Realizzazione di un modulo NS2 per il calcolo di percorsi punto-multipunto in reti MPLS

Tesi di laurea specialistica

Università di Pisa
Aprile 2008

UNIVERSITÀ DI PISA
Facoltà di Ingegneria

Corso di laurea in
INGEGNERIA DELLE TELECOMUNICAZIONI

Realizzazione di un modulo NS2 per il calcolo di percorsi punto-multipunto in reti MPLS

Relatori:
Prof. Stefano Giordano
Prof. Michele Pagano
Prof. Davide Adami

Candidato:
Lorenzo Bartoli

Anno Accademico 2007-2008

Ringraziamenti

Voglio ringraziare i Prof. Stefano Giordano, Michele Pagano, Davide Adami e, non ultimo, l'ing. Christin Callegari che mi hanno seguito in questo lavoro di tesi. La loro preparazione e la loro passione sono e saranno un esempio da seguire. Desidero ringraziare mio padre, mia madre e mio fratello Leo che mi hanno sostenuto in questo lavoro. È solo grazie a loro che sono riuscito a portare a termine questo percorso. Ringrazio inoltre tutti gli amici e compagni di studi. Infine, un ringraziamento speciale a Simona per il suo amore e la grande forza d'animo.

Indice

1 MPLS

In questo capitolo introduciamo il *MultiProtocol Label Switching* (*MPLS*) che rappresenta in qualche modo la convergenza di due approcci fondamentali del networking: datagram e virtual circuit. MPLS nasce a metà degli anni novanta dall'evoluzione di diverse tecnologie private; la piú nota è IP Switching della Ipsilon, nata nel 1996[1]. La base comune è usare esclusivamente una etichetta (label) di lunghezza fissa, presente nell'header del pacchetto, per stabilire qual'è il next hop per quel pacchetto e quindi inoltrarlo, dopo aver eventualmente cambiato la label. Per implementare il label swapping l'*IETF* (*Internet Engineering Task Force*) organizza nel Dicembre 1996 un BOF (Birds Of Feather) e successivamente un working group, che prese il nome "neutrale" di MultiProtocol Label Switching per non avere legami con nessuna delle soluzioni private, dando così vita all'MPLS.

Il resto del capitolo è così ordinato: il Paragrafo 1.1 introduce gli aspetti generali, il Paragrafo 1.2 tratta la componente di forwarding di MPLS e il Paragrafo 1.3 analizza la componente di controllo. Infine nel Paragrafo 1.4 viene introdotto un importante aspetto di MPLS, il *Traffic Engineering*.

[1] Già nel 1994 Toshiba aveva presentato uno schema simile, implementato successivamente nel Cell Switching Router (CSR). In seguito sono nate ulteriori proposte, il Tag Switching della Cisco e l'Aggregate Route-based IP Switching (ARIS) dell'IBM.

1.1 Aspetti generali

Come mostra la Figura 1.1, un dominio MPLS è un insieme contiguo di nodi detti *LSR* (*Label Switching Router*) che supportano la tecnologia MultiProtocol Label Switching e che sono quindi in grado di inoltrare i pacchetti in base alla label. Particolarmente importanti sono i nodi di confine o *LER* (*Label Edge Router*), i quali, oltre a dover assegnare le label ai pacchetti che entrano nel dominio MPLS, devono supportare sia il forwarding label switching che quello tradizionale, essendo di fatto il collegamento fra il dominio MPLS e il resto della rete. Quando un pacchetto entra in un dominio

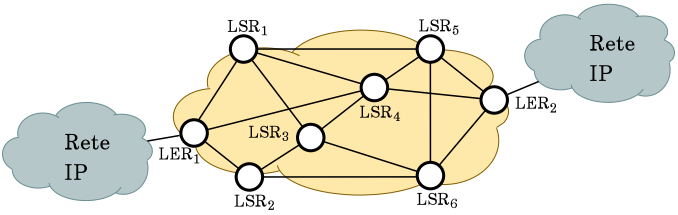


Fig. 1.1: Esempio di dominio MPLS.

MPLS il router di confine, detto ingress, gli assegna una label attraverso la quale è inoltrato lungo il proprio percorso dai LSR intermedi. All' uscita dal dominio un LER, denominato egress, ha il compito di togliere l'etichetta dal pacchetto ed instradarlo secondo le regole delforwarding tradizionale. Nel linguaggio MPLS, la connessione virtuale stabilita associando delle etichette tra due LSR prende il nome di *LSP* (*Label Switched Path*); un LSP è così l'analogo in una rete MPLS della connessione virtuale in reti connection-oriented e tutti i pacchetti appartenenti ad una connessione seguiranno il percorso individuato dal LSP. Gli LSP sono monodirezionali e trasportano traffico in una sola direzione; eventuale traffico nella direzione opposta viene trasportato da un LSP diverso.

Il routing di livello rete può essere distinto in due componenti fondamentali:

(i) *Parte di forwarding*;

(ii) *Parte di controllo*.

I prossimi due paragrafi presentano questi aspetti.

1.2 Parte di forwarding

La parte di forwarding è responsabile dell'inoltro dei pacchetti dall'ingresso all'uscita di un router. Per svolgere questa operazione utilizza due tipi di informazioni:

1. Le informazioni trasportate dal pacchetto (in questo caso le label);
2. Le tabelle di forwarding del router.

La parte di forwarding prevede inoltre una serie di algoritmi per la scelta di quali informazioni del pacchetto utilizzare e per la ricerca delle entry nelle forwarding table.

1.2.1 FEC e Label

Una *FEC* (*Forwarding Equivalent Class*) non è altro che un gruppo di pacchetti che vengono inoltrati allo stesso modo. Caratteristica importante della FEC è la *granularità*, che può essere di tipo *coarse*, cui corrispondono flussi di grandi dimensioni (macroflussi) o *fine*, con flussi di piccole dimensioni (microflussi). Una granularità coarse, se da un lato garantisce maggior scalabilità alla rete, richiedendo la gestione di un minor numero di flussi, dall'altro rende il sistema meno flessibile in termini di QoS non permettendo una differenziazione accurata. Per raggiungere un trade-off tra scalabiltà e flessibilità è necessario poter supportare un ampio spettro di granularità delle FEC.

Quando un LER riceve in ingresso un pacchetto IP senza label analizza l'header per stabilire a quale FEC appartiene il pacchetto, quindi aggiunge al pacchetto l'header MPLS, denominato *Shim header*. Lo Shim header, rappresentato in Figura 1.2, contiene i campi:

label (20 bit): valore della label[2] associata al pacchetto,

[2]Alcuni valori del campo sono riservati: 0 IPv4 Explicit Null: indica

exp (3 bit): campo per usi sperimentali,

S (1 bit): se settato ad uno indica che è l'ultimo elemento della pila;

TTL (8 bit): analogo al corrispondente campo del pacchetto IP.

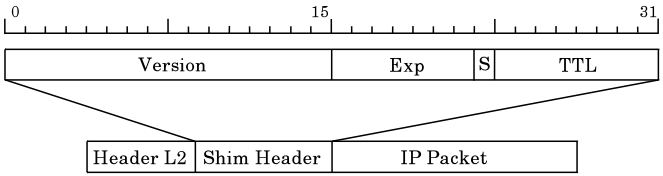


Fig. 1.2: Formato dello shim header MPLS.

Notiamo che il campo TTL dello shim header può essere gestito in due modi distinti:

(i) Può contenere il valore del corrispettivo campo dell'header IP al momento dell'incapsulamento, in tal caso il TTL viene decrementato in ogni LSR e ricopiato nel nuovo TTL IPv4 all'uscita dal dominio MPLS,

(ii) Può contenere un opportuno valore indipendente da quello contenuto nell'header IP. In tal caso il TTL IP sarà decrementato solo di 1 dal LER di uscita.

1.2.2 Label switching

MPLS utilizza per l'inoltro dei pacchetti un paradigma tipico delle reti connection-oriented: il *label switching* o *commutazione di etichetta*. Per l'inoltro dei pacchetti all'interno della rete vengono svolte le seguenti operazioni:

- Il pacchetto viene classificato quando giunge ad una rete MPLS assegnandolo ad una specifica FEC,

che lo Shim header deve essere rimosso e il pacchetto deve essere inoltrato a livello 3, 1 Router Alert Label: Indica che il pacchetto deve essere analizzato a livello software dal nodo, 2 IPv5 Explicit Null: corrispettivo IPv6 di 0, 3 IPv4 Implicit Null: utilizzato nel Penultimate Hop Popping (Paragrafo 1.2.3), 4-15 riservati.

- Al pacchetto vengono associate una o piú etichette, un Time To Live e un identificativo per segnalare la fine della pila di etichette,

- L'inoltro del pacchetto nei router successivi al primo avviene in base alle sole informazioni fornite dalla label;

- La label viene eliminata quando il pacchetto esce dalla rete MPLS.

Per prima cosa notiamo che l'operazione di assegnazione di un pacchetto ad una FEC avviene solo all'ingresso del dominio MPLS, semplificando di molto la procedura di inoltro nei router intermedi. In questo modo il sistema non solo risulta piú veloce, ma anche piú flessibile, in quanto l'assegnazione iniziale ad una FEC potrebbe risultare anche molto sofisticata, coinvolgendo piú campi delle intestazioni protocollari, senza gravare sul carico computazionale dei nodi intermedi. Una seconda osservazione riguarda il fatto che non è stata fatta alcuna ipotesi sul contenuto trasportato (che può essere un pacchetto di Livello 3 o una trama di Livello 2) e che il termine "pacchetto" è stato usato solo per comodità linguistica.

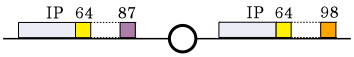
1.2.3 Gestione delle label

L'etichetta è un identificativo con validità locale che definisce la porta di uscita e le operazioni da compiere sulla pila di etichette, che sono:

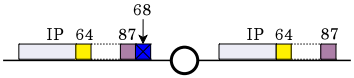
- *swap*: Il valore piú alto dell'etichetta viene cambiato, questa operazione è mostrata in Figura 1.3a.

- *pop*: Come mostrato in Figura 1.3b eliminiamo l'etichetta di livello piú alto;

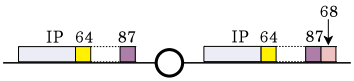
- *push*: Come mostrato in Figura 1.3c eseguiamo swap e inseriamo nella pila una nuova etichetta.



(a) Operazione si swap.



(b) Operazione di pop.



(c) Operazione di push.

Fig. 1.3: Operazioni sulle label.

Label Stacking

MPLS fornisce l'opportunità di annidare piú etichette all'interno di un pacchetto e tale opzione, denominata *Label Stacking*, gestisce le label utilizzando uno stack (LIFO: Last In Firt Out). Un esempio di Label Stacking è mostrato in Figura 1.4.

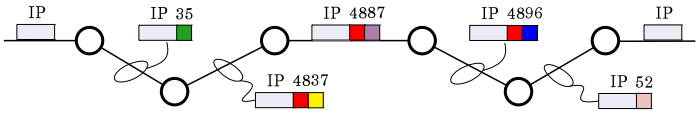


Fig. 1.4: Label Stacking.

Penultimate Hop Popping

Se è attiva la modalità *Penultimate Hop Popping* (*PHP*) il penultimo nodo del LSP si accorge che il successivo è il nodo di uscita ed esegua un'operazione di pop sulla pila inoltrando all'ultimo nodo un IP tradizionale. Come mostra la Figura 1.5, il PHP permette di evitare un'inutile consultazione della LIB.

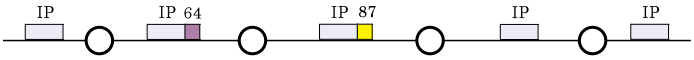


Fig. 1.5: Penultimate Hop Popping.

Label Merging

Il *Label Merging* fornisce ad un LSR la possibilità di aggregare piú flussi in un unico flusso. Infatti, come evidenziato in Figura 1.6, quando arrivano due pacchetti appartenenti ad una stessa FEC con label diverse, possono essere inoltrati sulla stessa interfaccia di uscita con un'unica label.

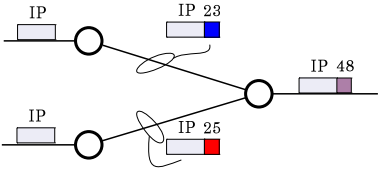


Fig. 1.6: Label Merging.

1.2.4 Tabelle di routing

In MPLS ogni LSR, oltre alle tradizionali tabelle di routing IP, mantiene al suo interno altre due tabelle:

- *LIB* (*Incoming Label Map*): come mostra la Figura 1.7, ogni label di ingresso punta ad una o piú entry detta *NHLFE* (*Next Hop Label Forwarding Entry*) che contiene il next hop, la label di uscita, le operazioni da effettuare (swap, pop, push) e informazioni aggiuntive come, ad esempio, l'hop count o l'incapsulamento di livello 2. La ILM[3] è utilizzata negli LSR intermedi.
- *FTN* (*FEC-To-NHLF*): ha una struttura del tutto analoga alla ILM ma si utilizza nei LER quando arriva un pacchetto non ancora etichettato, ad indicizzare la NHLFE è una FEC piuttosto che una label.

[3]Nella terminologia Cisco la ILM è denominata *LFIB* (*Label Forwarding Information Base*).

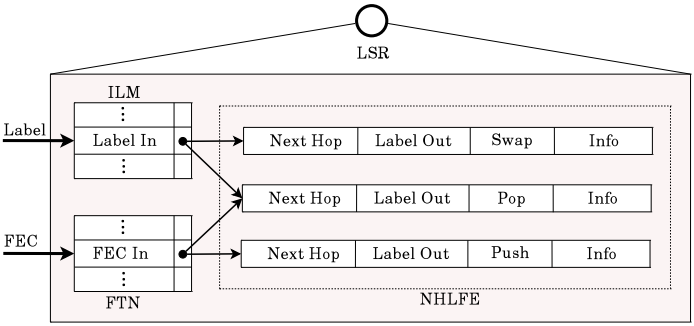


Fig. 1.7: Esempio di ILM, FTN e NHLFE.

Per la creazione di queste due tabelle ogni LSR si avvale di una terza tabella:

- *LIB* (*Label Information Base*): come mostra la Tabella 1.1, in una generica riga della LIB sono contenuti la FEC, un'etichetta locale ed una remota associate alla FEC, e l'identificativo dell'LSR remoto che ha comunicato il binding FEC-etichetta.

FEC	Local label	Remote label	Provenience
195.31.235.0/24	29	45	175.16.5.2
195.106.28.0/24	23	43	175.16.1.8

Tab. 1.1: Struttura della LIB.

Notiamo come, per diminuire l'occupazione di memoria e le etichette utilizzate, non tutti i binding FEC-label comunicati dai vari LSR servono alla creazione di ILM e FTN, per cui alcune associazioni non saranno salvate nella LIB. A tal proposito ogni LSR può scegliere tra due alternative:

(i) *conservative label retention mode*: ogni LSR scarta le informazioni ricevute dagli LSR che non sono il next hop per una certa FEC;

(ii) *liberal label retention mode*: ogni LSR mantiene in memoria tutte le associazioni FEC-etichetta ricevute, an-

che quelle non provenienti dal next hop per la FEC in esame.

Chiaramente il conservative label retention mode ha il vantaggio di accellerare la costruzione di ILM e FTN nel caso di modifiche alla topologia della rete.

1.3 Parte di controllo

Descriviamo adesso gli aspetti associati alla parte di controllo di MPLS, responsabile della distribuzione delle informazioni di routing fra gli LSR e della conversione di queste informazioni nelle tabelle di forwarding, utilizzate poi dalle componenti di forwarding label switching.

Per quanto riguarda la distribuzione delle informazioni di routing, può essere effettuata da tutti i classici protocolli di routing (OSPF, BGP, PIM, e così via). Sono poi necessarie altre procedure per il binding tra label e FEC e per la distribuzione di queste informazioni agli LSR. I due mapping combinati forniscono le informazioni necessarie alla costruzione delle tabelle di forwarding.

1.3.1 Creazione delle Label

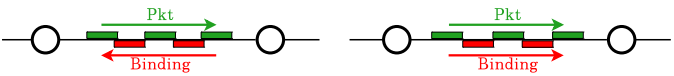
In MPLS esistono due alternative per la creazione delle label:

(i) associazione *downstream*;

(ii) associazione *upstream*.

Nel caso di associazione downstream, rappresentato in Figura 1.8a, il binding tra l'abel e FEC è svolto dal LSR successivo, rispetto all'andamento del traffico, al LSR che pone poi la label nel pacchetto. I pacchetti di binding viaggiano quindi in direzione opposta al traffico. Nel caso upstream invece il binding è svolto dal LSR che pone la label nel pacchetto e, come mostra la Figura 1.8b, i pacchetti di binding e quelli di traffico viaggiano nella stessa direzione.

La creazione o la distruzione di un binding tra una label e una FEC è sempre il risultato di un particolare evento



(a) Associazione downstream. **(b)** Associazione upstream.

Fig. 1.8: Associazione tra FEC e label.

che può essere causato dalla necessità dell'LSR di inoltrare pacchetti di dati oppure dal processing delle informazioni di controllo. Le due alternative prendono il nome di binding *data-driven* e *control-driven*, e al loro interno presentano una vasta gamma di opzioni. La scelta di un approccio o dell'altro comporta tutta una serie di ripercussioni sull'efficienza, sulla scalabilità e sulla robustezza della rete. Il working group MPLS ha scelto, dopo un'attenta analisi delle prestazioni e dell'attuale conformazione della rete mondiale, una modalità di binding downstream e control-driven[4].

1.3.2 Distribuzione delle label

Una volta che un LSR ha creato (o distrutto) un binding tra una label e una FEC, è necessario informare gli altri LSR. Per fare questo ci sono diversi metodi: il piggybacking, grazie al quale queste informazioni vengono trasportate insieme alle informazioni di routing. Il piggybacking presenta diversi vantaggi:

- Rende consistenti le informazioni di label binding e le informazioni di routing, permettendo di evitare situazioni in cui sono disponibili solo le une o le altre;
- Semplifica l'intero sistema, eliminando la necessità di un protocollo separato per la distribuzione di queste nuove informazioni.

Purtroppo il piggybacking presenta anche diversi svantaggi:

- Non tutti i protocolli di routing supportano efficientemente il trasporto di altre informazioni (per esempio l'OSPF presenta diversi problemi),

[4]Sebbene sia permessa anche la modalità data-driven.

- Alcuni nodi potrebbero non essere in grado di processare le informazioni aggiuntive,

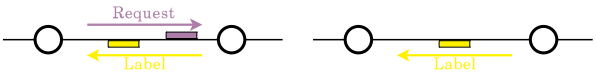
- Aumenta la complessità del sistema (pensiamo alle variazioni nel formato dei messaggi).

In alcuni casi si rende quindi necessario un protocollo distinto per la distribuzione delle informazioni di label binding, quelli standardizzati da IETF sono: LDP, LDP-CR e RSVP-TE. Tuttavia la RFC 3468 del febbraio 2003 ha stabilito che solo il protocollo RSVP-TE avrà un seguito nel processo di standardizzazione, mentre non saranno piú prodotti standard basati su CR-LDP. L'uso di un protocollo distinto può causare problemi quali l'inconsistenza fra le informazioni di binding e quelle di routing, competizione fra protocolli diversi e maggiore complessità.

Indipendentemente dal protocollo utilizzato MPLS prevede due modalità distinte per la distribuzione delle label:

(i) *downstream on demand*: il binding per una certa FEC è richiesto dal LSR downstream al suo next hop, così facendo non si propagano informazioni superflue sulla rete. Come in Figura 1.9a, un LSR non spedisce informazioni su un determinato binding al nodo upstream se non ha ricevuto informazioni su tale associazione dal nodo downstream;

(ii) *unsolicited downstream*: come mostrato in Figura 1.9b, un LSR decide di inviare il binding al LSR downstream, con una maggior velocità di instaurazione degli LSR, non rappresentando una soluzione "ordinata".



(a) Downstream on demand.

(b) Unsolicited downstream.

Fig. 1.9: Distribuzione delle label.

Anche la creazione di un LSP può seguire due modalità:

(i) *ordered control*: le label vengono assegnate da un capo all'altro dell'LSP;

(ii) *indipendent control*: le label vengono assegnate nodo per nodo in modo indipendente.

1.4 Traffic Engineering

In questo paragrafo illustreremo come MPLS può essere usato per supportare meccanismi di *Traffic Engineering* (*TE*). Le tradizionale architettura IP inoltra il traffico su percorsi calcolati in base ad algoritmi di routing, che solitamente selezionano il percorso a costo minimo[5], senza tener conto dello stato dei link. Lo scopo del TE è rendere efficienti ed affidabili le operazioni di rete ottimizzando contemporaneamente le risorse utilizzate. MPLS mette a disposizione tre meccanismi per fare TE:

1. Constraint Based Routing,
2. Fast Rerouting
3. Load Balancing.

A queste potenzialità sono dedicati i prossimi paragrafi.

1.4.1 Constrait Based Routing

Il *Constraint Based Routing* permette di scegliere il percorso su cui instradare i pacchetti basandoci non soltanto sulla metrica di costo associata ma permette di considerare altri fattori come banda e ritardo. Considerando la Figura 1.10 in cui i link hanno capacità di 4 Mbps e pansiamo che le sorgenti, CBR a 2 Mbps, collegate a LSR_1 vogliano parlare con i terminali collegati a LSR_5. Il routing tradizionale di IP instrada tutti i pacchetti sul percorso LSR_2-LSR_3-LSR_4-LSR_5 saturando i link. Con il Constraint Based Routing le prime due richieste seguono il percorso ottimo e la terza è instradate sul percorso LSR_2-LSR_7-LSR_6-LSR_5 non avendo

[5]Oppure quallo con il minor numero di hop.

piú risorse su quello ottimo. In definitiva con il routing Constraint Based Routing i flussi vengono instradati sui percorsi migliori tali da garantire certe prestazioni end-to-end.

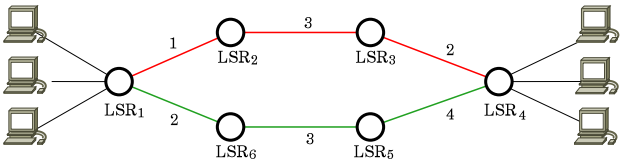


Fig. 1.10: Esempio di Constraint Based Routing.

1.4.2 Fast Reroouting

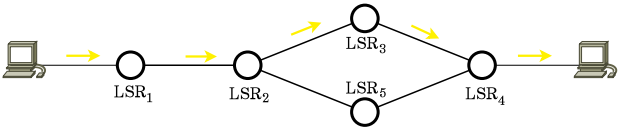
In una rete IP il guasto di un link genera un transitorio[6] in cui probabilmente abbiamo perdita di pacchetti. Con il *Fast Rerouting* MPLS fornisce una soluzione al problema creando un LSP secondario. Come mostra la Figura 1.4.2 il LSR collegato al link che ha subito il guasto, accorgendosi del malfunzionamento, instrada il pacchetto sul LSP secondario [7], garantendo una soluzione piú veloce ed elegante al problema.

1.4.3 Load Balancing

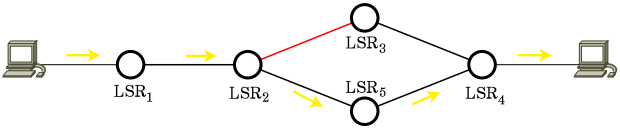
Considereiamo lo scenario rappresentato in Figura 1.12 e supponiamo che le sorgenti, collegate a LSR_1, producano traffico destinato al ricevitore collegato a LSR_4. Il protocollo IP instrada tutti i pacchetti sullo stesso percorso LSR_2 LSR_6, con MPLS è possibile creare un traffic trunk su LSR_1 LSR_2 che può essere ripartito in LSR_2 in tre traffic trunk e instradati a LSR_4. Così facendo abbiamo un risparmio di

[6] Determinato dal tempo necessario ai router per accorgersi del guasto, comunicare gli aggiornamenti a gli altri router e aggiornare la tabelle di forwarding.

[7] Il LSP secondario può essere calcolato sia a seguito del guasto che precalcolato, inoltre lo stesso LSP secondario può essere utilizzato, tramite il Label Stacking, per la protezione di piú LSP.



(a) In caso di corretto funzionamento i pacchetti usano l'LSP primario.



(b) In caso di guasto del link rosso i pacchetti sono reinstradati sull'LSP secondario.

Fig. 1.11: Esempio di Fast Rerouting.

informazioni di controllo e label sul link LSR_1 LSR_2 e un miglior bilanciamento delle risorse sulla seconda tratta.

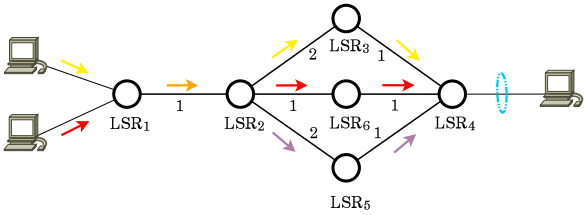


Fig. 1.12: Esempio di load Balancing.

2 RSVP-TE

Il protocollo *RSVP* (*Resource reSerVation Protocol*) [11] nasce alla fine degli anni novanta nell'ambito dell'architettura per la qualità del servizio *IntServ* (*Integrated Services*) per permettere una prenotazione delle risorse in modo simile a quanto avviene in una rete connection-oriented; consentendo a sorgenti, destinatari di traffico e router sul percorso di stabilire opportuni stati e modalità di trattamento dei pacchetti IP, all'interno dei router, al fine di supportare QoS per singoli microflussi di traffico. La modalità di funzionamento prevede che il terminale sorgente informi i destinatari sul tipo di traffico generato e sulle caratteristiche prestazionali di cui necessita in modo che il destinatario possa informare i router a monte sulla banda da riservare per ricevere il traffico con la data QoS.

È stato successivamente esteso con *RSVP-TE* (*Resource reSerVation Protocol-Traffic Engineering*) [5] per permettere la prenotazione di banda su un percorso precalcolato.

Nel seguito introduciamo questo protocollo e le sue estensioni infatti, dopo averne tracciato le caratteristiche di base

2.1 Caratteristiche principali

RSVP è *receiver-oriented*, ovvero sono i ricevitori e non i trasmettitori a stabilire l'ammontare delle risorse da richiedere ed a iniziare la prenotazione; favorendo la gestione del multicast e l'eterogeneità dei ricevitori. RSVP è inoltre indipendente dai protocolli di routing e *soft-state*, ovvero gli stati

delle prenotazioni hanno dei timer alla scadenza dei quali, se non sono rinfrescati, vengono cancellati.

Notiamo come RSVP, nella sua specifica originaria, si appoggi completamente ai tradizionali protocolli di routing IP per decidere i percorsi seguiti dai micro-flussi. Poiché i protocolli di routing IP sono basati sulla topologia della rete, non tengono generalmente in conto della distribuzione del traffico; inoltre potrebbero selezionare un percorso contenente router non RSVP e pertanto non in grado di elaborare messaggi RSVP.

Infine RSVP fornisce diversi stili di prenotazione, in dettaglio:

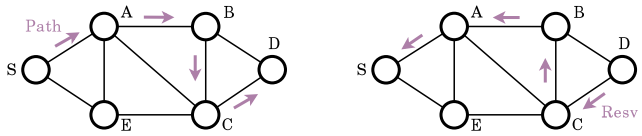
Fixed Filter (FF) : prevede una prenotazione distinta per ogni sorgente,

Shared Explicit (SE) : prevede una prenotazione condivisa da un insieme di sorgenti;

Wildcard Filter (WF) : prevede una singola prenotazione per rivevitore da condividere tra tutti le sorgenti.

2.2 Funzionamento base di RSVP-TE

Vediamo, tramite un esempio, il funzionamento di base del protocollo RSVP. Facciamo riferimento alla Figura 2.1a, supponiamo che il nodo S abbia traffico da inviare al nodo D e invii pertanto a questo un *Path Message* in cui, in particolare, inserisce il nodo attraversato nel campo *previous hop*, i *Sender TSpec* costituiti da rate e dimensione del bucket, IP del sender e porta del processo nel *Sender Template*. Processando questo messaggio ogni nodo costruisce una *path state block* (*psb*) e invia il messaggio verso il prossimo nodo per raggiungere la destinazione che, come mostra la Figura 2.1b, ricevuto il messaggio, può decidere di effettuare una prenotazione inviando un *Resv Message* verso il mittente che segue il percorso tracciato dal messaggio di Path. In particolare il messaggio di Resv contiene il *Filter Spec* (analogo al Sender Template) e le caratteristiche del serzio nel *Flow Spec*, formati da *RSpec* e *TSpec*. Ogni nodo sul percorso processa il messaggio di Resv e, se può soddisfare la richiesta,



(a) Invio del messaggio di Path.

(b) Invio del messaggio di Resv: D accetta la richiesta.

Fig. 2.1: Messaggi di path e resv.

manda avanti il processo fino a raggiungere la sorgente. Nel caso in cui un router non riesca a soddisfare la richiesta invia al ricevitore un messaggio di *ResvError*. Se il processo termina correttamente, la prenotazione viene registrata in tutti i router sul percorso tra sorgente e destinazione e la sorgente può iniziare ad inviare pacchetti con la certezza che siano riservate risorse sufficienti. Per rinnovare la prenotazione, il ricevitore deve inviare lo stesso Resv Message ogni 30 secondi, così da rinfrescare gli stati dei router.

Come indicato nella Figura 2.1, abbiamo ipotizzato che il messaggio di Path segua il percorso, fornito dal normale routing IP, formato dai nodi A, B, C, e D.

Supponiamo adesso di utilizzare l'estensione TE e di voler fare in modo che l'LSP sia formato dai nodi A, E, C e D. Il nuovo messaggio di Path contiene adesso, tra le altre cose, i nuovi oggetti:

- LABEL_REQUEST che serve a informare gli LSR sul percorso sulla necessità di associare una label MPLS al traffico del tunnel TE;

- l'EXPLICIT_ROUTE Object (ERO) che contiene la lista degli LSR da attraversare.

Il messaggio di Path può, opzionalmente, contenere un oggetto RECORD_ROUTE che conserva il percorso effettuato per successive operazioni di notifica, e rivelazione di loop e un oggetto SESSION_ATTRIBUTE per informazioni addizionali quali preemption, priorità, protezione e diagnostica. Nell' esempio proposto l'ERO che S invia al nodo A contiene

il percorso A_B_C_D. Il nodo A genera a sua volta un messaggio e lo invia al nodo B, dopo aver eliminato dall'ERO il proprio identificativo. Una volta che il nodo D ha ricevuto il messaggio di Path, genera un messaggio di Resv che invia al PHOP, ovvero all'interfaccia da cui ha ricevuto il messaggio di Path. Il messaggio di resv contiene, tra altre cose, un nuovo oggetto denominato LABEL dove viene memorizzata la label associata al traffico. Dopo che il messaggio di Resv ha raggiunto S, le tabelle FTN e ILM sono state impostate e può iniziare l'invio del traffico. Notiamo come RSVP-TE utilizzi una modalità di distribuzione delle associazioni FEC-label di tipo downstream-on-demand ordered control.

Oltre ai messaggi già visti, abbiamo:

- *PathErr*: per indicare un errore in risposta ad un messaggio di path, segue il percorso verso la sorgente,
- *PathTear*: per abbattere i path state di una prenotazione, dalla sorgente al destinatario,
- *ResvTear*: per abbattere i resv state di una prenotazione, dal destinatario alla sorgente;
- *ResvConf*: per la conferma della prenotazione, dalla sorgente al destinatario.

Sottolineamo infine che, poiché il traffico che transita lungo un LSP è definito tramite la label applicata dall'ingress, i path risultanti possono essere trattati come tunnel nascosti sotto il normale routing IP. Questo è uno degli aspetti piú interessanti di RSVP-TE infatti mentre l'applicazione, originaria di RSVP e del modello IntServ, a singoli micro-flussi conduceva a problemi di scalabilità, adesso la definizione di flusso è molto flessibile. Inoltre, poiché le label sono associate ai flussi di traffico, i router possono determinare il giusto reservation state basandosi esclusivamente sul valore della label, semplificando le varie operazioni.

2.3 Estensione di RSVP-TE per LSP P2MP

In questo paragrafo presentiamo una estensione di RSVP-TE per la gestione di LSP punto-multipunto (P2MP) [10] che è possibile facendo in modo che nodi non ingress della rete possano replicare il traffico verso piú destinazioni. Un LSP P2MP è formato da piú *source-to-leaf* (*S2L*) *sub-LSP* che vengono instaurati tra l'ingress e i LER utilizzando uno o piú messaggi di Path. Un tunnel P2MP TE è identificato univocamente da un oggetto P2MP_SESSION che contiene un P2MP Identifier (P2MP ID), un Tunnel Identifier (TID) e un Extended Tunnel Identifier (E-TID). Il P2MP ID è un intero di 32 bit unico all'interno dell'ingress e la tripletta <P2MP ID, TID, E-TID> rappresenta in modo univoco il tunnel P2MP. Un LSP P2MP unisce S2L sub-LSP multipli e pertanto potrebbe non essere possibile rappresentarne il suo completo stato in un singolo pacchetto IP, inoltre potrebbe essere richiesta la capacità di inserire o rimuovere LSP. Per tenere conto di queste necessità sono stati introdotti due campi addizionali, un sub-group identifier (Sub-Group ID) e un sub-group originator (Sub-Group Origininator ID), che insieme prendono il nome di campi Sub-Group, negli oggetti SENDER_TEMPLATE e FILTER_SPEC. Un S2L sub-LSP esiste solo all'interno del contesto di un LSP P2MP e per segnalare, opzionalmente, un percorso esplicito di un S2L sub-LSP viene usato un EXPLICIT_ROUTE Obj. (ERO) o un SECONDARY_EXPLICIT_ROUTE Obj. (SERO). Vediamo, tramite un esempio, come avviene la segnalazione di un LSP P2MP utilizzando un singolo messaggio di Path. Facciamo riferimento alla Figura 2.2 in cui gli egress sono i nodi F, H, M, O, Q, R e l'ingress è il nodo A. Per evitare l'invio di informazioni ridondanti per quei S2L sub-LSP che condividono nodi, si compie una compressione dei percorsi in modo da segnalare solo le differenze. In tal caso il messaggio che A invia a B contiene gli oggetti S2L sub-LSP e ERO/SERO riportati in Tabella 2.1. Il nodo B non fa altro che rigenerare il messaggio di Path eliminando, eventualmente, il suo identificativo dai percorsi espliciti; il messaggio che il nodo

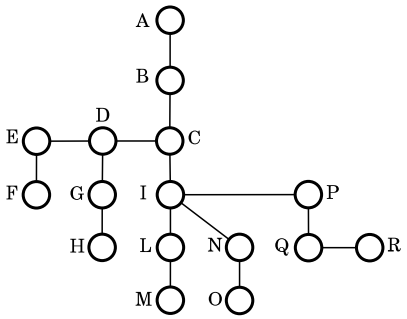


Fig. 2.2: Esempio di LSP P2MP.

S2L sub-LSP	ERO/SERO
F	ERO = {B, C, D, E, F}
H	SERO = {D, G, H}
M	SERO = {C, I, L, M}
O	SERO = {I, N, O}
Q	SERO = {I, P, Q}
R	SERO = {Q, R}

Tab. 2.1: Messaggio dal nodo A al nodo B.

S2L sub-LSP	ERO/SERO
F	ERO = {D, E, F}
H	SERO = {D, G, H}

Tab. 2.2: Messaggio dal nodo C al nodo D.

S2L sub-LSP	ERO/SERO
M	SERO = {I, L, M}
O	SERO = {I, N, O}
Q	SERO = {I, P, Q}
R	SERO = {Q, R}

Tab. 2.3: Messaggio dal nodo C al nodo I.

C invia a D contiene gli oggetti S2L sub-LSP e ERO/SERO riportati in Tabella 2.2, mentre quello che invia a I è riportato in Tabella 2.3. Il nodo D invia due messaggi, al nodo E

S2L sub-LSP	ERO/SERO
F	ERO = {E, F}

Tab. 2.4: Messaggio dal nodo C al nodo D.

e al nodo G, la cui composizione è riportata rispettivamente in Tabella 2.4 e in Tabella 2.5. Gli altri nodi a valle, ricevuti

S2L sub-LSP	ERO/SERO
H	ERO = {G, H}

Tab. 2.5: Messaggio dal nodo C al nodo D.

i messaggi, non fanno altro che processarli in modo analogo a quanto visto fino ache non siano stati raggiunti tutti gli egress

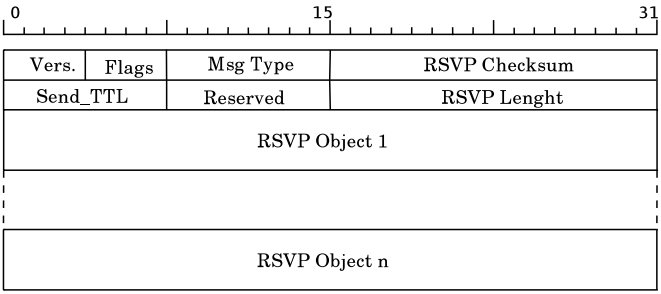


Fig. 2.3: Formato del messaggio RSVP.

Prima di vedere i nuovi oggetti, la Figura 2.3 riporta la struttura generale dei messaggi RSVP; abbiamo i campi:

Vers. (4 bit): contiene il numero corrispondente alla versione del protocollo,

Flags (4 bit): il suo significato non è specificato, tuttavia i valori tra 0x01 e 0x08 sono riservati,

Msg Type (8 bit): Indica il tipo di messaggi,

RSVPChecksum (16 bit): contiene il valore della checksum per rivelare eventuali errori,

Send_TTL (8 bit): contiene il valore del TTL,

Reserved (8 bit): bit risercati;

RSVP Lenght (16 bit): contiene la lunghezza, in byte, del messaggio RSVP comprensivo dell'header comune e degli oggetti, a lunghezza variabile, che seguono.

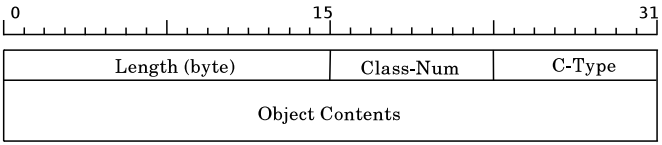


Fig. 2.4: Formato degli oggetti RSVP.

Un messaggio RSVP è formato da piú oggetti, a lunghezza variabile, la cui forma generale è riportata in Figura 2.4.

Lenght (16 bit): indica la lunghezza in byte dell'oggetto. É sempre un multiplo di quattro e minimo pari a quattro,

Class-Num (8 bit): indica la classe dell'oggetto;

Ctype (8 bit): all'interno di un Class-Num identifica il particolare oggetto.

Nuovi oggetti

Abbiamo fatto riferimento a nuovi oggetti introdotti in RSVP-TE per la gestione di LSP punto-multipunto, andiamo adesso a illustrarli in modo piú dettagliato, dopo aver mostrato la nuova struttura dei messaggi di Path e Resv.

La forma del messaggio di Path è la seguente.

<Path Message>::=	<Common Header> [<INTEGRITY>] [[<MESSAGE_ID_ACK> | <MESSAGE_ID_NACK>] ...] [<MESSAGE_ID>] <SESSION> <RSVP_HOP> <TIME_VALUE> [<EXPLICIT_ROUTE>] <LABEL_REQUEST> [<PROTECTION>] [<LABEL_SET> ...] [<SESSION_ATTRIBUTE>] [<NOTIFY_REQUEST>] [<ADMIN_STATUS>] [<POLICY_DATA> ...] <sender descriptor> [<S2L sub-LSP descriptor list>]

Nel caso in cui si voglia segnalare uno o piú S2L sub-LSP, questo è possibile includendo nel messaggio la lista dei descrittori dei singoli S2L sub-LSP. La lista dei descrittori di S2L sub-LSP ha la forma riportata di seguito.

<S2L sub-LSP descr. list>::=	<S2L sub-LSP descr.> [<S2L sub-LSP descr. list>]

in cui:

<S2L sub-LSP descr.>::=	<S2L_SUB_LSP> [<SERO>]

La forma del messaggio di Resv è riportata di seguito.

<Resv Message>::=	<Common Header> [<INTEGRITY>] [[<MESSAGE_ID_ACK> | <MESSAGE_ID_NACK>] ...] [<MESSAGE_ID>] <SESSION> <RSVP_HOP> <TIME_VALUE> [<RESV_CONFIRM>] [<SCOPE>] [<NOTIFY_REQUEST>] [<ADMIN_STATUS>] [<POLICY_DATA> ...] <STYLE> <flow descriptor list>

Vediamo adesso la struttura dei nuovi oggetti, presentando solo la versione relativa a IPv4.

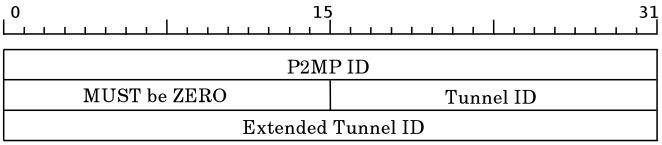


Fig. 2.5: P2MP SESSION Object.

SESSION Object La struttura del nuovo oggetto SESSION, riportata in Figura 2.5, è molto simila a quella per la gestione di percorsi punto-punto, tuttavia l'indirizzo di destinazione è sostituito con l'identificativo del percorso punto-multipunto, il P2MP ID. I nuovi C-Type sono 13 per IPv4 e 14 nel caso IPv6. Tutti i S2L sub-LSP che appartengono allo stesso percorso punto-multipunto condividono lo stesso oggetto SESSION, che definisce pertanto il tunnel P2MP. La combinazione degli oggetti SESSION, SENDER_TEMPLATE e S2L_SUB_LSP identifica ciascun S2L sub-LSP.

Il P2MP ID è un identificativo del tunnel P2MP, unico all'interno dell'ingress. Il Tunnel ID è un identificativo che rimane costante durante tutta la vita del tunnel P2MP, infine l'Extended Tunnel ID è utilizzato, assieme agli altri due campi, dagli ingress che desiderano avere un identificativo globalmente unico del tunnel P2MP e contiene l'indirizzo del sender.

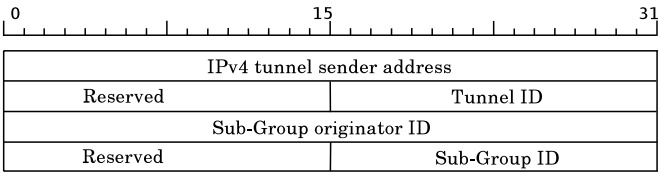


Fig. 2.6: P2MP SENDER_TEMPLATE Object.

SENDER_TEMPLATE Object La struttura del nuovo oggetto SENDER_TEMPLATE è mostrata in Figura 2.6. I nuovi C-Type sono 12 nel caso si utilizzi IPv4 e 13 nel caso IPv6. Il LSP ID può cambiare per permettere la condivisione di risorse, infatti, poiché è possibile avere piú istanze di tunnel P2MP, ciascuna è caratterizzata da un proprio LSP ID e i S2L sub-LSP che corrispondono ad una particolare instanza condividono lo stesso identificativo. Per distinguere diversi messaggi di Path che segnalano lo stesso P2MP si utilizza la coppia Sub-Group Originator ID e Sub-Group ID, il primo contenente l'indizzo del router che genera il messaggio e il secondo utilizzato per differenziare i vari messaggi.

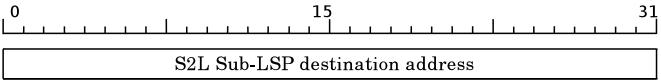


Fig. 2.7: P2MP S2L_SUB_LSP Object.

S2L_SUB_LSP Object L'oggetto S2L_SUB_LSP è mostrato in Figura 2.7. Il numero di classe attribuito a tale oggetto è il 50, i C-Type sono 1 nel caso si utilizzi IPv4 e 2 nel caso IPv6. L'unico campo dell'oggetto contiene l'indirizzo dell'egress del sub-LSP.

È stato introdotto anche un nuovo oggetto FILTER_SPEC il cui C-Type è 12 per IPv4 e 13 per IPv6 la cui struttura non viene riportata perché identica a quella dell'oggetto SENDER_TEMPLATE.

Per concludere, in Tabella 2.6 sono riportati tutti i nuovi oggetti con i relativi numeri di classe e C-Type.

OGGETTO	Class-Num	C-Type (IPv4/IPv6)
SESSION	1	13/14
SENDER_TEMPLATE	11	12/13
FILTER_SPEC	10	12/13
S2L_SUB_LSP	50	1/2

Tab. 2.6: Tabella riassuntiva dei nuovi oggetti introdotti.

3 Network Simulator 2

In questo capitolo diamo una breve introduzione al *Network Simulator 2* (*NS2*), per maggiori informazioni si rimanda a [8]. NS è un simulatore di reti ad eventi discreti che utilizza due linguaggi: il C++ basato sulla gerarchia di classi denominata compilata e il Tcl/OTcl basato sulla gerarchia di classi interpretata. In breve, con uno script Otcl configuriamo la simulazione e gestiamo delle classi, implementate in C++ e organizzate in maniera gerarchica, che costituiscono la base del simulatore. NS è nato nel 1989 come variante del simulatore REAL ed è stato sviluppato dai ricercatori del progetto VINT (Virtual InterNetwork Testbed), supportato da DARPA (Defense Advanced Research Projects Agency), con la collaborazione di UC Berkeley, LBL, USC/ISI, e Xerox PARC.

Il capitolo è organizzato come segue: nel Paragrafo 3.1 vediamo come creare una simulazione. Nel Paragrafo 3.2 presentiamo MNS, un'estensione del simulatore che implementa il protocollo MPLS e nel Paragrafo 3.3 introduciamo l'estenzione RSVP-TE.

3.1 Creare una simulazione

Il codice che permette a NS2 di interfacciarsi con l'interprete OTcl risiede nella directory `tclcl`, separata da quella del resto del simulatore. La directory dell'interprete comprende diverse classi e nel seguito focalizzeremo la nostra attenzione sulle seguenti:

- `class Tcl`: contiene i metodi utilizzati dal C++ per accedere all'interprete,
- `class TclObject`: è la classe base per tutti gli oggetti del simulatore che hanno un corrispondente oggetto nella gerarchia compilata,
- `class TclClass`: definisce la gerarchia di classi interpretata ed i metodi che permettono all'utente di instanziare oggetti `TclObject`,
- `class TclCommand`: definisce comandi globali per l'interprete,
- `class EmbeddedTcl`: contiene i metodi per caricare comandi pre-assemblati ad alto livello che semplificano la simulazione;
- `class InstVar`: permette di accedere alle variabili membro del C++ per mezzo di variabili instanziate da Tcl.

Una classe fondamentale di NS2 è quella `Simulator` [12], i cui metodi ci permettono di configurare la simulazione e schedulare gli eventi. Possiamo creare un'istanza della classe con il comando:

```
set ns [new Simulator]
```

in cui la variabile `ns` punta all'istanza. I metodi della classe `Simulator` possono essere divisi in tre categorie:

1. Metodi che configurano la topologia della rete,
2. Eventi che permettono di registrare l'evoluzione temporale;
3. Metodi che consentono di configurare lo scheduler degli eventi.

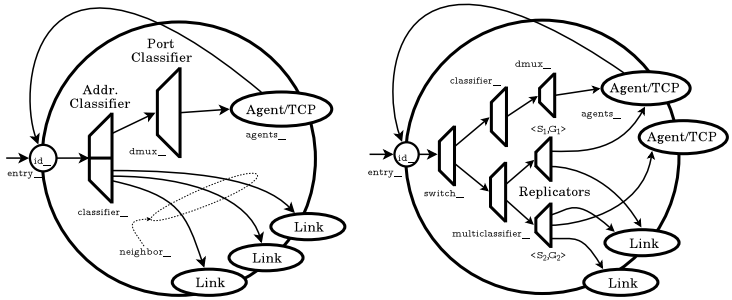
Altre classi utilizzate dal simulatore, che introduciamo nei prossimi paragrafi, sono: *Node*, *Link*, *Agent*, *Application* e la classe *Trace*.

3.1.1 Nodi e Link

Ogni nodo in *ns*è parte di una classe OTcl e gestisce le funzionalità del protocollo IP, definendo la struttura dell'indirizzamento usato e trasmettendo i pacchetti sui link prestabiliti o consegnandoli al protocollo di trasporto. Dopo aver instanziato una simulazione, possiamo creare due nodi scrivendo:

```
set n0 [ $ns node]
set n1 [ $ns node]
```

CosÃň facendo associamo a **n0 n1** due oggetti ritornati dal metodo **node** della classe **Simulator**. In Figura 3.1a è riportata l'architettura di un nodo unicast. Notiamo che **entry_** memorizza un riferimento all'istanza del nodo e **classifier_** contiene un riferimento ai classifier. Se la simulazione utilizza il multicast, come mostrato in Figura 3.1b, nei nodi sono presenti dei replicatori e uno **switch_** che permette di inoltrare i pacchetti al classifier appropriato, infatti in caso di simulazione multicast, se il primo bit del pacchetto è uno, il pacchetto è multicast, altrimenti unicast. Tutti i nodi



(a) Architettura nodo unicast. **(b)** Architettura nodo multicast.

Fig. 3.1: Architettura di un nodo NS2.

contengono:

- Un indirizzo o **id_** che cresce monotonicamente da uno,
- La lista **neighbor_**,
- La lista **agent_**,

- Un identificatore del tipo di nodo, `nodetype_`;
- Un modulo di routing.

Possiamo adesso unire i due nodi con un link unidirezionale tramite il comando:

```
$ns simplex-link $n0 $n1 10Mb 15ms DropTail
```

che crea un link di capacità 10 Mbps, ritardo di propagazione 15 ms e disciplina FIFO[1]. L'architettura di un link, riportata in Figura 3.2, contiene in particolare i seguenti oggetti:

- `head_`: puntatore al primo oggetto del link,
- `queue_`: puntatore all'oggetto coda della classe link,
- `link_`: punta all'oggetto che simula il comportamento del link in termini di ritardo di propagazione e capacità,
- `ttl_`: punta all'oggetto che gestisce il campo TTL del pacchetto;
- `drophead_`: punta all'oggetto coda.

Gli ulteriori elementi riportati in Figura 3.2 sono necessari sia alla visualizzazione della simulazione con NAM che per la raccolta di dati. Se è necessario collegare i due no-

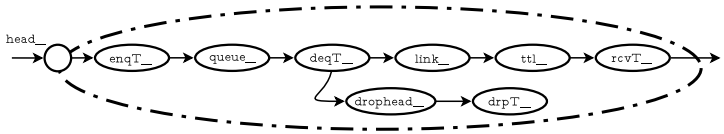


Fig. 3.2: Architettura di un link.

di in entrambe le direzioni basta sostituire `simplex-link` con `duplex-link`. Possiamo stabilire un valore `<queue_limit>` di pacchetti che potranno essere messi in coda su un link da `n0` a `n1`:

[1]Altre possibili discipline di servizio sono: RED (Random Early Discard), FR (Fair Queueing), DRR (Deficit Round Robin), SFQ (Stochasting Fair Queueing) e CBQ.

```
$ns queue-limit $n0 $n1 <queue\_limit>
```

È possibile assegnare una latenza di `<time_interval>` secondi a un link da `n0` a `n1`:

```
$ns delay $n0 $n1 <time_interval>
```

Si può assegnare a un link un costo `<cost_val>` scrivendo:

```
$ns cost $n0 $n1 <cost_val>
```

in cui `<cost_val>` è un numero intero; per default tutti i link hanno costo pari a 1.

3.1.2 Agent e Application

A questo punto per generare e ricevere pacchetti abbiamo bisogno di dotare i nodi nodi di oggetti Agent, che inseriamo scrivendo:

```
#Define Agent
set udp1 [ new Agent/UDP]
set sink1 [ new Agent/Null]
$udp1 set packetSize_ 1500
#Attach Agent and connect
$ns attach-agent $n0 $udp1
$ns attach-agent $n1 $sink1
$ns connect $udp1 $sink1
```

Oltre all'agente UDP sono disponibili agenti TCP che possiamo classificare come:

- Agenti TCP *unidirezionali*: si dividono ulteriormente in trasmettitori e ricevitori di vari tipi che implementano altrettante versioni con diverse opzioni per il congestion control e il flow control;
- Agenti TCP *bidirezionali*: abbiamo un'unica classe TCP bidirezionale, quella `Agent /TCP/FullTcp` che consente la trasmissione di dati in entrambe le direzioni e implementano la versione RENO del protocollo.

Entrambi gli agenti contengono diverse variabili membro che possiamo utilizzare per modificarne la configurazione.

Gli oggetti della classe `Application` si dividono in:

- *Traffic generator*: riproduce alcuni tipi di traffico, attualmente sono disponibili 4 classi:
  (i) `EXPOO_Traffic`: genera traffico secondo un modello ON/OFF con tempi di permanenza distribuiti esponenzialmente,
  (ii) `POO_Traffic`: i tempi di permanza seguono una distribuzione di Pareto,
  (iii) `CBR_Traffic`: genera traffico a rate costante;
  (iv) `TrafficTrace`: permette di generare pacchetti in base a dati contenuti in un file di traccia.
- *Simulated applications*: simulano il comportamento di alcune applicazioni reali, attualmente sono implementate `Application/FTP` e `Application/Telnet`.

Vediamo come creare un agente di traffico esponenziale:

```
set VBR1 [new Application/traffic/Exponential]
$VBR1 set rate_ 1600 kb
$VBR1 set packetSize 1000
$VBR1 set burst_time_ 0.2
$VBR1 set idle_time_ 2.0
$VBR1 attach-agent $udp1
#Attiviamo e disattiviamo la sorgente
$ns at 0.05 "VBR1 start"
$ns at 20 "VBR1 stop"
```

Per generare traffico di Poisson è sufficiente impostare:

```
$VBR1 set burst_time_ 0
$VBR1 set idle_time_ 1/lambda
```

con `lambda` pari alla frequenza di generazione dei pacchetti.

Notiamo come in NS2, per alleggerire la simulazione, un'applicazione non genera dati ma solo la dimensione del pacchetto corrispondente e il ricevitore non fa altro che scartare i pacchetti e inviare un eventuale riscontro.

3.1.3 Tracing

Spesso simulando una rete abbiamo bisogno di prelevare dei dati, a questo scopo NS2 mette a disposizione due tipologie di oggetti:

- *trace*: vengono solitamente creati e inseriti tra due nodi, associando un canale è possibile annotare gli eventi avvenuti durante la simulazione;
- *monitor*: permette di registrare, usando dei contatori, su un qualunque canale dei parametri globali come il numero di pacchetti trasmessi o quelli ricevuti.

Vediamo un esempio in cui, dopo aver creato due nodi, apriamo il file `traccia.tr` a cui `trace_file` punta. Il file di traccia è creato con il comando `trace-all`.

```
set ns [new Simulator]
set s1 [$ns node]
set d1 [$ns node]
$ns duplex-link 10Mb 10ms DropTail
set trace_file [open traccia.tr w]
$ns trace-all $s1 $d1 $trace_file
```

3.1.4 Altri comandi

Affinché simulazioni distinte diano risultati incorrelati è necessario utilizzare un generatore di numeri casuali con seme diverso per ogni simulazione. Questo si realizza con il comando:

```
ns-random 60
```

che setta il seme di numeri casuali a 60.

I file generati nella simulazione devono essere chiusi; i comandi e le operazioni che chiudono uno script vengono inseriti nella procedura `finish{}` che, ad esempio, assume la forma:

```
proc finish {} {
 global ns nam_file trace_file
 $ns flush-trace
 close $nam_file
 close $trace_file
 exit 0
}
```

A questo punto per far partire la simulazione basta digitare il comando:

```
$ns run
```

3.2 MNS (MPLS Network Simulator)

Il modulo MNS (MPLS Network Simulator), creato dai ricercatori della Southern California University Gaeil Ahn e Woojik Chun [6], implementa il protocollo MPLS nelle sue funzionalità principali, in dettaglio:

- Associazioni tra classi di traffoco e label in modalità control-driven e data-driven con distribuzione downstream in modalità control-driven e sia downstream che upstream per la modalità data-driven,
- Distribuzione independent-mode in modalità control-driven e sia independent che ordered-mode in modalità data-driven,
- Informazioni di binding di tipo conservative label retention mode;
- Possibilità di creare LSP espliciti.

Per gestire queste funzionalità sono stato aggiunti a NS2 gli elementi di Figura 3.3, ovvero:

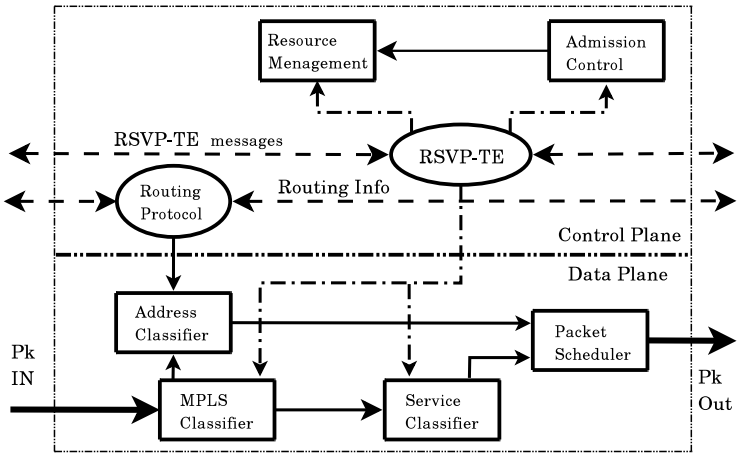


Fig. 3.3: Modello di MNS.

- LDP/CR-LDP: genera e processa i messaggi del protocollo di distribuzione delle label,
- MPLS Classifier: esegue le operazioni tipiche del label switching (swap, pop, push),
- Service Classifier: associa i pacchetti MPLS alla corretta reservation, processando l'header MPLS,
- Admission Control: controlla se il nodo MPLS in esame ha risorse sufficienti a garantire le specifiche di QoS richieste,
- Resource Managent: gestisce le code;
- Packet Scheduler: gestisce i pacchetti nelle code cosÃň da ricevere la QoS stabilita.

L'architettura di un nodo MPLS, riportata in Figura 3.4, comprende:

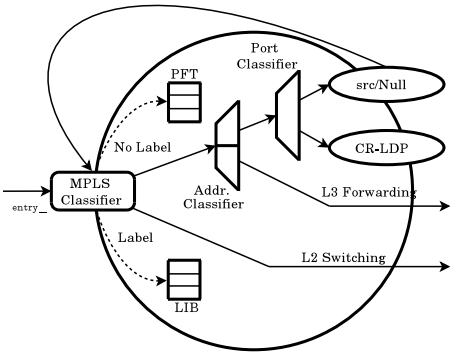


Fig. 3.4: Architettura di un nodo MPLS.

MPLS Classifier : se il pacchetto ricevuto ha una label viene eseguito lo switching a livello 2 (tabella LIB), in caso contrario (tabella PFT) se il pacchetto va associato ad un LSP gli viene aggiunto un header MPLS e inoltrato tramite switching di livello 2; altrimenti tramite routing di livello 3,

Address Classifier : esegue il forwarding di livello 3;

Port Classifier : se il nodo è la destinazione il pacchetto viene passato all'agent di livello superiore.

3.3 RSVP/ns

RSVP/ns [9] è il modulo che implementa il protocollo RSVP. Creato dal ricercatore dell'Università di Bonn Marc Greis, presenta quasi tutti gli oggetti di RSVP, sia nel formato per IPv4 che per IPv6, come mostra la Tabella 3.1. Ci sono alcune piccole differenze tra gli oggetti implementati in RSVP/ns e gli oggetti reali definiti da RSVP; per esempio, nell'oggetto SESSION di RSVP/ns sono omessi la porta di destinazione e l'identificativo del protocollo, visto che nella simulazione le sessioni sono identificate univocamente dal loro flow-id, mentre per semplificare il processing dei pacchetti è stato aggiunto un campo con l'indirizzo del destinatario. Gli unici

Object	Class-Num
SESSION	1
RSVP_HOP	3
TIME_VALUES	5
ERROR_SPEC	6
STYLE	8
FLOWSPEC	9
FILTER_SPEC	10
SENDER_TEMPLATE	11
RESV_CONFIRM	15

Tab. 3.1: Oggetti di RSVP/ns.

oggetti trascurati in RSVP/ns sono INTEGRITY (nella simulazione non possono esserci messaggi corrotti da utenti maliziosi), SCOPE (non è implementato lo stile di prenotazione WF), ADSPEC (tralasciato per semplicità) e POLICY_DATA (non è implementato alcun meccanismo di policing). Per quanto riguarda i messaggi, sono stati tutti implementati, tranne il PathErr Message: la sua unica funzione nel mondo

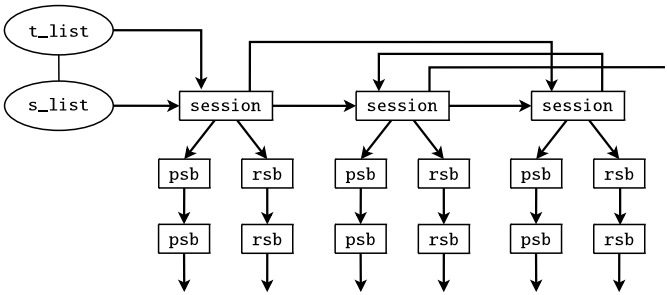


Fig. 3.5: Struttura di session e timer list.

reale, almeno in RSVP, è segnalare errori nati da mismatch coi numeri di porta, ma NS2 non utilizza le porte, rendendone così inutile la realizzazione. Altre limitazioni in RSVP/ns sono la presenza di un solo stile di prenotazione (FF) invece dei tre stabiliti dall'RFC e, nell'ambito degli Integrated Services, della sola classe controlled-load.

Gli agent RSVP conservano i path e i reservation state su tutti i nodi RSVP e generano e processano i messaggi. In ogni agent è presente una struttura **`session`** per ogni sessione: essa contiene vari campi, tra cui i puntatori alle liste di **`psb`** e **`rsb`**, un flag che segnala il refresh e un flag che segnala la volontà di inviare un ResvConf Message in risposta ad un Resv Message. Le strutture **`psb`** e **`rsb`** ricalcano quasi fedelmente la composizione indicata dall'RFC. Per le sessioni ci sono due liste, come mostra la Figura 3.5 tutte le sessioni sono contenute in una lista, denominata **`s_list`** e tutte le sessioni che devono essere rinfrescate sono in una **`t_list`**, posta in ordine cronologico di refresh. Il link per RSVP si basa su un **`duplex-intserv-link`**, al quale sono stati aggiunti un oggetto **`RSVPChecker`** (che ha il compito di intercettare i pacchetti RSVP sul collegamento e inviarli all'agent RSVP del nodo all'estremo del link) e uno scheduler WFQ (che usa gli algoritmi WFQ e WF2Q, con una coda best effort destinata a tutti i pacchetti che il classifier non riconosce e, grazie all'opzione "borrow", anche ai pacchetti delle varie classi che superano i limiti imposti).

Il comando per la creazione del link è:

```
$ns duplex-rsvp-link $n1 $n2 40Kb 10ms 0.8 1000 500 Param Null
```

che crea un collegamento bidirezionale tra i nodi `n1` e `n2` con banda pari a 40 Kbps e tempo di propagazione pari a 10 ms. Il parametro 0.8 sta ad indicare che l'80% della banda può essere allocata ai flussi RSVP e il restante 20% è destinato al traffico best effort. Inoltre 1000 bps sono riservati ai messaggi di segnalazione RSVP e la dimensione del buffer del link è fissata in 500 byte. Infine sono indicati l'algoritmo di admission control e lo stimatore utilizzato per tale algoritmo, selezionabili tra una lista di opzioni.

Concludiamo la descrizione del modulo RSVP/ns con un elenco di comandi messi a disposizione e una breve spiegazione della funzione svolta da ogni comando. Tra parentesi sono riportati gli argomenti di ognuno.

1. `add-rsvp-agent ( node )` : carica l'agent RSVP sul nodo indicato,
2. `get-rsvp-agent(node)` : restituisce un riferimento per l'agent RSVP del nodo indicato,
3. `session(rsvp_a, dst, fID)` : crea una sessione con i parametri indicati,
4. `release(rsvp_a, sID)` : rilascia una sessione;
5. `sender(rsvp_a, sID, rate, bucket, TTL)` : manda un essaggio di Path per segnalare la creazione di una sessione e i suoi parametri,
6. `reserve(rsvp_a, sID, style, flow des list)` : manda un messaggio di Resv in risposta al messaggio di Path e indica, tra gli altri parametri, la banda da allocare;
7. `confirm(rsvp_a, sID)` : manda un oggetto Resv_Conf col successivo messaggio di prenotazione,
8. `sessions(rsvp_agent)` : fornisce una lista di tutte le sessioni nell'agent,
9. `set-status(rsvp_a, sID, val)` : assegna un valore allo status della sessione,

10. `get-status(rsvp_a, sID)` : fornisce il valore dello stato della sessione,
11. `set-handle(rsvp_a, sID, obj)` : assegna un riferimento a quell'oggetto di quella sessione;
12. `get-handle(rsvp_a, sID)` : fornisce un riferimento a quell'oggetto per quella sessione.

3.3.1 RSVP-TE/ns

RSVP-TE/ns [4] è un modulo, sviluppato all'università di Pisa che implementa il protocollo RSVP-TE. Il modulo RSVP-TE/ns introduce diverse funzionalità che possono di fatto essere suddivise in quattro categorie:

1. LSP establishment,
2. LSP release,
3. Failure Handling (notifica ed abbattimento dell'LSP);
4. Strategie di LSP recovery.

vediamo adesso piú in dettaglio le varie funzionalità, descrivendo i comandi necessari.

LSP establishment Esistono due diversi comandi per la creazione di un LSP; con:

```
<Ingress> create-erlsp-rsvpte <Egress-LSR> <SID> <FlowID>
<TunnelID> <Er> <DiffServ Object> <Class Type>
```

si ha la possibilità di creare un LSP di tipo Explicit Route, mentre con:

```
<Ingress> create-erbwlsp-rsvpte <Egress-LSR> <SID> <FlowID>
<TunnelID> <Er> <DiffServ Object> <Class Type>
```

possiamo anche effettuare una prenotazione della banda. In entrambi i casi il messaggio di Path contiene, oltre ai classici oggetti RSVP, un oggetto LABEL_REQUEST utilizzato per richiedere ai nodi successivi l'allocazione delle etichette. Il simulatore prevede la possibilità di creare un LSP che segua un percorso esplicito specificato dal nodo di ingresso;

il campo <Er> contiene proprio l'indirizzo (id) degli LSR che dovranno essere attraversati e è utilizzato per creare l'EXPLICIT_ROUTE (ERO) da aggiungere al messaggio di Path. Quando un LSR riceve un messaggio di Path processa l'ERO e ne inserisce il contenuto nella psb per fare in modo che i successivi messaggi di Resv seguano lo stesso percorso, in senso contrario, di quelli di Path. Quando il messaggio arriva all'egress, questo non fa altro che fissare un nuovo valore per l'etichetta da comunicare al nodo precedente e, nel caso sia stata richiesta anche l'allocazione della banda, richiamare il Resource Manager per eseguire l'operazione di allocazione. Viene così creato un messaggio di Resv con il giusto campo LABEL ed inviato al nodo precedente secondo il percorso individuato dai precedenti messaggi di Path. Quando il nodo ingress riceve il messaggio, l'LSP è stato creato.

LSP release Oltre alla creazione di un LSP viene simulato anche il suo abbattimento con il conseguente rilascio della banda eventualmente allocata. Questo si realizza tramite il seguente comando:

```
<Ingress> release-LSP <SID> <FlowID>
```

In questo caso il nodo di ingresso invia all'egress un messaggio di Path Tear con il conseguente rilascio di tutte le risorse allocate per lo specifico LSP; inoltre, nelle tabelle vengono cancellate le righe relative all'LSP abbattuto.

Failure Handling All'interno di una rete è possibile che il collegamento fra due nodi non sia piú utilizzabile a causa, ad esempio, della rottura di un link. È possibile simulare un simile evento attraverso il comando:

```
<Up node> break-link  <Down node> <ID down node> <ID up node>
```

Vediamo come reagisce RSVP-TE alla caduta di un link. Un messaggio di Path Error viene spedito, per ogni LSP stabilito sul link rotto, in direzione upstream dall'Agent RSVP dell'`<Up node>` verso l'ingress LSR per comunicargli la caduta del link; stessa cosa viene fatta dall'Agent RSVP del `<Down node>` in direzione downstream inviando, però, un messaggio di Resv Error. Ricevuti i messaggi i router di ingresso ed

uscita inviano rispettivamente un messaggio di Path Tear e Resv Tear per comunicare l'abbattimento dell'LSP e cancellano dalle tabelle LIB, PFT e ERB le righe relative all'LSP abbattuto.

Strategie di LSP recovery Il modulo prevede due possibili strategie di rerouting consecutive alla caduta di un link: con la prima si ha una preallocazione di un LSP di backup, con la seconda il calcolo del percorso alternativo viene effettuato preventivamente, ma la sua allocazione è consecutiva all'abbattimento dell'LSP primario. Per selezionare la prima ipotesi si utilizza il comando:

```
<Ingress> reroute-prealloc <Source> <Egress> <Dst> <OldSID> <SID>
<FlowID> <TunnelID> <Rate> <Bucket> <TTL> <Er>
```

Al contrario, se siamo interessati alla seconda possibilità il comando da utilizzare è:

```
<Ingress> reroute-precomp <Source> <Egress> <Dst> <OldSID>
<FlowID> <TunnelID> <Rate> <Bucket> <TTL> <Er>
```

4 Modulo NS2 per la creazione di LSP P2MP

In questo capitolo vediamo l'estensione che abbiamo implementato al protocollo RSVP-TE per la gestione di LSP P2MP realizzata utilizzando il simulatore NS2. Descriviamo inoltre le due euristiche utilizzate per il calcolo dell'albero multicast. Al fine di realizzare la nostra estensione siamo partiti dalla versione 2.26 di Network Simulator, su cui abbiamo istallato il modulo che implementa l'MPLS (MNS, versione 2 per ns 2-26) e il modulo che implementa RSVP-TE (RSVP-TE/ns, patch per ns 2-26) e quello che abbiamo fatto è stato estendere il modulo RSVP-TE/ns con gli oggetti introdotti dal nuovo protocollo e permettere che il modulo MNS interagisca con esso per la distribuzione delle label. Per realizzare il modulo sono stati modificati molti file di RSVP-TE/ns e di MNSv2, cambiando delle funzioni o aggiungendone interamente delle altre. Sono state anche aggiunte due funzioni al modulo che implementa il protocollo di routing OSPF così da calcolare i percorsi punto-multipunto seguendo le due euristiche scelte.

Il capitolo è organizzato come segue: nella prima parte descriviamo in termini qualitativi le modifiche apportate e tutte le parti aggiunte, rimandando all'Appendice A per la lista completa dei file modificati; nella seconda vediamo la validazione del modulo riportando dei risultati ottenuti da simulazioni su reti generate utilizzando il generatore di topologie casuali BRITE [1].

4.1 Modifiche apportate

Abbiamo costruito il modulo per LSP P2MP integrando un agent RSVP-TE e un classifier MPLS, col primo che si occupa principalmente della prenotazione delle risorse e della distribuzione delle label e il secondo che si occupa invece delle operazioni di label switching per il forwarding dei pacchetti, replicando gli stessi nel caso in cui il nodo in questione sia punto di ramificazione per quella sessione P2MP. Inoltre sono state aggiunte due funzioni all'agente che implementa il protocollo di routing LS in modo che quest'ultimo calcoli i percorsi punto-multipunto seguendo due euristiche prestabilite.

4.1.1 I file di RSVP-TE/ns

Per quanto riguarda i file del modulo RSVP-TE/ns, tutti implementati in C++ [3], abbiamo principalmente aggiunto i nuovi oggetti previsti dall'estensione per LSP P2MP, inserito dei nuovi campi nella **psb** e implementato tutte le funzioni per la segnalazione degli LSP introducendo una nuova struttura pre gestire le sessioni punto-multipunto denominata **p2mp_session**. In particolare, abbiamo aggiunto gli oggetti di cui di seguito riportiamo l'interfaccia.

```
/*=========================*
 *    P2MP_FILTER_SPEC       *
 *=========================*/

class P2MP_FILTER_SPEC : public RSVPobject {
public:
  P2MP_FILTER_SPEC(nsaddr_t src, int t);
  P2MP_FILTER_SPEC(nsaddr_t src, int t,
                   int sg_or, int sg);
  P2MP_FILTER_SPEC(unsigned char *cont);
  virtual ~P2MP_FILTER_SPEC() {};
  inline nsaddr_t get_src() { return con->src; };
  inline int get_tid() { return con->tid; };
  inline int get_sg_or_id() { return con->sg_or_id; };
  inline int get_sg_id() { return con->sg_id; };
  void dump_object();
private:
  struct construct {
    nsaddr_t src;     /* sender addr */
```

```
    int      tid;     /* tunnel id */
    int      sg_or_id;/* sub-group originator id */
    int      sg_id;   /* sub-group id */
  };
  struct construct *con;
};

/*==========================*
 *    P2MP_SENDER_TEMPLATE   *
 *==========================*/

class P2MP_SENDER_TEMPLATE : public RSVPobject {
public:
  P2MP_SENDER_TEMPLATE(nsaddr_t src, int t,
                       int sg_or, int sg);
  P2MP_SENDER_TEMPLATE(nsaddr_t src, int t);
  P2MP_SENDER_TEMPLATE(unsigned char *cont);
  virtual ~P2MP_SENDER_TEMPLATE() {};
  inline nsaddr_t get_src() { return con->src; };
  inline int get_tid() { return con->tid; };
  inline int get_sg_or_id() { return con->sg_or_id; };
  inline int get_sg_id() { return con->sg_id; };
  void dump_object();
private:
  struct construct {
    nsaddr_t src;     /* sender addr */
    int      tid;     /* tunnel id */
    int      sg_or_id;/* sub-group originator id */
    int      sg_id;   /* sub-group id */
  };
  struct construct *con;
};

/*====================*
 *    EXPLICIT_ROUTE   *
 *====================*/

class EXPLICIT_ROUTE : public RSVPobject {
public:
  EXPLICIT_ROUTE(elem* e_path);
  EXPLICIT_ROUTE(unsigned char *cont);
  virtual ~EXPLICIT_ROUTE() {};
  inline elem* get_ero() { return con->p_head; };
  inline int get_nhop() { return con->nhop; };
  void dump_object();
private:
  struct construct {
    elem *p_head;/* The head of the linked path's list */
    int nhop;    /* The total number of hops */
  };
```

```
  struct construct *con;
};

/*======================*
 *    S2L_SUB_LSP       *
 *======================*/

class S2L_SUB_LSP : public RSVPobject {
public:
  S2L_SUB_LSP(nsaddr_t e);
  S2L_SUB_LSP(unsigned char *cont);
  virtual ~S2L_SUB_LSP() {};
  inline nsaddr_t get_egr() { return con->egr; }
  void dump_object();
private:
  struct construct {
    nsaddr_t egr;/* egress address */
  };
  struct construct *con;
};

/*=========================*
 *    SEC_EXPLICIT_ROUTE   *
 *=========================*/

class SEC_EXPLICIT_ROUTE : public RSVPobject {
public:
  SEC_EXPLICIT_ROUTE(elem* se_path);
  SEC_EXPLICIT_ROUTE(unsigned char *cont);
  virtual ~SEC_EXPLICIT_ROUTE() {};
  inline elem* get_sero(){ return con->p_head; };
  inline int get_nhop() { return con->nhop; };
  void dump_object();
private:
  struct construct {
    elem *p_head;/* The head of the linked path's list */
    int nhop;    /* The total number of hops */
  };
  struct construct *con;
};
```

Come vediamo dal codice riportato abbiamo implementato costruttori, distruttori, funzioni di stampa e funzioni per l'estrazione dei valori dai vari campi dell'oggetto.

Ci siamo poi occupati dell'aggiunta di nuove informazioni nella struttura **psb** a cui è stato aggiunto un puntatore all'oggetto **P2MP_SENDER_TEMPLATE** e un campo **int b_point** per poter stabilire se il nodo sia o meno un branching point.

Per la gestione delle sessioni punto-multipunto abbiamo scelto di definire una nuova struttura, **p2mp_session**, in modo che le nuove funzioni interferissero il meno possibile da quelle precedenti. La nuova struttura è riportata di seguito.

```
struct p2mp_session {
  p2mp_session() : p2mp_s(0), path_tv(0), resv_tv(0),
                   psb_list(0), rsb_list(0),
                   tcsb_list(0), next(0), t_next(0),
                   path_ref(-1), resv_ref(-1), ref_flag(0),
                   local(0), confirm(0) {};

  ~p2mp_session() {
    delete p2mp_s;
    delete path_tv;
    delete resv_tv;
  }
  int get_p2mp_id() { return p2mp_id; }

  P2MP_SESSION *p2mp_s;      /* P2MP session obj. ptr */
  TIME_VALUES  *path_tv;     /* Time value for path */
  TIME_VALUES  *resv_tv;     /* Time value for resv */
  psb          *psb_list;    /* The list of PSB */
  rsb          *rsb_list;    /* The list of RSB */
  tcsb         *tcsb_list;   /* The list of TCSB */
  p2mp_session *next;        /* The next session in the
                                session list */
  p2mp_session *t_next;      /* The next session in the
                                timer list */
  double   path_ref;         /* The time at which the path
                                state is refreshed */
  double   resv_ref;         /* The time at which the resv
                                state is refreshed */
  char     ref_flag;         /* Has this session been scheduled
                                for a refresh? */
  char     local;            /* Did this session originate from
                                a local API call? */
  char     confirm;          /* Will a RESV_CONFIRM object be
                                sent with the next Resv? */
  elem     *s2l_tbl[P2MP_ID];/* A "map" <p2mp_id, egr_list> */
  nsaddr_t ingress;          /* The ingress node */
  int      status;
  char     handle[8];
  int      p2mp_id;          /* P2MP session ID */
};
```

Conseguentemente a queste modifiche, abbiamo dovuto adattare tutte le funzioni riguardanti la gestione delle sessioni in

modo da gestire la nuova struttura `p2mp_session` e modificare alcune funzioni che manipolavano la `psb` così da tener conto della nuova configurazione. Abbiamo poi modificato le funzioni per il processing e l'invio dei messaggi di segnalazione, funzioni che devono tener conto di tutti i nuovi oggetti aggiunti e soprattutto devono saper gestire la segnalazione di percorsi punto-multipunto. Abbiamo dunque aggiunto tre nuovi messaggi RSVP che sono:

```
#define PMP_CE 22
#define PMP_PATH 23
#define PMP_RESV 24
```

Per fare in modo che il simulatore tratti correttamente questi messaggi, nell'enumerazione `packet_t` all'interno del file `common/packet.h`, abbiamo aggiunto:

```
enum packet_t {
  PT_TCP,
  /* ... */
  PT_PMP_CE,
  PT_PMP_PATH,
  PT_PMP_RESV,

  PT_NTYPE // This MUST be the LAST one
};
```

e, nello stesso file:

```
class p_info {
public:
  p_info() {
    name_[PT_TCP]= "tcp";
    /* ... */
    name_[PT_PMP_CE] = "PMPce";
    name_[PT_PMP_PATH] = "PMPpath";
    name_[PT_PMP_RESV] = "PMPresv";

    name_[PT_NTYPE] = "undefined";
}
```

Abbiamo quindi aggiunto le funzioni per il processing di cui riportiamo l'intestazione.

```
void send_p2mp_ce_message(nsaddr_t src, nsaddr_t ingr,
                          int egr_num, nsaddr_t egr_vect[],
                          nsaddr_t dst_vect[],
                          elem* t_paths[], int fid,
                          double rate, int buck, int ttl);
```

```
void send_p2mp_path_message(p2mp_session *p2mp_s, psb *p);
int  send_p2mp_resv_message(p2mp_session *p2mp_s, rsb *r,
                            char f);

void process_p2mp_ce_message(RSVPmessage *msg);
void process_p2mp_path_message(RSVPmessage *msg, int iface,
                               RSVPChecker *check);
void process_p2mp_resv_message(RSVPmessage *msg, nsaddr_t p_hop);
```

La scelta su quale delle funzioni eseguire viene effettuata tramite il codice seguente.

```
void RSVPAgent::give(Packet *p, RSVPChecker *ret) {

  hdr_rsvp* rsvp_hdr = hdr_rsvp::access(p);
  hdr_cmn *cmn_hdr = hdr_cmn::access(p);
  hdr_ip* ip_hdr = hdr_ip::access(p);

  RSVPmessage *msg = new RSVPmessage(p->accessdata());
  switch (msg->get_type()) {
    case PATH:
      process_path_message(msg, cmn_hdr->iface_, ret);
      break;
    case PATHTEAR:
      process_path_tear_message(msg, cmn_hdr->iface_, ret);
      break;
    case RESV:
      process_resv_message(msg, rsvp_hdr->fromhop);
      break;

    /* ... */

    case PMP_CE: {
      process_p2mp_ce_message(msg);
      break;
    }
    case PMP_PATH: {
      process_p2mp_path_message(msg, cmn_hdr->iface_, ret);
      break;
    }
    case PMP_RESV: {
      process_p2mp_resv_message(msg, rsvp_hdr->fromhop);
      break;
    }
  }
  delete msg;
  Packet::free(p);
}
```

4.1.2 I file di MNS

Per fare in modo che l'agent MPLS trattasse correttamente i pacchetti appartenenti alle sessioni punto-multipunto, abbiamo aggiunto, all'interno del file **classifier-addr-mpls.h**, la struttura seguente:

```
struct PMPEntry {
  PMPEntry() : src_(-1), pmp_id_(-1), fid_(-1),
               tid_(-1), is_egr_(0) {
    for(int i = 0; i < MPLS_MaxPMPptr; ++i) {
      mLIBptr_[i] = -1;
    }
  }
  int src_;
  int pmp_id_;
  int fid_;
  int tid_;
  int is_egr_;
  int mLIBptr_[MPLS_MaxPMPptr];
};

struct PMP {
  PMPEntry Entry_[MPLS_MaxPMPEntryNB];
  int NB_;
};
```

variabile membro della classe **MPLSAddressClassifier**, così da poter identificare i pacchetti appartenenti a una certa sessione P2MP e duplicare il traffico nel caso in cui il nodo sia branching point. All'interno della stessa classa abbiamo aggiunto le funzioni che la gestiscono di cui riportiamo l'intestazione:

```
  int PMPnewEntry();
  int PMPgetEntry(int src, int fid);
  int PMPsearch(int src, int fid);
  int PMPinsert(int src, int pmp_id, int fid, int tid,
                int lib_ptr, int is_egr, int is_ingr);
  int PMPgetNbrNum(int entrynb);
  void PMPdump(const char *id);
```

Tuttavia le principali modifiche al modulo MNS riguardano il file **ns-mpls-node.tcl** all'interno di **tcl/mns_v2.0**. Le due principali funzioni introdotte sono:

```
RtModule/MPLS instproc create-p2mp-SPH { src ingr e_l d_l fid
                                         rate {buck 5000}
                                         {ttl 32} }
```

e:

```
RtModule/MPLS instproc create-p2mp-SPHm { src ingr ctr_node e_l
                                          d_l fid rate
                                          {buck 5000} {ttl 32} }
```

Come possiamo notare le due funzioni richiedono come parametri la sorgente, la lista degli egress e quella delle destinazioni, il flow-id e il rate. Consentono inoltre di specificare i valori di bucket e TTL che, di default, assumono rispettivamente i valori 5000 e 32. La seconda procedura richiede di introdurre anche un ulteriore parametro, il riferimento ad un nodo della rete, denominato `ctr_node`.

Inserendo, da script, uno dei due comandi facciamo in modo che un nodo, scelto all'interno della rete, calcoli l'abero multicast e invii al nodo ingress un nuovo messeggio RSVP, che abbiamo denominato P2MP_CE, che contiene l'albero rappresentato nel modo previsto dal protocollo RSVP.

Con il comando `create-p2mp-SPH` l'albero è calcolato seguendo l'euristica *Shortest Path* [7], ovvero unendo i percorsi a costo minimo che connettono la sorgente agli egress. Inoltre, poiché vogliamo garantire alla nostra applicazione un certo requisito di banda, specificato nella variabile `rate`, il calcolo dell'albero viene effettuato dopo che dalla rete siano stati eliminati i link che non soddisfano al requisito di banda[1].

Con il `create-p2mp-SPHm` l'LSP punto-multipunto viene calcolato utilizzando una versione modificata della Shortest Path Heuristic in cui si fa in modo che tutti gli LSP condividano il percorso dall'ingress fino ad un certo nodo `ctr_node`, impostato da script. Ovvero l'albero multicast è formato unendo al percorso ottimo dall'ingress al `ctr_node` i percorsi ottimi dal `ctr_node` verso tutti gli egress.

Una volta che l'ingress ha processato il messaggio proveniente dal nodo CE (Computation Element) inizia il normale processo di segnalazione per P2MP LSP. Come previsto dalla RFC abbiamo fatto in modo che l'intero punto-multipunto venisse segnalato con un singolo messaggio di Path.

Per concludere riportiamo uno script di esempio in cui vediamo l'applicazione dei due nuovi comandi aggiunti. La

[1] Possiamo pensare di attribuire a questi link costo "infinito".

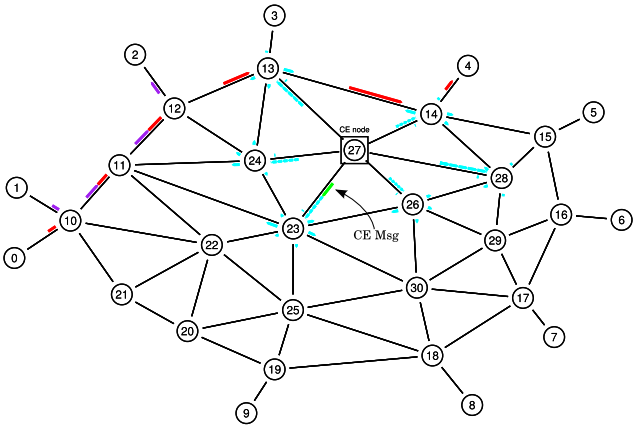


Fig. 4.1: Invio del messaggio P2MP_CE.

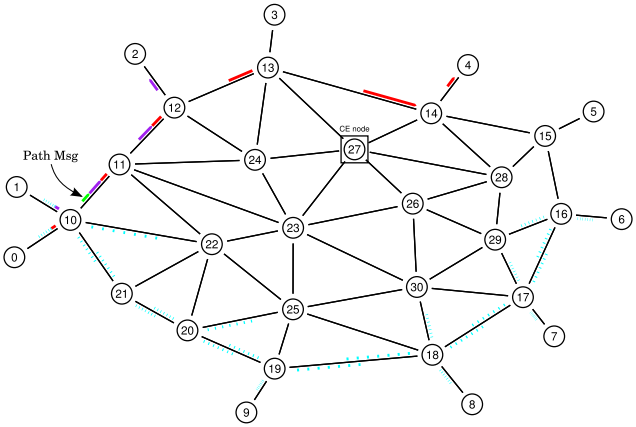


Fig. 4.2: Invio del messaggio di Path.

Figura 4.1 riporta l'invio del messaggio P2MP_CE da parte del nodo che effettua il calcolo dell'LSP punto-multipunto. In Figura 4.2 vediamo l'invio del messaggio di Path da parte dell'ingress.

```
set ns [new Simulator]

$ns rtproto LS

$ns color 0 cyan
#blue
$ns color 1 red
$ns color 2 purple
#$ns color 3 yellow pink
$ns color 4 magenta
$ns color 46 green

set nf [open filenam.nam w]
set tf [open filetrace.tr w]
$ns namtrace-all $nf
$ns trace-all $tf

set n0  [$ns node]
set n1  [$ns node]
set n2  [$ns node]
set n3  [$ns node]
set n4  [$ns node]
set n5  [$ns node]
set n6  [$ns node]
set n7  [$ns node]
set n8  [$ns node]
set n9  [$ns node]
set n10 [$ns mpls-node]
set n11 [$ns mpls-node]
set n12 [$ns mpls-node]
set n13 [$ns mpls-node]
set n14 [$ns mpls-node]
set n15 [$ns mpls-node]
set n16 [$ns mpls-node]
set n17 [$ns mpls-node]
set n18 [$ns mpls-node]
set n19 [$ns mpls-node]
set n20 [$ns mpls-node]
set n21 [$ns mpls-node]
set n22 [$ns mpls-node]
set n23 [$ns mpls-node]
set n24 [$ns mpls-node]
set n25 [$ns mpls-node]
set n26 [$ns mpls-node]
```

```
set n27 [$ns mpls-node]
set n28 [$ns mpls-node]
set n29 [$ns mpls-node]
set n30 [$ns mpls-node]

#$n26 shape "box"

$ns duplex-rsvp-link $n0  $n10 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n1  $n10 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n12 $n2  1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n13 $n3  1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n14 $n4  1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n15 $n5  1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n16 $n6  1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n17 $n7  1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n18 $n8  1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n19 $n9  1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n10 $n11 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n10 $n21 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n10 $n22 1Mb 10ms
0.99 1000 10000 Param Null
#$ns duplex-rsvp-link $n10 $n26 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n11 $n12 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n11 $n22 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n11 $n23 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n11 $n24 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n12 $n13 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n12 $n24 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n13 $n14 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n13 $n24 1Mb 10ms
```

```
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n13 $n27 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n14 $n15 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n14 $n27 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n14 $n28 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n15 $n16 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n15 $n28 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n16 $n17 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n16 $n29 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n17 $n29 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n17 $n30 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n17 $n18 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n18 $n19 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n18 $n30 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n19 $n20 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n20 $n22 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n21 $n22 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n21 $n20 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n22 $n23 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n22 $n25 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n23 $n24 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n23 $n25 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n23 $n26 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n23 $n27 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n23 $n30 1Mb 10ms
0.99 1000 10000 Param Null
#$ns duplex-rsvp-link $n24 $n26 1Mb 10ms
```

```
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n24 $n27 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n25 $n18 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n25 $n19 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n25 $n20 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n25 $n30 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n26 $n27 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n26 $n28 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n26 $n29 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n26 $n30 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n27 $n28 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n28 $n29 1Mb 10ms
0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n29 $n30 1Mb 10ms
0.99 1000 10000 Param Null

# Enable upcalls on all nodes
Agent/RSVP set noisy_ 255

set rsvp0 [$n0 add-rsvp-agent]
set rsvp1 [$n1 add-rsvp-agent]
set rsvp2 [$n2 add-rsvp-agent]
set rsvp3 [$n3 add-rsvp-agent]
set rsvp4 [$n4 add-rsvp-agent]
set rsvp5 [$n5 add-rsvp-agent]
set rsvp6 [$n6 add-rsvp-agent]
set rsvp7 [$n7 add-rsvp-agent]
set rsvp8 [$n8 add-rsvp-agent]
set rsvp9 [$n9 add-rsvp-agent]
set rsvp10 [$n10 add-rsvp-agent]
set rsvp11 [$n11 add-rsvp-agent]
set rsvp12 [$n12 add-rsvp-agent]
set rsvp13 [$n13 add-rsvp-agent]
set rsvp14 [$n14 add-rsvp-agent]
set rsvp15 [$n15 add-rsvp-agent]
set rsvp16 [$n16 add-rsvp-agent]
set rsvp17 [$n17 add-rsvp-agent]
set rsvp18 [$n18 add-rsvp-agent]
set rsvp19 [$n19 add-rsvp-agent]
set rsvp20 [$n20 add-rsvp-agent]
```

```
set rsvp21 [$n21 add-rsvp-agent]
set rsvp22 [$n22 add-rsvp-agent]
set rsvp23 [$n23 add-rsvp-agent]
set rsvp24 [$n24 add-rsvp-agent]
set rsvp25 [$n25 add-rsvp-agent]
set rsvp26 [$n26 add-rsvp-agent]
set rsvp27 [$n27 add-rsvp-agent]
set rsvp28 [$n28 add-rsvp-agent]
set rsvp29 [$n29 add-rsvp-agent]
set rsvp30 [$n30 add-rsvp-agent]

set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
$udp0 set packetSize_ 500
$udp0 set fid_ 1
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set rate_ 300Kb
$cbr0 attach-agent $udp0

set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1
$udp1 set packetSize_ 500
$udp1 set fid_ 2
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set rate_ 300Kb
$cbr1 attach-agent $udp1


set sink2 [new Agent/LossMonitor]
$ns attach-agent $n2 $sink2
set sink3 [new Agent/LossMonitor]
$ns attach-agent $n3 $sink3
set sink4 [new Agent/LossMonitor]
$ns attach-agent $n4 $sink4
set sink5 [new Agent/LossMonitor]
$ns attach-agent $n5 $sink5
set sink6 [new Agent/LossMonitor]
$ns attach-agent $n6 $sink6
set sink7 [new Agent/LossMonitor]
$ns attach-agent $n7 $sink7
set sink8 [new Agent/LossMonitor]
$ns attach-agent $n8 $sink8
set sink9 [new Agent/LossMonitor]
$ns attach-agent $n9 $sink9

$ns connect $udp0 $sink4
$ns connect $udp1 $sink2
```

```
for {set i 10} {$i < 31} {incr i} {
  set a n$i
  set m [eval $$a get-module "MPLS"]
  eval set LSR$i $m
}


proc finish {} {
  global ns nf tf
  close $nf
  $ns flush-trace
  close $tf
  exec nam filenam.nam &
  puts "END"
    exit 0
}


$ns at 0.2  "$cbr0 start"
$ns at 2.0  "$cbr0 stop"
$ns at 0.2  "$cbr1 start"
$ns at 2.0  "$cbr1 stop"

$ns at 0.0 "$n27 label \"CE node\""

$ns at 1.0 "$LSR27 create-p2mp-SPH $n0 $n10
n13_n14_n15_n17_n18_n19 n3_n4_n5_n7_n8_n9 1 +700000"

#$ns at 1.0 "$LSR27 create-p2mp-SPHm $n0 $n10 $n27
n13_n14_n15_n17_n18_n19 n3_n4_n5_n7_n8_n9 1 +700000"

$ns at 2.0 "finish"

$ns run
```

La Figura 4.3 riporta invece l'invio del messaggio di Resv e, infine, nella Figura 4.4 mostriamo l'albero ottenuto con l'euristica SPH e in Figura 4.5 quello ottenuto con l'euristica SPHm.

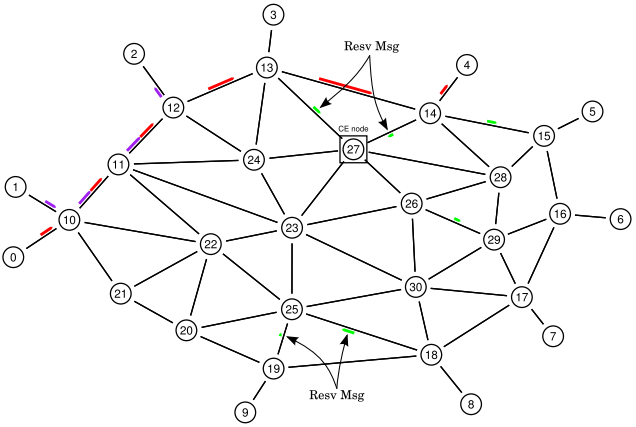


Fig. 4.3: Invio del messaggio di Resv.

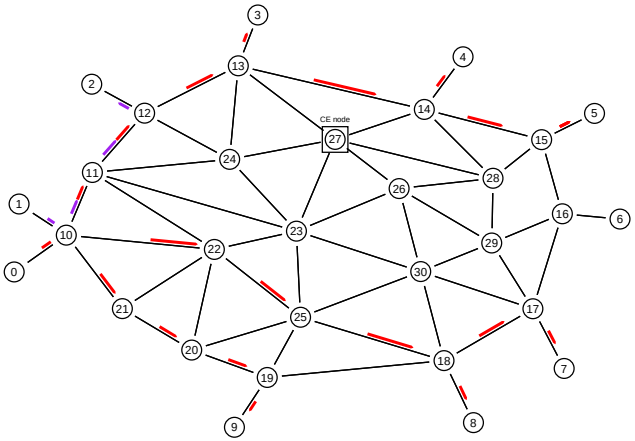


Fig. 4.4: Albero ottenuto con l'euristica SPH.

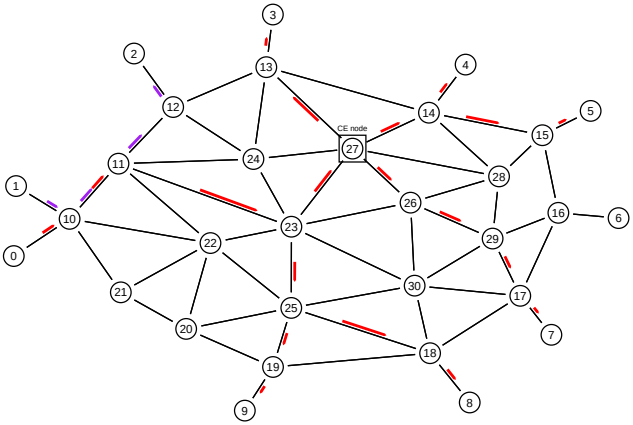


Fig. 4.5: Albero ottenuto con l'euristica SPHm.

4.2 Validazione del modulo

Per validare il nuovo modulo abbiamo utilizzato il generatore di topologie casuali BRITE. Per generare la rete abbiamo usato il modello di Waxman [2], un modello geografico in cui i nodi sono uniformemente distribuiti nel piano e tale che la probabilità che due nodi siano connessi è data da:

$$P(u,v) = \alpha e^{-d(u,v)/(\beta L)}$$

con $\alpha > 0$, $\beta \leqslant 1$ e $d(u,v)$ è la distanza euclidea tra i nodi. Per testare le due euristiche abbiamo generato 10 topologie casuali ciascuna di 30 nodi da cui ne abbiamo ottenute 50 mpdificando le posizioni relative di egress e sorgente all'interno della rete. I risultati ottenuti sono riportati in Figura 4.6. Possiamo osservare come l'euristica SPH ottenga mediamente risultati migliori, tuttavia come mostra la Figuira 4.2 la seconda euristica presenta un intervallo di confidenza maggiore ad indicare una maggiori fluttuazioni dei risultati ottenuti. Questo è principalmente dovuto al fatto che il costo della soluzione calcolata dalla seconda euristica è direttamente legato alla scelta del nodo centrale da cui calcoliamo i percorsi verso tutti gli egress. Una scelta sbagliata in questo senso può portare ad ottenere alberi con costi mol-

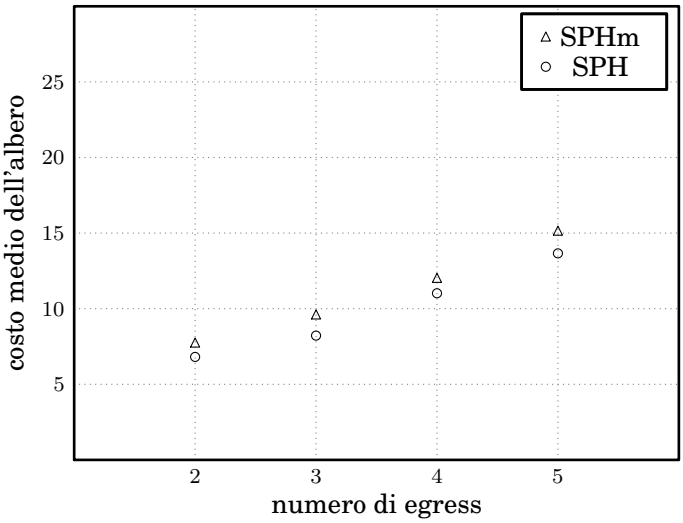


Fig. 4.6: Risultati ottenuti.

Shortest Path Heuristic				
Num. Egr.	2	3	4	5
v.m.	6.81	8.22	11.02	13.66
dev. std.	1.67	1.5	1.4	1.78
I.C. 99%	1.21	1.1	1	1.3

Shortest Path Heuristic modified				
Num. Egr.	2	3	4	5
v.m.	7.76	9.62	12.04	15.16
dev. std.	1.87	2	1.76	2.24
I.C. 99%	1.37	1.52	1.28	1.63

Fig. 4.7: Intervalli di confidenza delle due euristiche.

to elevati. Un possibile criterio di scelta potrebbere essere quello di scegliere il nodo centrale minimizzando la somma costo dei percorsi verso tutti gli egress, in formule:

$$v = \underset{\forall v \in V \setminus \{G\}}{\operatorname{argmin}} \sum_{g \in G} P(v, g)$$

Tuttavia una tale strada potrebbe essere computazionalmente troppo complessa da seguire[2], un modo per mitigare il fenomeno potrebbe essere quello di limitare la scelta solo tra un sottoinsieme limitato di nodi.

[2]La complessità di questa soluzione è poporzionale a $\mathcal{O}(V^3)$.

5 Conclusioni

In questo lavoro di tesi ci siamo occupati dell'estensione a RSVP-TE per la gestione di LSP P2MP. Dopo una prima fase in cui abbbiamo curato gli aspetti puramente teorici, la parte di lavoro che ha richiesto decisamente maggior tempo è stata la realizzazione del modulo che fornisse al simulatore di reti NS2 la possibilità di instaurare LSP punto-multipunto. Per realizzare il nostro obbiettivo è stato necessario utilizzare una versione un pò "datata"[1] di NS2, ovvero la release 2.26 del 2003[2]; infatti solo per questa versione avevamo a disposizione il modulo che implementa l'MPLS, MNS e la patch per RSVP-TE, RSVP-TE/ns. Queste due precedenti estensioni sono state la base del nostro lavoro, infatti analizzando le funzionalità già implementate è stato possibile capire le modifiche da apportare per realizzare il nostro scopo.

Il nostro lavoro può essere pensato diviso in tre fasi. Nella prima fase quello che abbiamo fatto è stato implementare i nuovi oggetti previsti dalla RFC4875 e fare in modo che un certo nodo, inserito da script, fosse in grado di calcolare prcorsi punto-multipunto. Abbiamo quindi aggiunto un nuovo tipo di messaggio RSVP, che abiamo denomnato P2MP_CE, per trasportare questa informazione e il codice necessario al processing di quest'ultimo una volta raggiunto l'ingress. Nella seconda fase ci siamo occupati dell'invio del messaggio di Path e del suo processing, che nella versione punto-multipunto deve tener conto dei nodi branching

[1] Attualmente, l'ultima release è la 2.33 di marzo 2008.

[2] Questo ha causato diversi problemi dovuti al fatto che necessita di versioni oramai superate di librerie e, ancor di piú, compilatore.

point così da consegnare correttamente i pacchetti. L'ultima tappa ha riguardato l'invio del messaggio di Resv e il binding del flusso di cui è stata fatta la segnalazione sull'LSP punto-multipunto.

Siamo quindi passati alla validazione del modulo che abbiamo svolto utilizzando il generatore di topologie casuali BRITE. Quello che abbiamo fatto è stato generare un certo numero di topologie casuali e, su queste, confrontare i risultati ottenuti, in termini di costo della soluzione, dalle due euristiche implementate. Quello che abbiamo osservato è stato che la prima euristica fornisce in media risultati migliori, anche se la seconda presenta un intervallo di cofidenza maggiore e dunque soluzioni piú fluttuanti attorno al valor medio.

Questa tesi non rappreseta certamente un lavoro finito, infatti non sono stati presi in considerazione tutti gli aspetti relativi a LSP P2MP. Quello che resta da esaminare rigurda la possibilità di gestire i gruppi in modo dinamico, le strategie da impiegare in caso di rottura di link e l'eventuale implementazione di altre euristiche per il calcolo di percorsi punto-multipnto.

A File modificati

In questa appendice riportiamo la lista completa dei file di NS2 modificati per la realizzazione del modulo[1].

In dettaglio, all'interno della cartella `common` abbiamo modificato il file:

- `packet.h`.

Nella cartella `linkstate` abbiamo modificato i file:

- `rtProtoLs.h`;
- `rtProtoLS.cc`.

All'interno della cartella `mns_v2.0` abbiamo modificato i file:

- `classifier-addr-mpls.h`,
- `classifier-addr-mpls.cc`,
- `rsvp.h`,
- `rsvp.cc`,
- `rsvp-object.h`,
- `rsvp-object.cc`,
- `rsvp-message.h`;
- `rsvp-message.cc`.

[1] Per ottenere il codice del nostro modulo inviare una e-mail all'indirizzo `bartoli.lorenzo@gmail.com`, specificando l'uso.

Nella cartella `tcl/lib` abbiamo modificato il file:

- `ns-lib.tcl`.

All'interno della cartella `tcl/rtglib` abbiamo modificato il file:

- `ns-rtProtoLS.tcl`.

Infine, della cartella `tcl/mns_v2.0` abbiamo modificato il file:

- `ns-mpls-node.tcl`.

Bibliografia

[1] Alberto Medina e Anukool Lakhina e Ibrahim Matta e John Byers. BRITE: Universal Topology Generation from a User's Perspective, Aprile 2001. Progetto non piú supportato.

[2] Bernard M. Waxman. Routing of multipoint connections. pages 347–352, 1991.

[3] Bjarne Stroustrup. *C++. Linguaggio, libreria standard, principi di programmazione. Terza Edizione*. Addison-Wesley, San Francisco, CA, USA, 2000.

[4] D. Adami e C. Callegari e S. Giordano e F. Mustacchio e M. Pagano e F.Vitucci. Signalling Protocols in DiffServ-aware MPLS Networks: Design and Implementation of RSVP-TE Network Simulator. GLOBECOM, Novembre 2005.

[5] D. Awduche e L. Berger e D. Gan e T. Li e V. Srinivasan e G. Swallow. RFC 3209: RSVP-TE: Extension to RSVP for LSP Tunnels. RFC 3209, Dicembre 2001. Aggiornata dalle RFC 3936, 4420.

[6] Gaeil Ahn e Woojik Chun. Design and implementation of MPLS Network Simulator (MNS), 2000.

[7] Hiromitsu Takahashi e Akira Matsuyama. An Approximate Solution for the Steiner Problem in Graphs. *Mathematica Japonica*, 24(6):573–577, 1980.

[8] Kevin Fall e Kannam Varadhan. The *ns* Manual (formely *ns* Notes and Documentation). `http://www.isi.edu/nsnam/ns/ns-documentation.html`.

[9] M.Greis. RSVP/ns: An Implementation of RSVP for the Network Simulator ns-2, 2002.

[10] R. Aggarwal e D. Papadimitriou e S. Yasukawa. RFC 4875: Extension to RSVP-TE for P2MP TE Label Switched Paths LSPs. RFC 4875, Maggio 2007.

[11] R. Braden, Ed. e L. Zhang e S. Berson e S. Herzog e S. Jamin. RFC 2205: Resource ReSerVation Protocol (RSVP) - version 1 functional specification. RFC 2205, Settembre 1997.

[12] Rosario G. Garroppo. Appunti di Progetto e Simulazione di Reti di Telecomunicazioni: Network Simulator vers. 2 e sue Applicazioni. SEU, 2003. Pisa, Italia.