distribuzione delle label: le modifiche consistono nella possibilità di inviare messaggi broadcast e multicast sfruttando la natura broadcast del mezzo wireless, messaggi che prima venivano sistematicamente scartati dal primo nodo della WMN, in seguito abbiamo modificato i componenti che effettuavano il label switching aggiungendo un elemento che permettesse l'utilizzo della tecnica di fast reroute; abbiamo fornito anche un'interfaccia che permette l'attivazione di questa tecnica da futuri componenti di Link Management, specializzati nel rilevamento dello stato di un link o di un nodo. Successivamente abbiamo implementato il sistema di distribuzione delle label come realizzazione parziale dello standard RFC 5036, in modo da permettere l'installazione e la rimozione di LSP. Inoltre sfruttando le caratteristiche del protocollo di routing Srcr esistente, abbiamo adattato la procedura di installazione di un LSP in modo che sfrutti le rotte calcolate da esso e vi si mantenga solidale. In seguito, il protocollo di scambio delle label

è stato esteso aggiungendo la funzione automatica di rilevamento e installazione di detour a protezione dei nodi lungo il LSP; l'utilizzo del detour inoltre permette di ovviare ai tempi di riconvergenza del protocollo di routing e garantire la continuità del servizio. L'attivazione e la disattivazione del traffico lungo queste rotte di backup è resa possibile grazie alla funzione di fast reroute aggiunta. Infine abbiamo realizzato un pannello di controllo remoto che permette la visualizzazione agevolata dello stato dei nodi della rete, l'installazione e rimozione di un qualunque LSP nella rete e anche l'attivazione e disattivazione di un qualunque detour della rete. Al termine del lavoro sono stati svolti gli opportuni test di validazione del protocollo che hanno rilevato la correttezza dell'implementazione nei casi di interesse previsti e i test sulle prestazioni di installazione di un LSP lungo la rete che ha mostrato i limiti introdotti dal protocollo di routing nel calcolare nuove rotte. In particolare quest'ultimo punto mostra la necessità di verificare il comportamento del protocollo di distribuzione delle label in combinazione con diversi protocolli di routing per ricercare prestazioni migliori.

Bibliografia

[1] L. Andersson, Ed. Acreo AB I. Minei, Ed. Juniper Networks B. Thomas, Ed. Cisco Systems, Inc. , *LDP Specification*, RFC 5036 Ottobre 2007

[2] M. Atzori, S. Rossi. *Realizzazione e analisi delle prestazioni di un prototipo di rete wireless mesh 802.11 con funzionalit‘ di label switching*. 2008

[3] J. Bicket, D. Aguayo, S. Biswas, R. Morris. *Architecture and Evaluation of an Unplanned 802.11b Mesh Network*. Settembre 2005

[4] H. Lundgren, K. Ramachandran, E. Belding-Royer, K. Almeroth, M. Benny, A. Hewatt, A.Touma, A. Jardosh. *Experiences from the Design, Deployment and Usage of the UCSB MeshNet Testbed*. Aprile 2006

[5] M. Gunes, U. Meis, J. Ritzerfeld, A. Zimmermann, M. Wenig. *How to study Wireless Mesh Networks: A hybrid Approach*, 2007

[6] D. Aguayo, J. Ticket, R. Morris SrcRR: *A High Throughput Routing Protocol for 802.11 Mesh Networks*. Maggio 2006

[7] Tiziano Fotoni: *MPLS fondamenti e applicazioni alle reti IP* , Hoepli Incormatica, 2003

[8] Uyless Black: *MPLS and Label Switcing Networks*, Prentice Hall PTR, 2002

[9] Thomas D. Nadeau: *MPLS Network Management*, Morgan Kaufmann, 2003

[10] *Multiprotocol Label Switching Architecture.* `http://www.ietf.org/rfc/rfc3031.txt`, Gennaio 2001

[11] *Click Modular Router.* `http://www.pdos.lcs.mit.edu/click`, Febbraio 2008

[12] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti,M. Frans Kaashoek. *The Click Modular Router, ACM Transactions on Computer Systems*. Agosto2000

[13] Eddie Kohler. *The Click Modular Router*. 2001

[14] *Click Modular Router Element Class Reference*. `www.read.cs.ucla.edu/click/doxygen/classElement.html`, Marzo 2008

[15] *OpenWrt*. `http://openwrt.org/`, Febbraio 2008

[16] Daniel Aguayo, John Bicket, Sanjit Biswas, Glenn Judd, and Robert Morris. *A measurement study of a rooftop 802.11b mesh network*. In Proc. ACM SIGCOMM Conference (SIGCOMM 2004), September 2004

[17] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. *A high-throughput path metric for multi-hop wireless routing*. In Proceedings of the 9th ACM International Conference on Mobile Computing and Networking (MobiCom 03), San Diego, California, September 2003.

[18] R. Draves, J. Padhye, and B. Zill. *Comparison of routing metrics for static multi-hop wireless networks*. In Proc. ACM SIGCOMM Conference (SIGCOMM 2004), September 2004.

[19] David B. Johnson. *Routing in ad hoc networks of mobile hosts*. In Proc. of the IEEE Workshop on Mobile Computing Systems and Applications, pag. 158163, December 1994.

[20] David B. Johnson, David A. Maltz, Josh Broch. *DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks*. 1999

[21] Jinyang Li, Charles Blake, Douglas S. J. De Couto, Hu Imm Lee, and Robert Morris. *Capacity of ad hoc wireless networks*. In Proceedings of the 7th ACM International Conference on Mobile Computing and Networking, pag 6169, Rome, Italia, Luglio 2001.

[22] N. Shen, E. Chen, A. Tian. *Discovering LDP Next-Next-Hop Labels*. Novembre 2005

[23] Fast Light Toolkit. `www.fltk.org`

[24] FLTK Programing manual and class reference. `www.fltk.org/doc-1.1/toc.html`