POLITECNICO DI MILANO

V Facoltà di Ingegneria
Corso di Laurea Specialistica in Ingegneria delle Telecomunicazioni
Dipartimento di Elettronica e Informazione

Sviluppo ed Implementazione di Soluzioni di Risparmio Energetico in Reti OSPF: Approccio Centralizzato

Relatore: Prof. Antonio Capone

Tesi di Laurea di:
Emanuele Colombo Matr. 740094
Riccardo Locatelli Matr. 735020

Anno Accademico 2009 – 2010

Indice

Elenco delle figure

Elenco delle tabelle

Sommario

Italiano

Negli ultimi anni, stiamo assistendo ad una crescita dell'interesse nel campo del risparmio energetico. Un settore particolarmente interessato da questo fenomeno, e in forte sviluppo, è quello dell'ICT. Questo interesse deriva dalla consapevolezza che nei prossimi anni l'utilizzo di Internet aumenterà, facendo divenire il settore ICT uno dei maggiori consumatori di risorse energetiche.

L'utilizzo delle reti alterna periodi di picco a periodi di basso carico, attestati soprattutto nelle ore notturne. Il *Green Networking* si pone l'obiettivo di sfruttare le situazioni di basso carico per ridurre il consumo energetico.

In questa tesi, presentiamo un metodo che permette di porre in *sleeping* i dispositivi di rete poco utilizzati e di gestirne il risveglio. Nell'approccio mostrato è presente un'entità centrale, alla quale sono demandati i compiti decisionali. Lo scenario considerato è una rete di tipo OSPF nella quale abbiamo sfruttato le potenzialità del protocollo SNMP per ottenere le caratteristiche desiderate.

I risultati mostrano come sia possibile ottenere un risparmio energetico rilevante, mantenendo inalterata la qualità del servizio.

English

In the last few years, we are assisting to a growth of interest in the field of energy saving. A particularly involved sector in this process is the ICT sector. This interest comes from the awareness that in the coming years the use of Internet will increase, making the ICT sector one of the most important consumer of energy resources.

Network utilization alternates peak load periods to low load ones, which occur especially in night-time. The purpose of the *Green Networking* is to take advantage of low load situations to reduce energy consumption.

In this work, we introduce a method which allows to set in *sleeping* mode the underutilized network devices and manage their awakening. Said approach is based on a central entity which is responsable of decisional choices. We consider an OSPF scenario in which we have exploited the SNMP protocol capabilities to obtain desired features.

The results show that is possible to obtain a considerable energy saving, maintaining the same quality of service.

Introduzione

Negli ultimi anni, la forte crescita del settore ICT, e di Internet in particolare, ha portato ad un considerevole aumento dei consumi energetici globali. La richiesta di risorse delle moderne applicazioni di networking continua a crescere, facendo di Internet uno dei principali consumatori di energia mondiali.

Diversamente da quanto accade per le reti mobili, che sono già studiate per il risparmio energetico, le reti cablate sono state concepite senza preoccuparsi di tale problema, con consumi che si mantengono costanti anche in situazioni di basso carico della rete.

I primi a sottolineare l'importanza della riduzione del consumo energetico delle reti cablate sono stati Gupta e Singh [29], nel 2003. Essi hanno individuato come strategia più appropriata l'implementazione di stati di risparmio energetico all'interno dei dispositivi di rete. Hanno analizzato le problematiche derivanti da una simile modifica nelle apparecchiature e nei principali protocolli di instradamento utilizzati in Internet, come OSPF. La presenza di uno stato a basso consumo energetico diventa vantaggiosa solo se i dispositivi di rete riescono a permanere in questo stato abbastanza a lungo da compensare il picco di consumo necessario per il loro risveglio. Da questo punto di vista, l'attuale implementazione di OSPF creerebbe problemi: infatti, ogni nodo della rete si annuncia agli altri attraverso messaggi periodici detti *Hello*. La durata di un intervallo di *sleeping* non dovrebbe essere maggiore del tempo che intercorre tra due *Hello*. Appare evidente che ciò limiterebbe molto il risparmio energetico.

Porre un dispositivo in *sleeping* comporterebbe inoltre un intenso scambio di ag-

giornamenti (pacchetti LSA) sulla situazione della rete tra i nodi. Questo causerebbe continue variazioni delle tabelle di instradamento come nel caso di un guasto (non necessario nel caso di *sleeping*). Per questo motivo nasce la necessità di separare logicamente le due situazioni.

Da qui è nato questo lavoro, con l'obiettivo di implementare stati di basso consumo energetico all'interno di reti OSPF. Abbiamo osservato fin dal principio come la modifica di un protocollo affermato come OSPF non fosse un obiettivo banale. Per questo motivo, abbiamo deciso di affrontare il problema ad un livello protocollare superiore individuando la modifica del MIB di SNMP come possibile alternativa (v. Capitolo 3). Ci siamo quindi dedicati all'aggiunta dello *sleeping* tra i possibili stati amministrativi assumibili dalle interfacce dei dispositivi. Nonostante questa scelta, è comunque necessario prevedere delle piccole modifiche al protocollo OSPF. Abbiamo infatti individuato la necessità di discriminare, anche a livello OSPF, lo stato di guasto da quello a basso consumo e di diffondere in rete questa informazione. Per ovviare a questi problemi abbiamo proposto delle modifiche al protocollo di Hello utilizzato da OSPF e l'utilizzo di particolari pacchetti, gli Opaque-LSA [18], per l'inoltro delle nuove informazioni.

Per implementare e testare le modifiche descritte abbiamo lavorato in ambiente Linux, data la vasta disponibilità di software open source modificabili a seconda delle proprie esigenze. Ci siamo serviti di Netkit, un emulatore di reti, che abbiamo dovuto integrare con la nostra nuova versione di Net-SNMP, software che ci ha permesso di implementare le modifiche pensate per il MIB del protocollo SNMP.

Abbiamo individuato tre differenti scenari applicativi:

- scenario centralizzato statico: solo l'entità centrale prende decisioni riguardo lo *sleeping*;

- scenario centralizzato dinamico: l'entità centrale prende decisioni riguardo lo *sleeping* ma periodicamente gli altri dispositivi possono reagire chiedendo ad essa l'eventuale necessità di risveglio;

- scenario distribuito: tutti i dispositivi prendono decisioni sul proprio stato, conoscendo la situazione della rete.

In questa tesi ci siamo occupati principalmente dell'implementazione dei primi due, ottenuta sfruttando le modifiche apportate a Net-SNMP all'interno di Netkit e la creazione di appositi script di gestione da eseguire nelle macchine virtuali. Attraverso i risultati ottenuti tramite l'emulazione in Netkit, abbiamo valutato che la soluzione proposta permette un risparmio energetico rilevante in reti OSPF.

Il lavoro di tesi è stato svolto presso il Politecnico di Milano. La struttura del documento è illustrata in seguito.

Il Capitolo 1 descrive la situazione attuale delle reti Internet dal punto di vista del consumo energetico. Vengono inoltre spiegate diverse tecniche per limitare l'utilizzo di energia all'interno delle reti stesse.

Nel Capitolo 2 viene descritto brevemente il protocollo OSPF e le sue principali caratteristiche.

Il Capitolo 3 si occupa di introdurre il protocollo SNMP utilizzato nel lavoro di tesi per la gestione dei dispositivi remoti e per il controllo della rete.

Nel Capitolo 4 è descritta la soluzione da noi proposta per implementare stati di basso consumo energetico in reti OSPF attraverso il protocollo SNMP. Sono descritti anche gli scenari applicativi di interesse individuati.

Il Capitolo 5 presenta gli strumenti utilizzati per avere a disposizione una piattaforma emulativa funzionale. Include la descrizione delle modifiche applicate ai sorgenti dei software utilizzati per ottenere le caratteristiche desiderate.

Nel capitolo 6 è presente la descrizione delle scelte progettuali e l'implementazione di una rete OSPF emulata che preveda uno stato di basso consumo energetico delle interfacce di rete. Sono presentati gli script ideati per la gestione dello *sleeping*, i test effettuati ed i risultati ottenuti.

Infine, il Capitolo 7 riassume il lavoro svolto e presenta delle valutazioni finali. Vengono inoltre illustrate dell proposte per dei possibili sviluppi futuri dell'approccio trattato.

Capitolo 1

Tecniche di risparmio energetico

Nel 2003, Gupta e Singh [29] hanno posto per primi la loro attenzione sulla riduzione dei consumi delle reti cablate. Hanno individuato nell'implementazione di stati di risparmio energetico all'interno dei dispositivi la soluzione più appropriata al problema. Hanno analizzato, inoltre, le eventuali modifiche che si sarebbero rese necessarie sia all'interno delle apparecchiature di rete sia a livello di protocolli di instradamento. Tra quelli citati vi è OSPF, di cui sono state analizzate le problematiche relative all'implementazione di strategie di sleeping nella rete. Gupta e Singh hanno mostrato anche la necessità di dispositivi in grado di supportare tali stati di basso consumo energetico ed appositi algoritmi di management per gestirli.

Sono stati perciò tra i primi a focalizzarsi sull'obiettivo del green networking, ossia progettare e gestire reti tenendo conto del loro consumo energetico. Da allora, molti sono stati i lavori relativi al miglioramento dell'efficienza nelle reti cablate.

In [28], ne viene presentata una interessante classificazione, basata su tre aspetti:

- **Target** (obiettivo): l'obiettivo della ricerca è l'elemento analizzato per ottenere la riduzione dei consumi energetici. È possibile dividere questi elementi in mezzi di comunicazione (fibra ottica), componenti (router, switch) e cammino, dove quest'ultimo è l'unione dei primi due;

- **Technique** (tecnica): vengono definite tre principali categorie di tecniche: la prima consiste nello studio di nuovi materiali e tecniche di fabbricazione. La

seconda consiste nello spegnimento dei dispositivi di rete in condizioni di basso carico. L'ultima riguarda l'integrazione dei componenti fisici.

- **Metric** (metrica): l'obiettivo specifico del metodo di risparmio energetico. Esempi di metrica sono il pattern di traffico, l'energia persa, il consumo energetico totale della rete.

1.1 Struttura di rete e consumo energetico

Nello studio delle tecniche di risparmio energetico, uno degli elementi chiave è la definizione del modello di consumo energetico. Può essere utile suddividere il problema analizzando i consumi di rete dal punto di vista dei tre principali livelli gerarchici da cui essa è costituita:

- **Accesso**: rappresenta gli utenti finali, che si collegano ai POP via wireless, ADSL, fibra. Comprende perciò i singoli elementi di rete delle abitazioni che si collegano poi ai concentratori;
- **Metro**: aggrega il traffico proveniente dagli utenti finali e lo incapsula in pacchetti IP in formato SONET/SDH per la trasmissione alla rete core;
- **Core**: è la parte centrale della rete, permette il collegamento tra utenti collegati da diverse reti di accesso. I nodi sono complessi router multipiattaforma e la rete ha una struttura mesh, permettendo cammini multipli tra nodi in modo da garantire elevata affidabilità in caso di guasti.

Il numero di core router e di router di bordo dipende dal rate di oversubscription usato dall'ISP, definito come $(M - N)/N$, dove M è il numero di utenti connessi ed N è il numero di utenti supportati simultaneamente a rate di picco. Per esempio, un oversubscription rate pari ad 1 significa che solo la metà degli utenti ha accesso contemporaneamente alla rete a piena velocità. In passato, quando la principale applicazione era il web-browsing, gli ISP potevano alzare molto l'oversubscription rate. Inoltre, il numero di utenti collegati in un dato istante era solitamente molto

inferiore al totale degli abbonati. Con la recente diffusione di applicazioni quali, per esempio, l'IPTV , il peer-to-peer e lo streaming, tale valore è stato abbassato a causa delle maggiori richieste di banda e dei più stringenti requisiti in termini di QoS.

Per minimizzare i costi di installazione richiesti per connettere ogni casa ad uno dei nodi di bordo, ogni concentratore è dotato di uno o più splitter passivi che dividono una singola fibra da una OLT (Optical Line Termination), che risiede in un nodo di bordo, in tante fibre. Ognuna di queste si collega ad una ONU (Optical Network Unit) connessa ad ogni abitazione. Ogni gruppo di ONU che condividono la stessa connessione verso un OLT comunica con quest'ultimo tramite un meccanismo a multiplazione di tempo: l'OLT assegna uno slot temporale ad ogni ONU. L'oversubscription rate può essere cambiato facendo variare il numero di ONU che condividono la connessione con l'OLT.

In [34] viene presentato un modello network-based per la misurazione del consumo energetico di Internet. Vengono calcolati i consumi della metro-access network e dei core routers, utilizzando dati relativi ad apparecchiature Cisco, e quelli relativi ai link WDM (Wavelength Division Multiplexing). Il rate di accesso medio è dato da $R(\rho+1)$, dove R è il picco del rate di accesso e ρ è l'oversubscription rate. Dalla Figura 1.1 emerge come il consumo sia imputabile principalmente ai router più che ai singoli link.

1.2 Tecniche di sleeping

Le reti degli Internet Service Providers sono dimensionate su valori di traffico di picco stimati. Il livello di traffico segue un comportamento giornaliero periodico, mentre il consumo energetico si mantiene pressoché costante [20] (v. Figura 1.2). Da qui nacque l'idea di implementare stati di basso consumo energetico (sleeping). Tramite le tecniche di sleeping, i componenti di rete vengono messi in uno stato di stand-by. In [29], si discute sulla possibilità di ridurre il consumo energetico di un router o di uno switch: si può pensare di mettere in sleeping alcuni o tutti i suoi componenti,

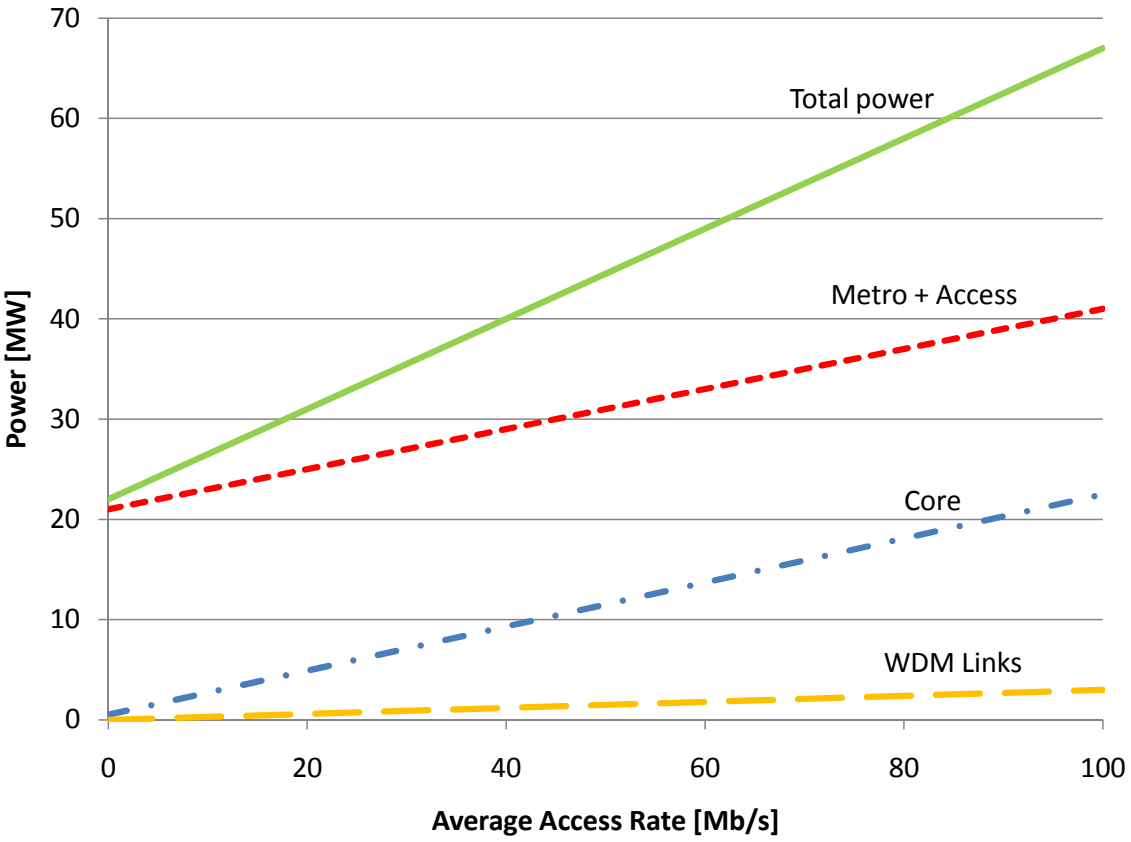

Figura 1.1: *Consumo energetico dei differenti livelli di rete per l'intero intervallo di tassi di accesso medi, con 2 milioni di abitazioni ed un picco del tasso di accesso di 100 Mbit/s.*

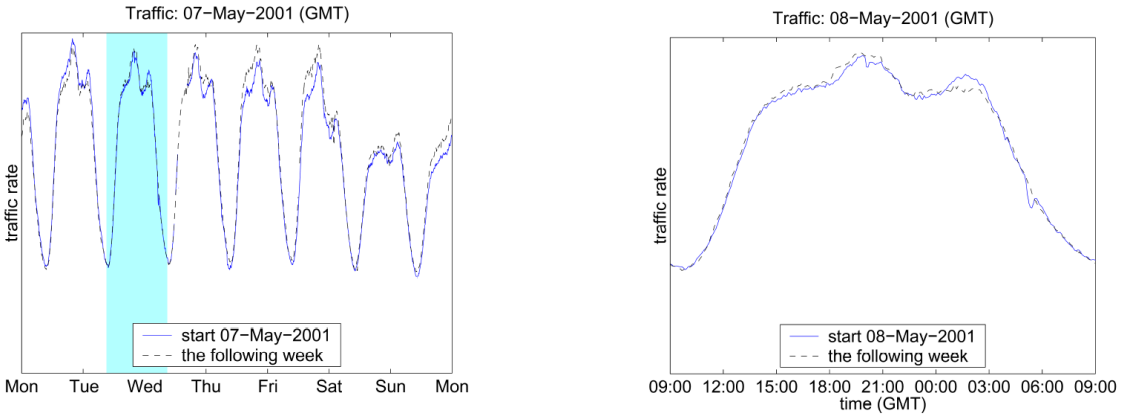

Figura 1.2: *Diagramma del traffico di una regione in due settimane consecutive. Il secondo grafico si concentra sulla regione evidenziata nel primo.*

quali la memoria, il processore principale, il bus, il line card processor/ASIC, la struttura di switching.

Le principali questioni che emergono riguardano il tempo in cui un componente può rimanere in sleeping, come prendere la decisione, e quali sono i router/switch che sono più adatti ad essere spenti.

Il risveglio dei componenti comporta un picco di consumo energetico, pertanto sarebbe opportuno avere una durata del periodo di sleeping sufficientemente lunga da compensarlo. Tale periodo deve anche essere superiore a quello necessario per il risveglio, e la sua massima durata è condizionata dal carico della rete e dalle caratteristiche del protocollo di routing utilizzato.

La seconda questione riguarda come prendere le decisioni, in modo non coordinato o coordinato. In un approccio non coordinato, un router/switch può monitorare il traffico solo sulle sue interfacce, mentre nel caso coordinato gli intervalli di sleeping sono calcolati basandosi su stime del flusso di traffico all'interno dell'AS (Autonomous System).

Per quanto riguarda gli elementi più adatti per poter essere messi in sleeping, occorre considerare la posizione dei router (core network, access network, ...), la tipologia di traffico gestito (intra/inter AS) ed il numero di rotte all'interno dell'AS.

In [17], viene presentato per la prima volta un modello matematico in PLI in grado di minimizzare il totale consumo energetico della rete data una certa domanda di traffico. Prevede la possibilità di spegnere router o link quando questi non sono necessari all'interno della rete. Elemento chiave per una gestione intelligente dei consumi è la variazione di traffico in un intervallo di tempo; occorre considerare congiuntamente i differenti scenari che ne derivano, per tenere conto dei vincoli sul routing, dell'overhead dovuto a segnalazione e del possibile degrado della qualità dovuto al cambio delle rotte. Si considera di poter dividere l'intervallo di tempo in tanti sottointervalli, ciascuno caratterizzato da un certo scenario di traffico. Ipotizzando di lavorare con router del tipo in Figura 1.3, le line cards possono essere accese o spente dalla piattaforma di management ed ognuno di esse ha associato un certo consumo energetico. Lo chassis può essere spento solo quando le line cards sono tutte spente. Vengono prese in considerazione due situazioni, la prima nella quale il

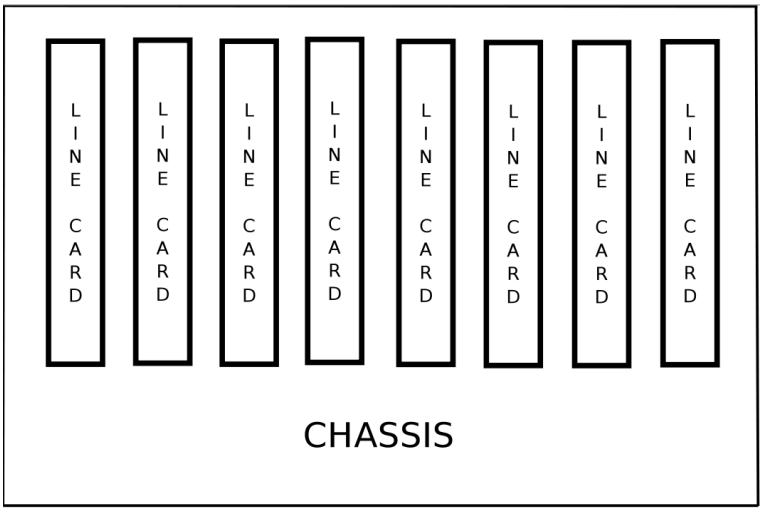

Figura 1.3: *Chassis semplificato di un router con 8 schede per i collegamenti*

routing rimane fisso, denominata FPARP (Fixed Power Aware Routing Problem), la seconda nella quale il routing cambia in funzione della nuova domanda di traffico e dello stato del dispositivo, detta VPARP (Variable Power Aware Routing Problem).

I test sono stati effettuati su una rete da 9 nodi ed una da 20. Ciò che è emerso in entrambi i casi è che l'utilizzo di un routing fisso limita la capacità del meccanismo di gestione dell'energia di reagire alle variazioni di traffico cambiando lo stato dei router e delle line cards. Vi è comunque un'apprezzabile riduzione dei consumi, che risulta però più marcata con il routing variabile: in questo caso il meccanismo risponde meglio alle variazioni di traffico, modificando la configurazione di rete. Ne deriva un consumo energetico circa proporzionale al carico di traffico, ed inferiore a quello del caso di routing fisso. Rispetto al caso con rete da 9 nodi, le differenze tra routing fisso e variabile si assottigliano con rete da 20 nodi ed il consumo totale aumenta.

In [35], viene presentata una nuova procedura di sleeping basata sulla predizione della lunghezza dei periodi di inattività (idle), basandosi sul fatto che il traffico Internet è caratterizzato dalla presenza di lunghe successioni di piccoli periodi di inattività. Quando il periodo di inattività predetto è più lungo di una soglia calcolata in precedenza, un dispositivo entra in stato di sleeping finchè un evento non ne

forza il risveglio. In caso contrario, il dispositivo rimane acceso. La struttura del periodo di inattività è illustrata in Figura 1.4. La soglia è $T = E + W$. La lunghezza

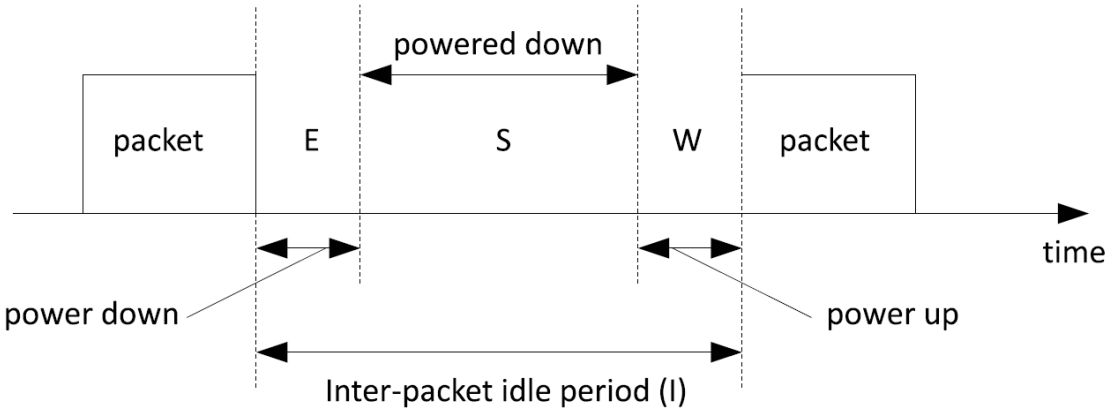

Figura 1.4: *Periodo di inattività tra due pacchetti*

del periodo di inattività è calcolata tramite l'espressione $P_{i+1} = P_i \times (1-\alpha) + I_i \times \alpha$; P_{i+1} è la nuova predizione, P_i è la predizione precedente, I_i è la lunghezza del periodo di inattività precedente e α è una costante ($0 \leq \alpha \leq 1$). La procedura completa è illustrata in figura ed è composta da quattro fasi: running (dispositivo attivo), running-wait (dispositivo attivo, contando un timer T1 e monitorando l'arrivo di pacchetti), sleep-wait (device in stato sleep, contando un timer T2 e monitorando l'arrivo di pacchetti) e sleep (device in stato sleep e monitorando l'arrivo di pacchetti). Lo schema dell'algoritmo di predizione è presentato in Figura 1.5, mentre il valore di Q è calcolato con un algoritmo specifico [43].

1.3 ALR- Adaptive Link Rate

È stimato che l'Ethernet Network Interface Controller (NIC) consumi molta energia, centinaia di milioni di dollari solo negli USA. L'energia usata dai link Ethernet sta crescendo rapidamente, così come il loro data rate, anche se sono sottoutilizzati. In [46], [19] e [41] gli autori si concentrano sulla possibilità di risparmiare considerevoli quantità di energia facendo lavorare i link ad un rate più basso durante i periodi

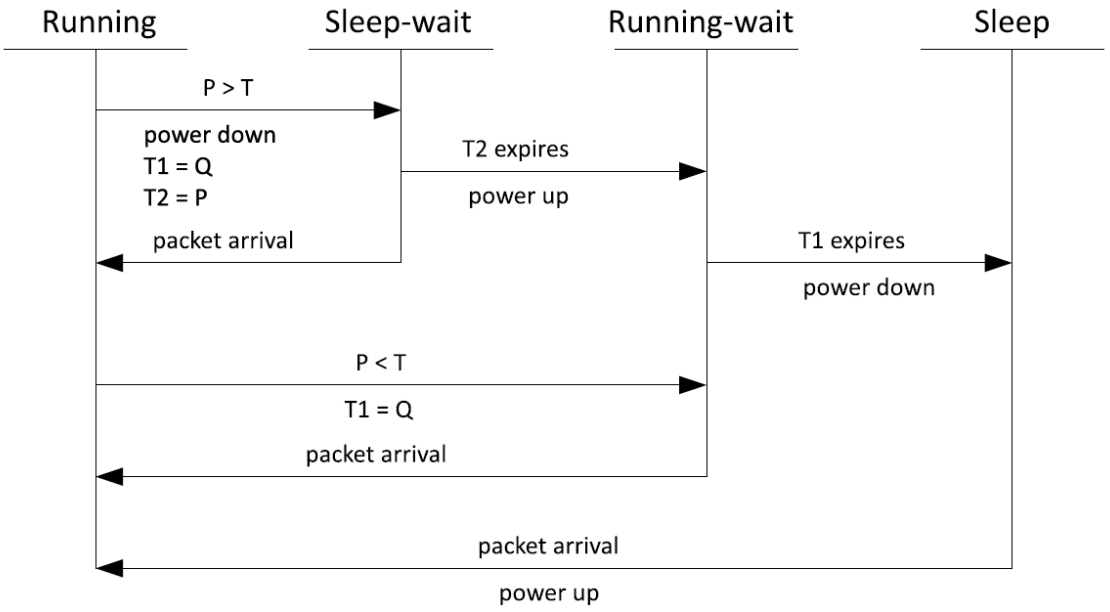

Figura 1.5: *Schema dell'algoritmo di sleeping predittivo presentato in [35]*

di basso utilizzo. Infatti un data rate inferiore implica direttamente un consumo inferiore: ad esempio, un link Ethernet ad 1 Gbps consuma circa 4 W in più di un link a 100 Mbps, ed un link a 10 Gbps può consumare fino a 20 W in più. Questa tecnica viene chiamata Adaptive Link Rate e consiste nel far variare il rate del link Ethernet per adattarsi al carico o all'utilizzo, rendendo il consumo energetico proporzionale all'utilizzo del link. ALR è pensato per usare i data rate esistenti di Ethernet ed è concepito per essere impiegato principalmente nei link edge. I miglioramenti chiave per sviluppare una tecnica ALR sono:

- la definizione di un meccanismo sufficientemente veloce per determinare come il link data rate deve essere cambiato;
- la definizione di una politica per determinare quando cambiare il data rate di un link, al fine di massimizzare il risparmio energetico e minimizzando il ritardo aggiuntivo sui pacchetti.

In [41] gli autori illustrano un meccanismo di switching (ALR MAC Frame Handshake Mechanism) e tre differenti politiche di switching (ALR Dual-Threshold

Policy, ALR Utilization-Threshold Policy, ALR Time-Out-Threshold Policy).

L'ALR MAC Frame Handshake Mechanism è un rapido handshake implementato utilizzando le trame MAC Ethernet, che può essere completato in meno di 100 microsecondi. Per prima cosa, il link che necessita di un aumento o diminuzione del suo data rate invia una richiesta di cambio del data rate usando un ALR REQUEST MAC frame. Successivamente, il link in ricezione apprende della richiesta di cambio di data rate con un ALR ACK MAC frame se è d'accordo a cambiare il data rate, oppure con un ALR NACK MAC frame se non è d'accordo. Dopo una risposta ALR ACK il data rate del link può essere cambiato ed il link risincronizzato. Il meccanismo è illustrato in Figura 1.6. Una buona politica ALR deve essere robusta, evitando oscillazioni del data rate. La politica più semplice è la ALR Dual-Threshold Policy, basata sull'occupazione del buffer in uscita. Vengono introdotte due soglie: se la lunghezza della coda nel buffer in uscita eccede un certo valore, allora il data rate deve essere incrementato. Se la medesima coda decresce poi al di sotto di una seconda soglia, il data rate va ridotto.

Migliori risultati nella riduzione dell'oscillazione del data rate possono essere ottenuti attraverso la ALR Utilization-Threshold Policy, dove, in aggiunta alle soglie sulla lunghezza dei buffer, si tiene in considerazione anche l'utilizzo del link. Pertanto i cambi di data rate avvengono se si verifica il superamento di due differenti soglie contemporaneamente, una relativa all'occupazione dei buffer ed una all'utilizzo del link.

L'ultima politica, la ALR-Time-Out-Policy, introduce due timer che indicano il minimo tempo di permanenza in stato alto o basso dopo una variazione del data rate.

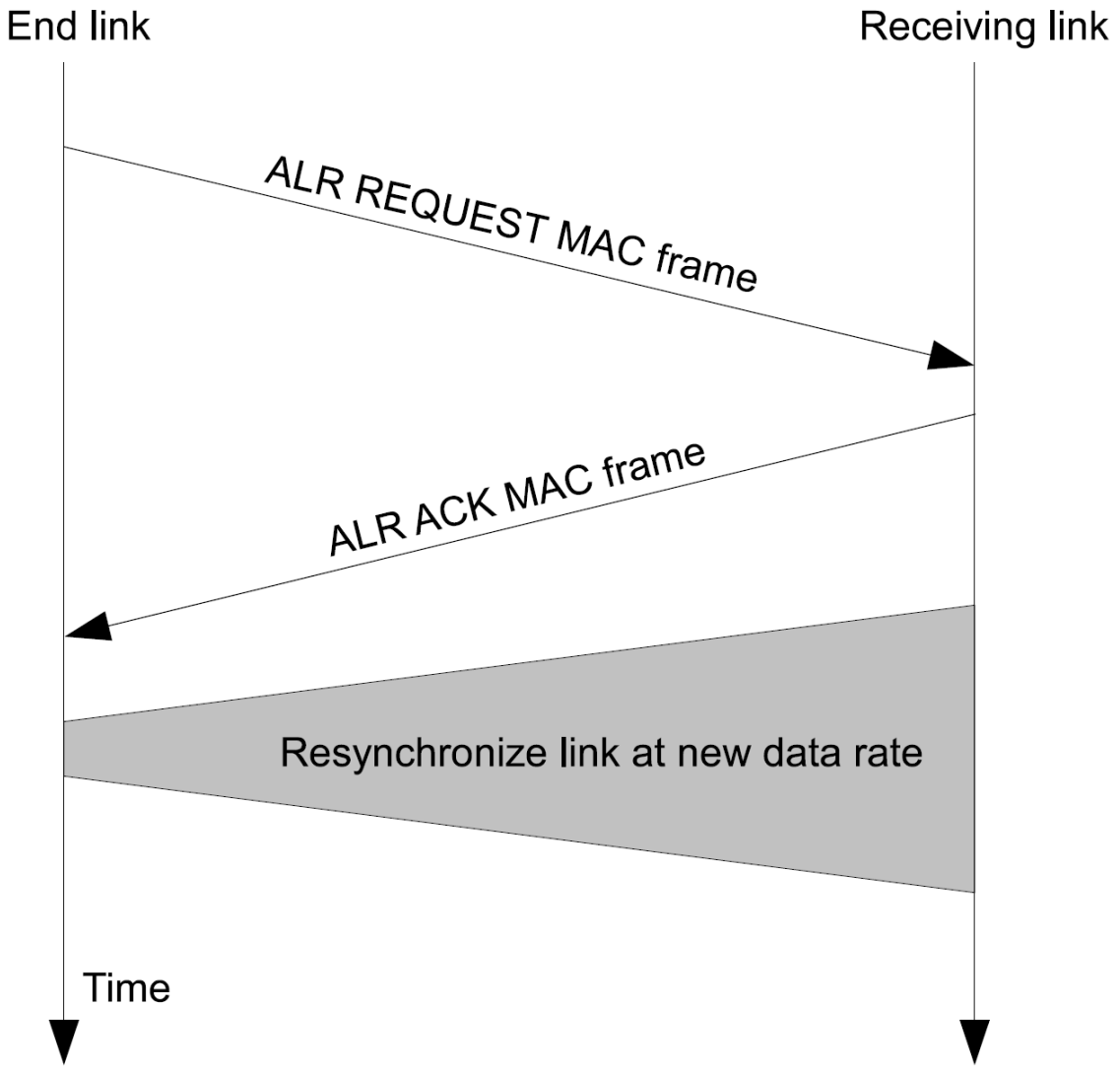

Figura 1.6: *Scambio di trame MAC nel meccanismo ALR [41]*

Capitolo 2

Open Shortest Path First (OSPF)

OSPF [39] è un protocollo di instradamento di tipo dinamico che garantisce una veloce reazione ai cambiamenti topologici; infatti, dopo un periodo di convergenza nel quale vengono scambiate informazioni tra i nodi della rete, il protocollo è in grado di ricalcolare dei nuovi percorsi esenti da cicli.

In un protocollo di tipo link-state come OSPF, ogni router mantiene una base di dati descrittiva della topologia dell'AS (Autonomous System) e tutti i router che fanno parte dell'AS ne possiedono una identica, nel caso in cui non sia presente una suddivisione in aree. Ogni router distribuisce l'informazione riguardo il suo stato attraverso il flooding; questo, però, può essere molto dispendioso in termini di informazioni aggiuntive da far transitare in rete. Con lo scopo di limitare questi scambi è prevista la possibilità di raggruppare porzioni di AS in aree per avere una suddivisione gerarchica della rete; il flooding avverrà così soltanto nell'area di appartenenza del router.

Ogni router esegue lo stesso algoritmo parallelamente, calcolando, a partire dalla propria base di dati, l'albero dei cammini più brevi per ogni destinazione nell'AS. È quindi possibile avere diverse rotte a pari costo per una data destinazione; in questo caso, il protocollo provvederà a distribuire il traffico equamente su questi percorsi. Il costo di una rotta è descritto da una metrica monodimensionale.

OSPF permette la configurazione di sottoreti IP; ogni rotta distribuita da OSPF possiede una destinazione con relativa maschera di rete. Le sottoreti possono essere

di lunghezza variabile: in caso di indirizzi corrispondenti con maschere di diversa lunghezza il pacchetto sarà instradato verso la migliore corrispondenza (ad esempio rete più lunga o destinazione più specifica).

Tutti gli scambi protocollari di OSPF sono autenticati facendo si che solo router fidati possano prendere parte all'instradamento dei pacchetti nell'AS. Possono essere utilizzati diversi schemi di autenticazione per le diverse sottoreti presenti.

Vediamo ora in modo più dettagliato gli aspetti importanti di questo protocollo:

- base di dati link-state;
- suddivisione in aree;
- protocollo di Hello e creazione di adiacenze;
- propagazione delle informazioni con pacchetti LSA;
- stati delle interfacce.

2.1 Base di dati link-state

Come già accennato questa base di dati è fondamentale per ogni nodo della rete in quanto permette di immagazzinare le informazioni da processare per il calcolo della tabella di instradamento. Le informazioni sono ricevute dagli altri nodi della rete sotto forma di Link State Advertisement (LSA); lo scambio di questi pacchetti porta alla generazione della base di dati link-state. In Figura 2.1 è illustrato un esempio molto semplice di AS e in Tabella 2.1 la base di dati corrispondente. La tabella di instradamento viene generata da ogni nodo della rete calcolando l'albero dei cammini minimi a partire dal grafo contenuto nella base di dati.

OSPF prevede la divulgazione di informazioni di instradamento esterne all'AS, che possono essere generate da altri protocolli o configurate staticamente. Queste informazioni vengono inoltrate inalterate attraverso l'AS. Sono previste due tipologie di metriche esterne:

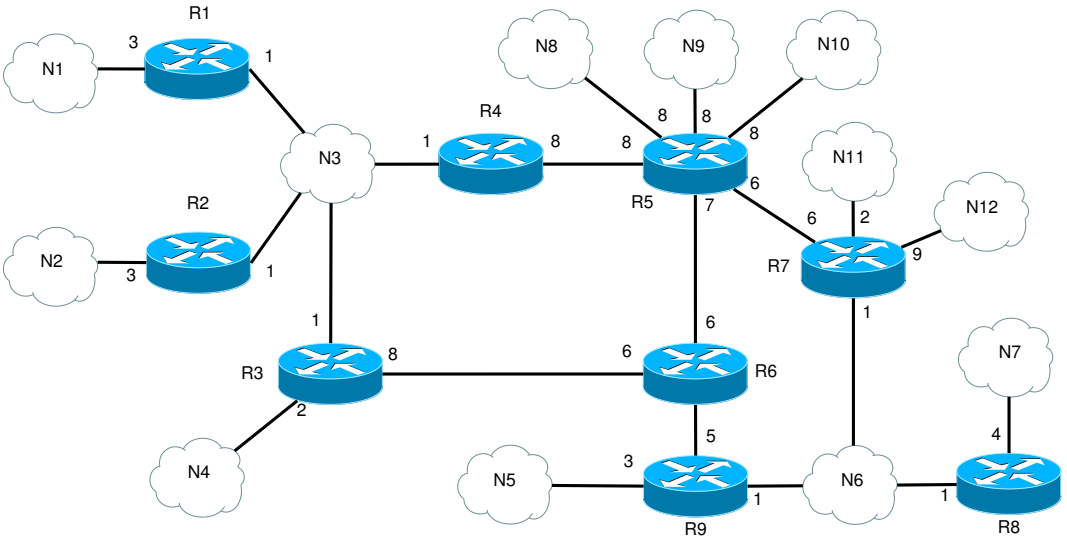

Figura 2.1: *Esempio di AS monoarea.*

1. metrica espressa nella stessa unità del costo delle interfacce OSPF;

2. metrica di un ordine di grandezza più grande. Ogni metrica di questo tipo viene considerata maggiore di qualsiasi costo di percorso interno, il che permette di considerare con maggior peso le rotte esterne e di evitare la conversione di queste metriche in quelle interne.

Queste metriche possono coesistere, in questo caso sarà sempre considerata la prima.

2.2 Suddivisione in aree

Reti interconnesse possono essere raggruppate, formando un'area OSPF. Ogni area esegue separatamente dalle altre l'algoritmo di instradamento e quindi avrà una base di dati link-state differente.

I router interni ad una precisa area non conoscono nulla riguardo i dettagli della topologia esterna ad essa, permettendo una notevole riduzione del traffico di instradamento. Con la presenza delle aree è ovvio quindi che un router deve mantenere

	R1	R2	R3	R4	R5	R6	R7	R8	R9	N3	N6
R1										0	
R2										0	
R3						6				0	
R4					8					0	
R5				8		6	6				
R6			8		7				5		
R7					6						0
R8											0
R9						7					0
N1	3										
N2		3									
N3	1	1	1	1							
N4			2								
N5									3		
N6							1	1	1		
N7								4			
N8					8						
N9					8						
N10					8						
N11							2				
N12							9				

Tabella 2.1: *Le colonne rappresentano le sorgenti, le righe le destinazioni ed il valore di riempimento la metrica OSPF.*

una base di dati link-state per ogni area cui esso è connesso. Router appartenenti alla stessa area avranno basi di dati link-state uguali.

Nell'AS dovrà quindi aver luogo un instradamento su due livelli:

1. instradamento intra-area, nel quale i pacchetti saranno instradati basandosi esclusivamente sulle informazioni interne all'area per evitare di processare informazioni sbagliate provenienti dall'esterno;

2. instradamento inter-area, necessario per l'inoltro di pacchetti tra aree distinte. I pacchetti dovranno necessariamente transitare dall'area di backbone. L'AS potrà quindi essere assimilato ad una topologia a stella dove il centro è costituito dall'area di backbone.

Oltre all'area di backbone, sono presenti altre tipologie di aree, come le aree di

transito e le aree stub (aree connesse al backbone attraverso un unico router). Nel caso in cui queste ultime siano supportate, assumono una particolare importanza per il protocollo OSPF. In alcuni AS, infatti, la maggior parte della base di dati link-state può riguardare informazioni esterne allo stesso AS che vengono inoltrate attraverso l'intero AS. Per risparmiare risorse in termini di traffico di instradamento e dimensioni della base di dati link-state è quindi possibile configurare delle aree come stub, impedendo così alle informazioni esterne di propagarsi all'interno di queste particolari aree. L'instradamento verso destinazioni esterne all'AS in queste aree viene ottenuto attraverso rotte prestabilite.

OSPF, inoltre, presenta la possibilità di configurare dei collegamenti virtuali. Questi sono visti dalla rete come collegamenti punto-punto tra diverse aree: l'utilizzo più importante che ne viene fatto è appunto quello di rendere adiacenti aree che non lo sono fisicamente, permettendo ad esempio di collegare al backbone un'area che non è ad esso adiacente. Non è però possibile usare collegamenti virtuali attraverso aree di tipo stub. Con la suddivisione dell'AS in aree, i router vengono distinti, in base alla loro funzione, in:

1. router interni: tutte le loro interfacce sono connesse con reti facenti parte della stessa area e dovranno quindi eseguire soltanto una copia dell'algoritmo di instradamento. Scelgono il punto migliore per uscire dall'area;

2. router di bordo area: sono connessi con diverse aree dell'AS, e per questo dovranno eseguire una copia dell'algoritmo di instradamento per ogni area alla quale sono interfacciati. Diffondono in un'area informazioni riassuntive raccolte nelle altre;

3. router del backbone: hanno almeno un'interfaccia di rete connessa con l'area di backbone;

4. router di bordo AS: scambiano informazioni di instradamento con router appartenenti ad altri AS. Il percorso verso ogni router di questa tipologia deve

essere conosciuto da tutti. Questa classificazione è indipendente da quelle sopracitate in quanto un router di questo tipo può essere interno, di bordo oppure facente parte del backbone.

2.3 Protocollo di Hello e creazione di adiacenze

Il protocollo di Hello ha l'importante funzione di stabilire e mantenere le relazioni di vicinato tra i diversi nodi della rete, assicurando inoltre che la comunicazione tra vicini sia bidirezionale. Il suo funzionamento prevede l'invio periodico di pacchetti, detti di Hello, da tutte le interfacce del router. Questi pacchetti sono caratterizzati da diversi campi che permettono di definire la periodicità e il tempo di validità del pacchetto stesso, oltre ad una lista dei vicini da cui si è ricevuto un pacchetto di Hello valido. Nel momento in cui un router legge il proprio identificativo nel pacchetto di Hello ricevuto, la comunicazione può essere definita bidirezionale.

OSPF crea le adiacenze tra router vicini con lo scopo di scambiare informazioni di instradamento; tuttavia, non tutti i vicini diventano adiacenti. Il protocollo di Hello è responsabile dell'elezione del Designated Router (DR) e del Backup Designated Router (BDR). L'elezione avviene per mezzo dell'informazione contenuta in un campo del pacchetto di Hello che rappresenta la priorità del router che l'ha generato. Quando un router si collega ad una rete controlla l'esistenza di un DR: se ne è già stato nominato uno, questo viene accettato indipendentemente dalla sua priorità. In caso contrario, il router in questione diventa DR se ha la priorità più alta all'interno della rete. Questo processo garantisce che i cambi di DR siano poco frequenti. Un cambio di DR sarebbe infatti dannoso per la rete, dando luogo ad una serie di scambi di messaggi protocollari per poter ristabilire le adiacenze create con il DR precedente. Anche il BDR viene eletto con una procedura analoga.

Dopo l'elezione del DR, la scoperta di un vicino e la creazione di una connessione bidirezionale, il protocollo decide se instaurare o meno una relazione di adiacenza. Il primo passo è la sincronizzazione delle basi di dati link-state dei vicini. Ogni router invia dei pacchetti descrittivi della sua base di dati ai propri vicini: se un

router si accorge che la copia di LSA in suo possesso è meno recente, lo annota per richiederla in un secondo momento. Questo è il processo di scambio di base di dati, alla fine del quale ogni router ha una lista di LSA di cui i vicini hanno delle copie più aggiornate. A questo punto saranno mandati dei pacchetti di richiesta di aggiornamento e, quando tutte le richieste saranno state soddisfatte, le basi di dati saranno sincronizzate ed i router saranno completamente adiacenti.

Il DR occupa un ruolo fondamentale per il protocollo di instradamento. Esso svolge due importanti funzioni:

1. origina un network-LSA che contiene la lista dei router correntemente connessi alla rete;

2. diventa adiacente con tutti i router della rete avendo quindi un ruolo centrale nel processo di sincronizzazione delle basi di dati.

Anche il BDR ha un ruolo importante, in quanto rende meno oneroso il processo di transizione verso un nuovo DR: anch'esso è infatti adiacente a tutti i router della rete e quindi pronto a prendere il posto del DR in caso di guasti.

2.4 Propagazione delle informazioni, i pacchetti LSA

In OSPF i pacchetti di instradamento, fatta eccezione per quelli di Hello, sono mandati solamente ai nodi adiacenti. In base a questo si può affermare che questi pacchetti viaggiano attraverso un solo *salto* IP, escludendo le adiacenze virtuali.

Ogni router dell'AS origina uno o più pacchetti LSA che formeranno la base di dati link-state. Questi sono caratterizzati da un'intestazione comune per poi differenziarsi in alcune tipologie in base alla funzione che svolgono. L'intestazione contiene informazioni utili a identificare univocamente ogni LSA e alcuni campi necessari per il funzionamento del protocollo e per il suo aggiornamento. Un esempio sono i campi che definiscono l'*età* di un LSA e il suo numero di sequenza, che servono

per considerare solo i pacchetti più recenti in fase di aggiornamento delle basi di dati. Vediamo ora più in dettaglio le differenti topologie associate al loro codice identificativo:

1. Router-LSA: un router ne genera uno per ogni area di cui fa parte, descrive gli stati delle proprie interfacce connesse con l'area stessa. Questi pacchetti rimangono nell'area in cui sono stati generati e non possono propagarsi oltre. Il router è in grado di indicare se si tratta di router di bordo area oppure di bordo AS settando degli opportuni bit. Anche la tipologia dei collegamenti annunciati dal LSA è descritta nello stesso modo;

2. Network-LSA: originato da ogni rete di transito (rete che possiede almeno due router), descrive tutti i router che fanno parte della rete in questione. Questo LSA è originato dal DR nel caso in cui questi sia completamente adiacente con almeno un altro router della rete ed è propagato soltanto all'interno dell'area che contiene la rete di transito;

3,4. Summary-LSA: originato dai router di bordo area, descrive destinazioni esterne all'area in cui viene generato, ma comunque interne all'AS. Viene propagato soltanto internamente all'area di provenienza. Le rotte descritte da questo pacchetto sono determinate applicando un particolare algoritmo alla tabella di instradamento. Summary-LSA di tipo 3 descrivono delle rotte verso reti mentre il tipo 4 descrive rotte verso router di bordo AS;

5. AS-external-LSA: originato dai router di bordo AS, descrive quindi le rotte verso destinazioni esterne all'AS stesso. Ne viene generato uno per ogni rotta esterna che il router ha appreso, sia attraverso altri protocolli che attraverso informazioni di configurazione. Questa tipologia di LSA è l'unica che viene propagata in tutto l'AS, ovviamente fatta eccezione per le aree stub.

6. Group Membership LSA [38]: definito per le estensioni multicast di OSPF, non molto usato. Potrà essere riassegnato in futuro;

7. Not So Stubby Area LSA [30]: la NSSA è una tipologia particolare di area stub, può importare rotte esterne e propagarle verso altre aree. Comunque non può ricevere informazioni su rotte esterne all'AS da altre aree. Questo LSA è generato da un router di bordo AS, utilizzato per mandare informazioni di instradamento da ridistribuire nell'AS. Una volta ricevuto da un router di bordo area, questi lo trasforma in un normale LSA di tipo 5 per il suo inoltro;

8. External Attributes LSA [40]: originariamente pensato per sostituire IBGP (Internal Border Gateway Protocol) negli AS di transito. Di fatto la maggior parte delle implementazioni di OSPF non ha mai incluso questa funzionalità;

9,10,11. Opaque-LSA [18]: questo tipo di LSA è molto particolare in quanto può essere utilizzato direttamente da OSPF oppure indirettamente da applicazioni esterne. Il nome opaco deriva dal fatto che sono in grado di trasportare informazioni provenienti anche da livelli superiori dello stack protocollare. Il tipo 9 denota un collegamento in ambito locale, il 10 denota un'area in ambito locale mentre il tipo 11 è inoltrato attraverso l'intero AS.

É importante capire quando è necessario generare un LSA per mantenere aggiornate le basi di dati, questo avviene in seguito a questi eventi:

1. il campo *età* di uno dei router che ha originato il LSA raggiunge il particolare valore di aggiornamento, viene generata una nuova istanza con gli stessi contenuti;

2. cambia lo stato di un'interfaccia;

3. cambia il DR di una rete connessa;

4. uno dei vicini cambia il suo stato uscendo o entrando dalla piena adiacenza;

5. viene aggiunta o eliminata una rotta intra-area nella tabella di instradamento;

6. viene aggiunta o eliminata una rotta inter-area nella tabella di instradamento;

7. il router diventa parte di una nuova area;
8. viene cambiato lo stato di un router attraverso cui è configurato un collegamento virtuale;
9. una rotta esterna viene aggiunta;
10. un router cessa di essere un router di bordo AS.

2.5 Stati delle interfacce

La finalità del lavoro è quella di proporre una soluzione, all'interno di una rete basata su OSPF, che renda possibile l'accensione e lo spegnimento dei nodi e delle interfacce al fine di produrre un risparmio energetico. A questo proposito è chiara l'importanza di studiare i vari stati che possono assumere le interfacce di un router OSPF.

Si definisce interfaccia OSPF la connessione tra un router ed una rete: tutti i pacchetti del protocollo di instradamento originati dall'interfaccia di un determinato router dovranno essere etichettati con l'identificativo dell'area a cui fa riferimento l'interfaccia stessa. Un'interfaccia di un router OSPF potrà assumere i seguenti stati:

- Down: stato iniziale, l'interfaccia non è utilizzabile;
- Loopback: non utilizzabile per un regolare traffico dati, utile per ricavare informazioni sulla qualità attraverso Internet Control Message Protocol (ICMP) o bit error test. Per questo deve comunque poter ricevere pacchetti IP;
- Waiting: stato di monitoraggio dei pacchetti di Hello per cercare di determinare il DR (BDR);
- Point-to-point: stato operazionale, l'interfaccia è connessa ad una rete point-to-point o a un collegamento virtuale;

- DR Other: l'interfaccia è in una rete in cui è già stato eletto il DR;
- Backup: il router è il BDR della rete;
- DR: il router è il DR della rete.

Per poter gestire l'accensione e lo spegnimento delle interfacce si è pensato di aggiungere un ulteriore stato possibile, lo stato di Sleeping. Questo stato permetterebbe a un'interfaccia di non essere considerata operativa dalle altre, con il relativo risparmio di traffico IP. In questo stato si rimarrebbe in ascolto di particolari messaggi che consentirebbero il risveglio dell'interfaccia in caso di bisogno. Lo schema degli stati con l'aggiunta di quello di Sleeping sarebbe quello illustrato in Figura 2.2.

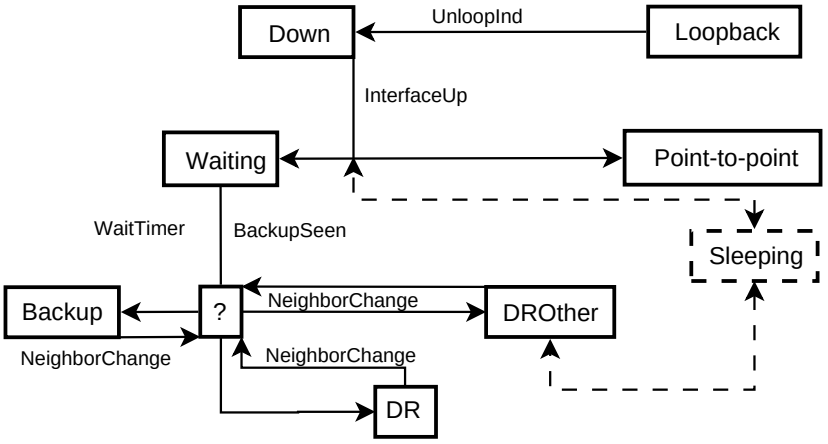

Figura 2.2: *Possibili stati delle interfacce OSPF.*

Capitolo 3

Network Management

Lo sviluppo delle reti di telecomunicazione negli ultimi decenni ha portato ad un inevitabile incremento delle loro dimensioni. Se fino a poco tempo fa il concetto di rete era legato ad ambiti locali, adesso le reti coprono intere aree geografiche. Questa crescita in dimensioni e in numero di host collegati crea agli amministratori notevoli problemi di gestione e manutenzione della rete; infatti se una rete si estende in un'area molto vasta non è detto che l'amministratore sia in grado di raggiungere fisicamente e in qualsiasi momento tutti gli host e i nodi di rete. Per ovviare a questi problemi sono nati protocolli e programmi in grado di monitorare la rete e di prevenire possibili guasti ai dispositivi collegati o situazioni indesiderate nella rete.

Nel 1988 nasce il protocollo SNMP che viene pensato come punto di partenza su cui sviluppare dei sistemi che siano in grado di risolvere e prevedere tutti i problemi che nascono dalla gestione di una rete. La versatilità di questo protocollo gli ha permesso di trovare da subito un riscontro positivo dal mondo degli sviluppatori e dalle aziende del settore.

3.1 SNMP

Simple Network Management Protocol (SNMP) appartiene alla suite di protocolli Internet definita dalla Internet Engineering Task Force (IETF). Il protocollo opera al livello Applicazione del modello Open Systems Interconnection (OSI) e consiste in una serie di standard per la gestione e la supervisione di apparati di rete come

router, switch, server, workstation, stampanti, e molto altro.
SNMP è passato attraverso alcune revisioni, attualmente vi sono 3 versioni:

- **SNMPv1** (RFC 1155, RFC 1156 e RFC 1157), rappresenta la prima implementazione del protocollo SNMP;
- **SNMPv2** (RFC 1441 – RFC 1452), aggiunge nuove funzionalità e risolve alcune debolezze presenti nella prima versione.
- **SNMPv3** (RFC 2571 – RFC 2575), è lo standard finale, ma al momento raramente utilizzato.

SNMPv1 è la versione primordiale, pienamente compatibile con gli standard della IETF. È il protocollo di management più diffuso in quanto può essere implementato in dispositivi relativamente semplici, tuttavia questa prima versione ha molte limitazioni e svantaggi [25]. Per garantire la sicurezza i dispositivi da gestire sono associati a delle community, che altro non sono che password. La sicurezza tuttavia non è garantita pienamente perché queste stringhe vengono trasmesse in chiaro e ciò ha condotto alla necessità di una versione successiva del protocollo. Inoltre, SNMPv1 non permette l'invio efficiente di grandi quantità di dati e supporta un solo meccanismo di polling che viene sempre iniziato dalla stazione di gestione; un agente può inviare messaggi in modo asincrono tramite *trap*.

SNMPv2 ha introdotto miglioramenti in performance, sicurezza e comunicazione, introducendo tra l'altro nuove funzionalità come la possibilità di richiedere più informazioni con un'unica richiesta, l'invio di messaggi asincroni con riscontro e una miglior definizione delle trap. Con SNMPv2 vengono introdotti l'uso di un sistema di cifratura per i dati e l'uso di un sistema di autenticazione. Inoltre, viene offerta la possibilità di usare time stamp per mantenere la sincronizzazione dei clock ed evitare possibili attacchi di repliche.
Questi nuovi meccanismi permettono l'implementazione di tre differenti livelli di sicurezza:

- senza autenticazione nè crittografia dei dati;
- autenticazione con MD5 ma assenza di crittografia dei dati;
- autenticazione con MD5 e crittografia dati.

Lo standard finale tuttavia portò alla definizione di **SNMPv2c**, dove la lettera "c", solitamente omessa, stà per community, e di SNMPv2p. Questo stà a significare che il primo standard utilizza lo stesso meccanismo di autenticazione di SNMPv1 mantenendo le nuove funzionalità grazie all'adozione della nuova PDU, mentre il secondo implementa tutti i nuovi meccanismi di sicurezza.

Con SNMPv3 non vengono introdotti grossi cambiamenti al protocollo ma i problemi di sicurezza vengono definitivamente risolti. Esso fornisce tre servizi importanti: autenticazione, privacy e controllo di accesso. SNMPv3 è attualmente la versione standard, tuttavia lo standard de facto prevede l'utilizzo della versione 2 (SNMPv2c), di seguito dettagliata.

3.2 Architettura

L'architettura di un sistema di gestione della rete è concettualmente identica alla semplice analogia di organizzazione umana. Identifichiamo tre componenti principali nell'architettura di gestione della rete [26]: un'**entità di gestione**, i **dispositivi da gestire** e un **protocollo di gestione** della rete.

Un sistema SNMP è composto da una **Stazione di Gestione** (*Management Station*) e dagli **Agenti SNMP** (*SNMP Agent*) in funzione sui dispositivi di rete. Le prime interrogano attraverso un meccanismo a polling gli agenti, i quali rispondono inviando le informazioni richieste (Figura 3.1). Gli oggetti gestiti dagli agenti sono raccolti, in ogni dispositivo, in un database chiamato **MIB** (Management Information Base) secondo la struttura definita nella **SMI** (Structure Management Information). L'agente non è altro che un software eseguito sul dispositivo di rete controllato. Può essere un programma a sé, per esempio un demone in ambiente Unix, oppure può essere integrato nel sistema operativo del dispositivo.

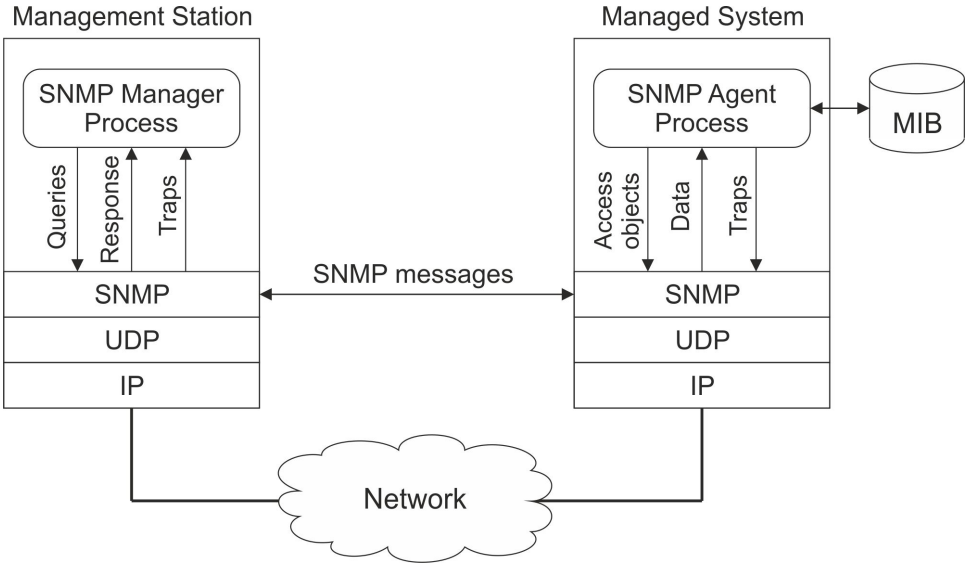

Figura 3.1: *Architettura di un sistema SNMP*

SNMP è dunque un'implementazione di tipo client/server. L'applicazione client, chiamata stazione di gestione, crea una connessione virtuale verso un'altra applicazione server, chiamata agente SNMP, che gira su un dispositivo remoto. Esiste un caso particolare dove l'apparato di rete non viene interrogato, ma è esso stesso a generare il messaggio ed inviarlo alle stazioni di gestione. È infatti possibile configurare gli agenti in modo da inviare un particolare messaggio al verificarsi di determinati eventi, ovvero impostare le cosidette trap. Grazie all'impostazione delle trap è possibile, ad esempio, sapere quando un'interfaccia di rete smette di funzionare: al verificarsi del guasto, l'agente SNMP che esegue il monitoraggio dell'apparato invia alla Management Station un messaggio che identifica il problema.

È importante ricordare che le interrogazioni con meccanismo a polling e le trap possono avvenire in qualsiasi momento, anche contemporaneamente. Non ci sono quindi restrizioni su quando una stazione di gestione può interrogare un agente o quando un agente può inviare una trap.

SNMP utilizza come protocollo di trasmissione UDP, in modo da ottenere migliori performance e minore overhead della rete. In particolare, viene utilizzata la porta UDP 161 per le interrogazioni e le risposte, mentre come destinazione dei messaggi

trap SNMP, generati dagli agenti SNMP, viene usata la porta UDP 162. A causa della natura connectionless di UDP, in cui non sono previsti riscontri a livello protocollare, sta a SNMP determinare se un datagramma è stato ricevuto correttamente o è andato perso. Questa operazione viene svolta con l'ausilio di semplici timeout e con dei messaggi di risposta previsti dal protocollo SNMP. L'unico tipo di messaggio che non prevede un riscontro è la trap.

3.3 Management Information Base

SNMP utilizza una chiara separazione fra il protocollo di gestione e la struttura dell'oggetto gestito. Nell'architettura SNMP, per ogni sottosistema è definita una base dati detta MIB (Management Information Base), la quale rappresenta lo stato del sottosistema gestito, o meglio, una proiezione di tale stato limitata agli aspetti di cui si vuole consentire la gestione. Ogni modifica al MIB causa un corrispondente mutamento nello stato del sottosistema rappresentato, e viceversa. Garantire questa proprietà del MIB è la funzione principale del *subagent* che la gestisce.

L'accesso al MIB, in lettura e scrittura, rappresenta l'interfaccia fornita alla stazione di gestione per controllare il sistema. Ogni MIB, pur variando nei contenuti specifici, ha la medesima struttura generale e i medesimi meccanismi di accesso da parte della stazione di gestione per la lettura e la scrittura dei dati. Grazie alla connessione causale del MIB è quindi possibile, per la stazione di gestione, agire sullo stato del sottosistema in un modo che è largamente indipendente dalle procedure concrete che devono essere messe in atto per estrarre le informazioni di stato rappresentate nel MIB, o attuare le modifiche di stato a seguito di cambiamenti dei contenuti nel MIB. Così, per esempio, si potrebbe avere un dato di MIB che rappresenta l'indirizzo IP del sistema gestito; per modificare tale indirizzo, alla stazione di gestione, è sufficiente accedere al MIB sovrascrivendo il dato corrispondente, prescindendo dei dettagli di come una tale modifica venga poi concretamente attuata sul sistema gestito. Lo stesso discorso vale anche per l'accensione o lo spegnimento di un'interfaccia.

Esistono diversi MIB, dipendenti dal protocollo e dal dispositivo. Per quanto riguarda SNMP, sono di particolare importanza MIB-I e MIB-II. MIB-I si riferisce alla definizione iniziale contenuta nel Rfc 1066, mentre MIB-II si riferisce alla definizione attuale, Rfc 1213. Un agente SNMP può implementare più MIB, ma tutti gli agenti devono implementare necessariamente MIB-II.

3.3.1 Structure Management Information

La Structure Management Information (SMI) è una notazione che definisce come devono essere strutturate le informazioni e la loro gerarchia per essere inserite nel MIB e quindi gestite dal protocollo SNMP. Questo linguaggio di definizione è necessario per assicurare che sintassi e semantica dei dati di gestione della rete siano ben definiti e privi di ambiguità. In [37] sono specificati i tipi di dati base nella SMI per il linguaggio di definizione dei moduli MIB. Questi dati possono essere numeri interi, con e senza segno, stringhe di byte, indirizzi IP, contatori di vario tipo, e misuratori di tempo. Oltre ai tipi di dati il linguaggio di definizione dei dati SMI fornisce anche dei costrutti di livello più alto. Come esempio vediamo il costrutto OBJECT-TYPE, usato per specificare il tipo di dati, lo stato e la semantica di un oggetto da gestire, in questo caso ifAdminStatus (Codice 3.1). Esso definisce un intero che tiene traccia dello stato amministrativo di un'interfaccia. La frase MAX-ACCESS specifica se l'oggetto da gestire può essere letto, scritto, creato o se il suo valore può essere inserito in una notifica. La frase STATUS indica se la definizione di un oggetto è attuale e valida, obsoleta o recuperabile. La frase DESCRIPTION contiene una definizione testuale dell'oggetto leggibile dall'uomo.

Codice 3.1: *Oggetto ifAdminStatus contenuto in IF-MIB*

```
ifAdminStatus OBJECT-TYPE
SYNTAX  INTEGER {
            up(1),       -- ready to pass packets
            down(2),
            testing(3)   -- in some test mode
        }
MAX-ACCESS  read-write
STATUS      current
```

```
DESCRIPTION
        "The desired state of the interface. The testing(3)
        state indicates that no operational packets can be
        passed. When a managed system initializes, all
        interfaces start with ifAdminStatus in the down(2)
        state. As a result of either explicit management
        action or per configuration information retained by
        the managed system, ifAdminStatus is then changed to
        either the up(1) or testing(3) states (or remains in
        the down(2) state)."
::= { ifEntry 7 }
```

La IETF ha avuto il suo da fare nella standardizzazione dei moduli MIB associati con router, host e altri equipaggiamenti di rete [26]. Poiché al momento del rilascio di SNMP esistevano diversi moduli MIB basati sugli standard e altrettante specifiche (private) dei costruttori dei moduli MIB, la IETF aveva la necessità di trovare un modo di identificare e battezzare i moduli standardizzati, così come di gestire gli specifici oggetti all'interno di un modulo. Per fare questo la IETF adottò una struttura standardizzata per l'identificazione degli oggetti che era già stata utilizzata dall'International Organization for Standardization (ISO).

La struttura di identificazione dell'oggetto adottata dall'ISO è parte del linguaggio ASN.1 ed ha una struttura ad albero (Figura 3.2) che raggruppa oggetti MIB in una gerarchia al fine di definire degli oggetti che siano facilmente gestibili. Ad ogni nodo dell'albero viene associato un nome ed un numero, che consente di creare un percorso univoco per ogni oggetto del MIB, a partire dalla radice. Questo percorso viene chiamato OID (Object IDentifier) e identifica un oggetto in maniera non ambigua; OID non arriva necessariamente al dettaglio di una sola variabile. Quando ad esempio si sottomette un'operazione SNMP di tipo Get, la stazione di gestione invia l'OID all'agente, il quale determina se l'OID è supportato dal sistema e, in caso di riscontro positivo, restituisce le informazioni associate all'oggetto stesso. Riprendendo l'esempio precedente riferito all'oggetto ifAdminStatus, avremo che il percorso all'interno dell'albero, ossia l'OID, sarà:

`1.3.6.1.2.1.2.2.1.7.`

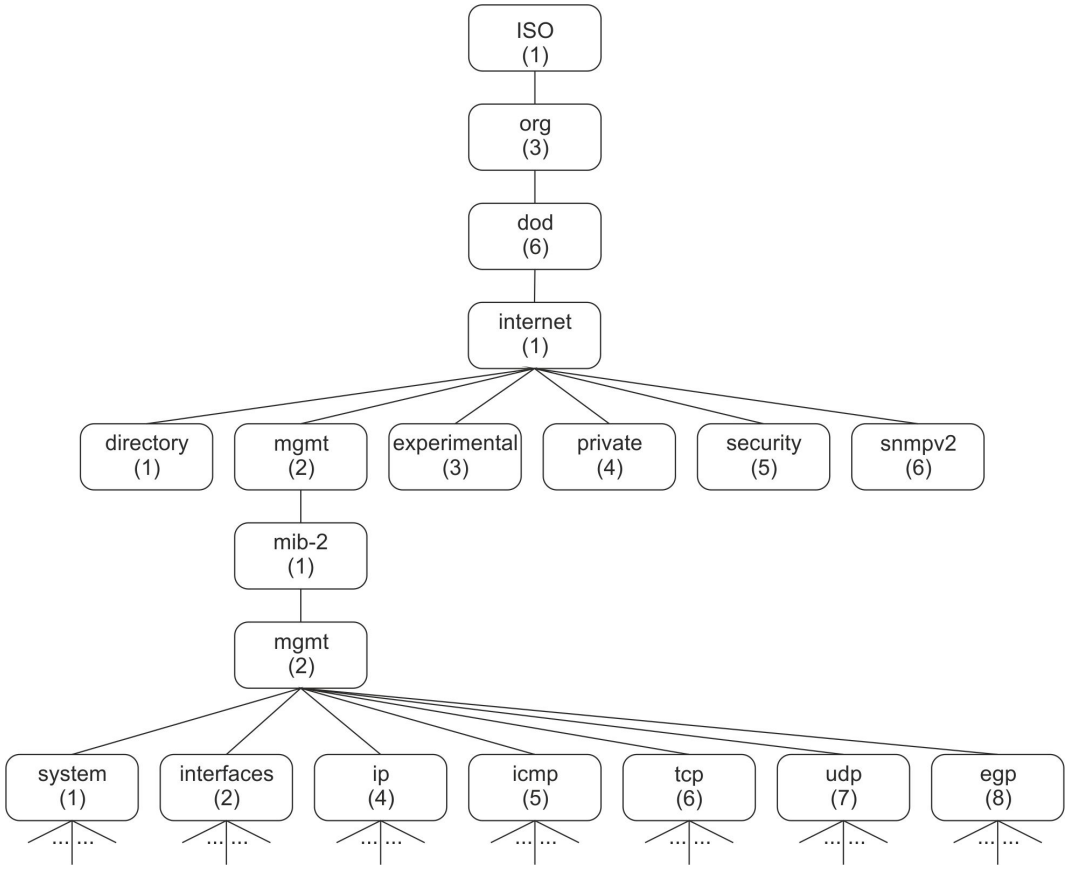

Figura 3.2: *Albero delle MIB*

3.4 Messaggi SNMP

La PDU (Protocol Data Unit) definita in SNMP è costituita da tre campi:

- **Version number**: descrive tramite un numero intero la versione di SNMP utilizzata nel messaggio;
- **Community String**: identifica la community SNMP del mittente del messaggio;
- **SNMP PDU**: rappresenta il corpo del messaggio.

L'insieme degli apparati di rete gestiti dal protocollo SNMP appartiene ad una community. La community rappresenta un identificativo che permette di garantire

la sicurezza delle interrogazioni SNMP tramite i meccanismi di controllo dell'accesso previsti dal protocollo. Un agente SNMP risponde solo alle richieste di informazioni effettuate da una stazione di gestione appartenente alla stessa community.
Esistono tre diversi tipi di community:

- **monitor**: permette di lavorare in sola lettura, quindi di effettuare solamente interrogazioni agli agenti (il cui nome di community deve corrispondere a quello della stazione di gestione che ne ha fatto la richiesta);

- **control**: permette tramite gli agenti SNMP di effettuare delle operazioni in lettura/scrittura sul dispositivo, quindi di variarne delle impostazioni sempre previo controllo di sicurezza;

- **trap**: permette ad un agente di inviare un messaggio trap SNMP alla stazione di gestione secondo la propria configurazione.

Spesso i nomi di community di default predefiniti sono *public* per le community di sola lettura e *write* o *private* per quelle in lettura/scrittura.

SNMP utilizza sette tipi di messaggi di base per svolgere il proprio lavoro. Ogni messaggio è definito in PDU separate, e precisamente:

- **GetRequest**: generata dalla stazione di gestione per interrogare un oggetto in un MIB su un agente SNMP;

- **GetNextRequest**: è utilizzata dalla stazione di gestione per leggere in modo sequenziale gli oggetti in un MIB ed è concettualmente equivalente ad una sequenza di GetRequest;

- **GetBulkRequest**: permette alla stazione di gestione di leggere un MIB o una sua parte con una unica richiesta anziché utilizzare più messaggi GetNext;

- **SetRequest**: utilizzata per la modifica delle informazioni in lettura/scrittura contenute nel MIB di un agente da parte della stazione di gestione;

- **Response**: generata dall'agente in risposta ad ognuna delle precedenti per inviare informazioni di risposta alla stazione di gestione (in caso di Get) o per confermare l'operazione richiesta (in caso di Set);

- **Trap**: messaggio asincrono per la segnalazione da parte di un agente ad una stazione di gestione di eventi considerati come eccezioni al normale funzionamento. Per l'operazione di trap sono previsti un numero limitato di eventi predefiniti per i quali è definito l'utilizzo di questa operazione.
  Alcune trap sono predefinite:

  - **coldStart**: generata quando l'agente SNMP si reinizializza e la configurazione è stata cambiata (Es. reboot del sistema);
  - **warmStart**: generata quando l'agente SNMP si reinizializza ma senza cambiamenti nella configurazione;
  - **linkDown**: generata quando il collegamento con l'agente non funziona correttamente;
  - **linkUp**: generata quando il collegamento con l'agente viene ripristinato;
  - **authenticationFailure**: generata da un'autenticazione con l'agent non andata a buon fine (Es. nome di community errato);
  - **egpNeighborLoss**: generata da problemi di EGP (Exterior Gateway Protocol - utilizzato dai router);
  - **enterpriseSpecific**: evento definito dal produttore del dispositivo;

- **InformRequest**: si tratta di un messaggio di trap con riscontro introdotto in SNMPv2. SNMP per inoltrare i messaggi si appoggia a UDP, il quale non assicura la corretta ricezione del messaggio. InformRequest risolve questo problema facendo si che il destinatario della notifica invii un messaggio di risposta.

Le varie versioni di SNMP si differenziano anche per il formato di PDU utilizzato. Il formato di una PDU SNMPv2, mostrato in Figura 3.3, è descritto in [42] ed è simile al formato definito in SNMPv1.

- **PDU type**: numero intero che identifica il tipo di PDU contenuta nel messaggio (0=GetRequest, 2=Response, 3=SetRequest, 7=Trap);

- **Request ID**: numero usato per identificare in modo univoco una richiesta con la relativa risposta; viene generato dal dispositivo che invia la richiesta e viene copiato nel corrispondente campo presente nella Response PDU;

- **Error Status**: indica una tipologia di errore. Solo l'operazione di risposta eseguita dall'agente può impostare questo campo, tutte le altre operazioni impostano questo campo a zero. Un valore pari a zero presente in una Response indica che non ci sono stati errori;

- **Error index**: quando il campo Error Status contiene un valore diverso da zero questo campo contiene un puntatore all'oggetto che lo ha generato. Nei messaggi di richiesta viene posto a zero;

- **Variable bindings**: è il campo dati vero e proprio all'interno delle PDU SNMPv2. Ogni *variable binding* associa una particolare istanza di un oggetto al sua valore attuale (ad eccezione delle richieste Get per le quali il valore viene ignorato). Consiste in una sequenza di coppie nome-variabile. Il nome identifica l'oggetto da leggere/scrivere nel MIB mentre la variabile contiene il valore da scrivere in caso di SetRequest o il valore letto nel caso di Response.

Tutti i messaggi SNMP hanno la stessa struttura, ad eccezione del messaggio GetBulk che segue una struttura diversa:

- **PDU type**: numero intero che identifica il tipo di PDU contenuta nel messaggio che per una GetBulk Request è 5;

- **Request ID**: numero usato per identificare in modo univoco una richiesta con la relativa risposta; viene generato dal dispositivo che invia la richiesta e viene copiato nel corrispondente campo presente nella Response PDU;
- **Non Repeaters**: specifica il numero di istanze di oggetti nel campo *variable bindings* che non dovrebbero essere recuperate più di una volta dall'inizio della richiesta. Questo campo viene usato quando alcune istanze sono oggetti scalari con una sola variabile;
- **Max Repetitions**: definisce il numero massimo di volte che ogni altra variabile, eccetto quelle specificate dal campo Non Repeaters, dovrebbe essere recuperata;
- **Variable bindings**: consiste in una sequenza di coppie nome-variabile degli oggetti richiesti. Il nome identifica l'oggetto da leggere nel MIB mentre la variabile viene ignorata in quanto si tratta di un comando di lettura.

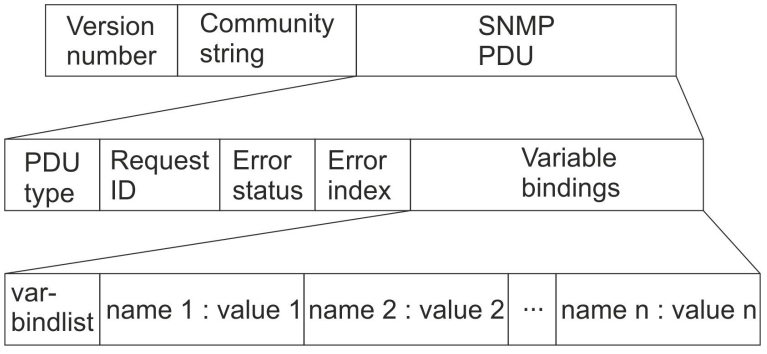

Figura 3.3: *Formato dei messaggi SNMP*

Capitolo 4

Soluzione proposta

Nel 2003, Gupta e Singh [29] hanno posto la propria attenzione sull'importanza della riduzione del consumo energetico in reti cablate. Essi hanno individuato nell'implementazione di stati di sleeping la soluzione ideale al problema, concentrandosi sulle problematiche da affrontare.

Per prima cosa, si sono concentrati sulle modifiche da effettuare sui dispositivi di rete per renderli efficienti dal punto di vista del consumo energetico. Hanno identificato come possibili componenti da mettere in sleeping: memoria, processore centrale, bus, line card processor/ASIC e struttura di switching. Poter effettuare tali operazioni permetterebbe un notevole risparmio energetico. I due autori hanno quindi analizzato la fattibilità di questa tecnica: si sono concentrati sia sulla necessità di modifiche hardware che sulle problematiche da affrontare a livello di protocolli di instradamento.

Il primo problema riguardante le procedure di sleeping consiste nel decidere quando e come utilizzarle. Innanzitutto, è necessario determinare quanto a lungo un particolare componente possa rimanere sleeping. La procedura di risveglio causa un picco nel consumo energetico; i componenti dovrebbero pertanto restare in sleep abbastanza a lungo da compensare l'energia addizionale spesa per risvegliarli. Inoltre, un componente dovrebbe rimanere sleeping per un intervallo più lungo del tempo necessario a risvegliarlo. Il massimo intervallo di sleeping è condizionato dal carico di traffico e dalle caratteristiche del protocollo: per esempio, adottare uno sleeping

non coordinato unitamente al protocollo OSPF (Open Shortest Path First) implica una durata massima del periodo di sleeping non superiore al tempo tra due messaggi di Hello consecutivi: questo permetterebbe di non perdere connettività tra i nodi, a fronte però di un risparmio energetico ridotto.

Il secondo problema riguarda il modo con cui vengono prese le decisioni sullo stato di sleep. Se un router/switch usa un approccio non coordinato può solo monitorare il traffico su tutte le sue interfacce, stimando il tempo di interarrivo tra i pacchetti. Invece, in caso di approcci coordinati, gli intervalli di sleeping sono calcolati attraverso considerazioni basati su stime dei flussi di traffico a livello di AS (Autonomous System).

Infine, l'ultimo problema è rappresentato dalla scelta degli elementi più adatti ad essere spenti. Molti sono i fattori da prendere in considerazione, tra cui: la posizione dei router (backbone, accesso, bordo AS, ecc.), la tipologia di traffico gestito (inter/intra AS) ed il numero di rotte all'interno degli AS.

Gupta e Singh hanno analizzato le conseguenze di tutto questo sui principali protocolli di routing utilizzati nelle reti cablate: OSPF e IBGP. Per quanto riguarda OSPF, porre un'interfaccia di rete in stato di sleep comporterebbe la generazione di pacchetti LSA indicanti un collegamento guasto. Ciò causerebbe un fitto scambio di aggiornamenti LSA seguiti dal ricalcolo dei percorsi da parte di tutti i router. Questo comportamento è chiaramente non necessario e costoso, pertanto l'implementazione attuale di OSPF richiederebbe delle modifiche che permettano di trattare lo sleeping non coordinato diversamente dai guasti. Lo stesso vale per gli altri protocolli.

In caso di sleeping coordinato, sarebbero richiesti significativi cambiamenti ai protocolli esistenti che potrebbero portare ad effetti indesiderati. Durante i periodi di basso carico il protocollo di routing dovrebbe identificare percorsi singoli anziché multipli tra le coppie sorgente/destinazione. Considerando di porre in sleeping solamente le interfacce di un router anzichè l'intero nodo, il problema diventa più flessibile e si traduce nella minimizzazione del numero di interfacce da mantenere attive. In OSPF, ciò comporterebbe i seguenti cambiamenti:

- durante i periodi in cui lo sleeping coordinato sia possibile, bisognerebbe rimpiazzare l'algoritmo SPF del protocollo con uno ad hoc che identifichi il numero minimo di link necessario a mantenere la connettività tra tutti i nodi e a rispettare i vincoli di QoS;

- utilizzare singoli percorsi comporta conseguenze peggiori in caso di guasto di un router o un collegamento;

- i messaggi di Hello dovrebbero essere inviati solamente dalle interfacce attive;

- bisognerebbe sviluppare algoritmi in grado di prevedere condizioni di basso carico della rete ed incorporarli in OSPF: ciò permetterebbero ai router di sapere quando poter passare in stato di sleep;

- ci sarebbe bisogno di un'intelligenza centrale che possa prendere decisioni su accensione e spegnimento dei dispositivi di rete.

Dalla lettura di questo articolo è nato il nostro interesse verso lo sviluppo di una soluzione al problema del risparmio energetico in reti OSPF.

Le prime considerazioni hanno riguardato le possibili modifiche da applicare al protocollo per ovviare ai problemi presentati da Gupta e Singh. Data la complessità di un simile approccio, abbiamo pensato di demandare la gestione dello *sleeping* ad un livello protocollare superiore. Per farlo, abbiamo sfruttato SNMP, il più diffuso protocollo per la gestione dei dispositivi di rete.

L'implementazione di uno stato a basso consumo energetico ha richiesto l'introduzione di un nuovo stato amministrativo delle interfacce di rete dei dispositivi, denominato appunto *sleeping*. Questo è controllato dall'oggetto IfAdminStatus, contenuto all'interno del IF-MIB e configurabile attraverso *snmpset*. Abbiamo così ottenuto una distinzione, quantomeno a livello logico SNMP, tra le interfacce attive e quelle in stato di basso consumo energetico. Le modifiche necessarie a rendere funzionale lo *sleeping* sono illustrate nel capitolo successivo.

Differentemente da Gupta e Singh, abbiamo pensato che ogni dispositivo posto in *sleeping* in una rete OSPF debba informare del suo stato gli altri nodi attraverso messaggi di *Hello*. Questo permetterebbe così di avere periodi di *sleeping* di una durata maggiore di un *Hello Interval*, ottenendo un risparmio energetico superiore. Inoltre, si eviterebbe di tentare di risvegliare nodi che siano guasti o inattivi, e si agirebbe solamente su quelli in stato di *sleep*. Per rendere tale soluzione funzionante, occorre affrontare le seguenti problematiche:

- attualmente, il protocollo di *Hello* permette ai nodi OSPF di inviare messaggi periodici che consentono di sapere solamente se i vicini sono attivi o meno. Un nodo attivo invia un pacchetto di *Hello* ogni *Hello Interval* (10) secondi, mentre un nodo inattivo non ne invia. Di conseguenza, un dispositivo è considerato inattivo dal suo vicino quando quest'ultimo non riceve pacchetti di *Hello* per *Dead Interval* (40) secondi. È quindi utile diffondere in rete l'informazione sullo *sleeping*, ottenendo una discriminazione tra stato a basso consumo e stato di inattività. Una possibile soluzione a questo problema può essere l'utilizzo di due *Hello Interval* differenti: uno denoterà un nodo attivo (2), l'altro permetterà di riconoscere la situazione di *sleeping* (1). Questo richiederà, di conseguenza, la modifica del *Dead Interval*;

- un nodo che riceverà da un vicino un pacchetto di *Hello* con un *Hello Interval* pari a quello previsto per i dispositivi in *sleeping* saprà che il mittente del messaggio è nello stato a basso consumo energetico. In questo modo, l'informazione relativa allo *sleeping* si diffonderebbe senza problemi tra i nodi vicini, come in Figura 4.1. Per divulgarla in tutta la rete si potrebbero utilizzare gli Opaque-LSA [18] configurati come in Figura 4.2. Un nodo inattivo continuerà a non inviare pacchetti di *Hello*;

- allo stato attuale, OSPF non prevede la possibilità di utilizzare due *Hello Interval* differenti all'interno del medesimo AS. Per rendere funzionale tale modifica serve una riconfigurazione del protocollo.

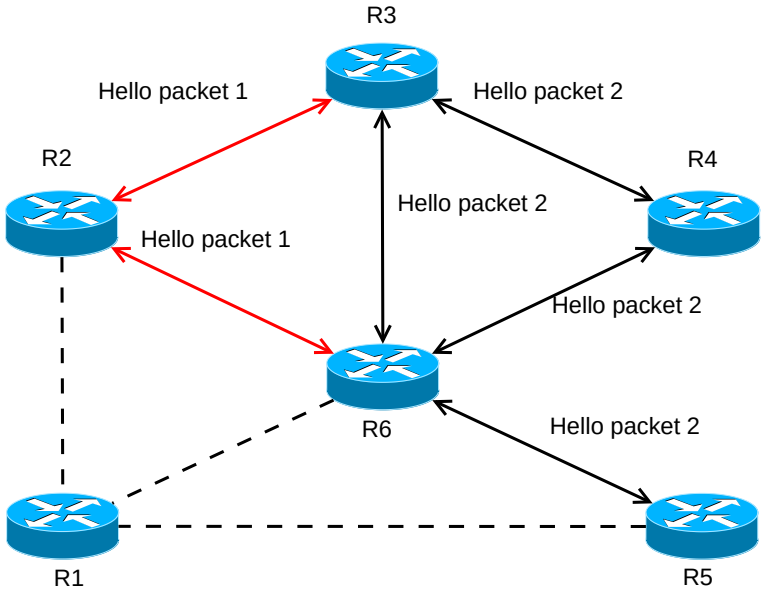

Figura 4.1: *Le frecce indicano uno scambio di pacchetti Hello. Nella particolare situazione si può osservare come R2, essendo in sleeping, invii pacchetti Hello di tipo 1 ai suoi vicini attivi. R1 è inattivo e quindi non scambia questo tipo di pacchetti. Gli altri router sono attivi e per questo motivo inviano pacchetti Hello di tipo 2.*

Ls age	Options	10
Opaque Type	Opaque ID	
Advertising router		
LS Sequence Number		
Ls checksum	Length	
R2 Sleeping		

Figura 4.2: *Opaque-LSA di R6, Figura 4.1. Sfruttiamo l'informazione opaca per comunicare lo sleeping di R2 agli altri vicini.*

Per testare le funzionalità delle nostre modifiche, abbiamo deciso di lavorare con un emulatore, Netkit, in maniera tale da avere una riproduzione del comportamento

della rete quanto più realistico possibile. Abbiamo pensato, inoltre, che un simulatore non ci avrebbe garantito la stessa libertà nell'effettuare le modifiche di cui avevamo bisogno. Congiuntamente a Netkit abbiamo utilizzato Net-SNMP per gestire il protocollo SNMP all'interno dell'emulatore. Netkit ci ha permesso di effettuare i nostri test all'interno di reti OSPF, dandoci la possibilità di verificare l'effettiva applicabilità del nostro approccio e l'impatto dell'implementazione dello stato a basso consumo sul protocollo. Questo ci ha portato ad ulteriori considerazioni:

- lasciando le tempistiche di segnalazione sui valori standard, non è possibile ottenere sleeping di breve durata né avere reazioni rapide in caso di variazioni del carico di rete;

- occorre considerare il tempo fisico necessario allo spegnimento ed alla riaccensione dei nodi.

Tuttavia, assumendo di lavorare in situazione di basso carico della rete, ciò non crea problemi. L'assunzione ha un senso perché si prevede che i dispositivi di rete possano entrare in stato in sleep proprio nel caso in cui la rete sia abbastanza scarica.

Infine, abbiamo pensato a degli scenari applicativi di interesse per testare il funzionamento delle nostre modifiche, illustrati nella sezione seguente. Il funzionamento di Netkit, la configurazione di Net-SNMP e gli altri software utilizzati sono invece ampiamente descritti nel capitolo successivo.

4.1 Scenari di risparmio energetico

Come già detto, con questo lavoro si vuole dimostrare che in una rete OSPF è possibile applicare dei cambiamenti per garantire un minore consumo energetico delle apparecchiature di rete. Abbiamo quindi valutato le diverse possibilità per il raggiungimento di questo obiettivo attraverso diversi approcci. Ogni approccio differisce dall'altro per le interazioni tra gli elementi di rete e per le modalità di adattamento alle condizioni di rete. Le possibilità individuate sono risultate le seguenti:

- scenario centralizzato statico;
- scenario centralizzato dinamico;
- scenario distribuito.

4.1.1 Scenario centralizzato statico

Questo scenario è caratterizzato dalla presenza di un router che occupa un ruolo centrale nella rete. Il router centrale, come rappresentato in Figura 4.3, mantiene una tabella oraria. Attraverso consultazioni con questa tabella, il router centrale

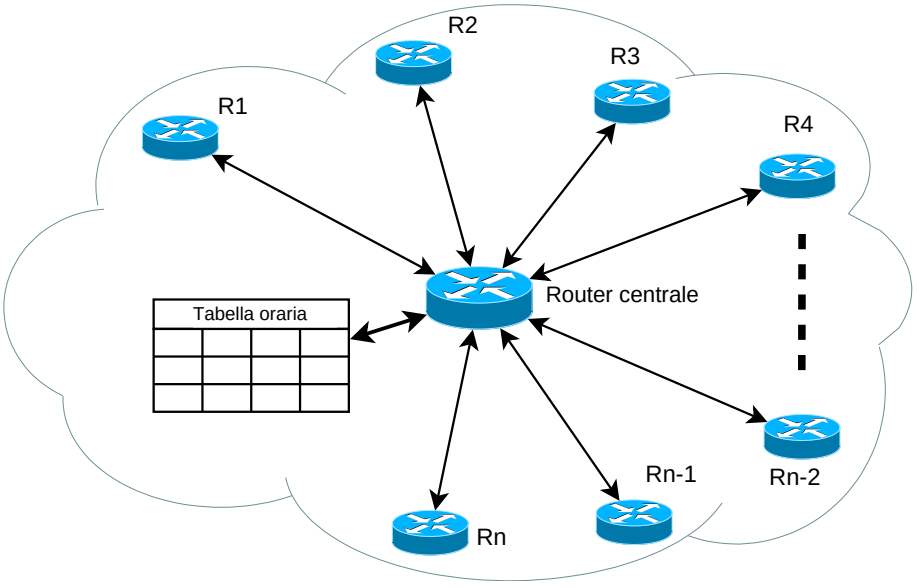

Figura 4.3: *I collegamenti raffigurati rappresentano i possibili percorsi dei messaggi scaturiti dalle operazioni di snmpset*

decide quali interfacce di quali router porre in stato di *sleeping*. Ovviamente la tabella oraria sarà costruita in modo di garantire che in orari prestabiliti (ottenuti attraverso osservazioni delle normali condizioni di carico della rete), in cui la rete risulti particolarmente scarica, si entri in una condizione di risparmio energetico. Il punto di forza di questo scenario è il numero limitato di messaggi di gestione, mentre risulta particolarmente limitante la staticità. Infatti tutti i cambiamenti di stato di router ed interfacce saranno decisi esclusivamente in base alla tabella oraria, non

permettendo di reagire ad anomalie che porterebbero ad aumentare il carico della rete e quindi la necessità di riaccendere dei componenti di rete.

Questo scenario è di poco interesse pratico, è servito principalmente a prendere familiarità con l'ambiente emulato di Netkit e il software di management Net-SNMP.

4.1.2 Scenario centralizzato dinamico

Questo scenario rappresenta un'evoluzione di quello precedente: infatti, come si può notare da Figura 4.4, permangono la concezione di router centrale e tabella oraria. La novità introdotta riguarda la possibilità di reagire a situazioni in cui il carico di

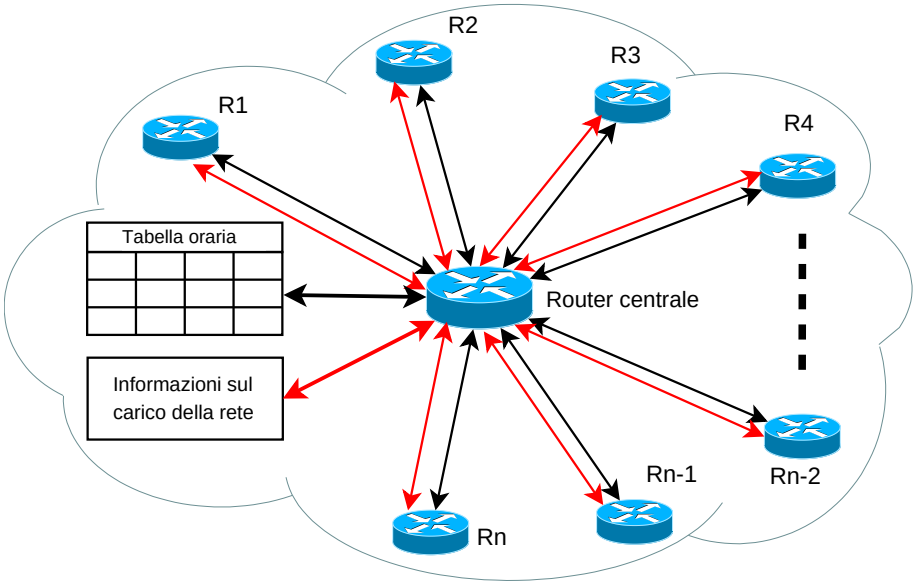

Figura 4.4: *I collegamenti raffigurati in nero rappresentano i possibili percorsi dei messaggi scaturiti dalle operazioni di snmpset previste dalla tabella oraria. Le interazioni raffigurate in rosso riguardano invece la messaggistica necessaria per la reazione a situazioni di alto carico.*

rete tende ad aumentare. In questo caso la tabella oraria, che aveva previsto una situazione di basso consumo in quella determinata fascia oraria, si dimostra inattendibile a causa di un aumento di traffico non previsto o di una situazione di guasto. È quindi utile disporre di uno scenario che permetta ai nodi in stato di *sleeping* di risvegliarsi periodicamente per controllare la situazione della rete attraverso il router centrale. Ciò può essere reso possibile attraverso l'uso di messaggistica SNMP come *snmpset* e *snmptrap*.

Rispetto allo scenario precedente abbiamo introdotto la possibilità di reagire alle diverse condizioni della rete. Per gestire ciò è però necessario che in rete siano presenti molti più messaggi di gestione.

4.1.3 Scenario distribuito

L'ultimo scenario individuato è quello distribuito. Come visibile in Figura 4.5 non è presente una figura centrale che permetta la gestione delle situazioni di basso consumo nella rete. Infatti ogni router esegue lo stesso algoritmo, convergendo alla stessa soluzione. In questo modo, dopo l'esecuzione dell'algoritmo, ogni elemento

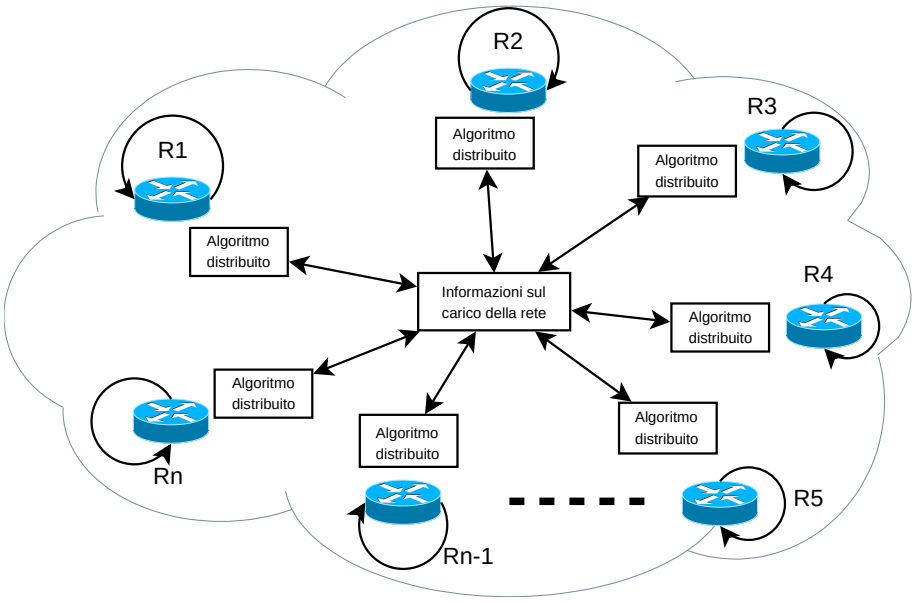

Figura 4.5: *Gli autoanelli rappresentano le decisioni prese indipendentemente da ogni nodo, in base all'algoritmo distribuito, riguardanti il proprio stato di sleeping.*

di rete sarà a conoscenza della situazione della rete, decidendo indipendentemente se mettersi o meno in stato di *sleeping*. Anche in questo caso dovrà essere previsto un risveglio periodico. Ciò permetterà di adattarsi alle nuove domande di traffico, informazione che sarà resa disponibile a tutti i router della rete in modo centralizzato.

Il lavoro di tesi qui presentato si focalizza sul secondo scenario descritto. Abbiamo implementato nell'emulatore Netkit uno scenario di rete in cui è presente una intelligenza centrale in grado di prendere decisioni relative all'accensione e lo spe-

gnimento di nodi o link. Per realizzare una piattaforma funzionante abbiamo ideato alcuni script di gestione dedicati da eseguire all'interno delle macchine di rete, con la possibilità di porre in *sleeping* i link di rete e di risvegliarli all'occorrenza.

Capitolo 5

Ambiente di lavoro

La possibilità di testare una configurazione di una rete prima di implementarla è particolarmente interessante ed utile sia per gli amministratori che per gli informatici. I primi possono trarre vantaggio da una fase di test per controllare se una particolare configurazione funziona, prima ancora di effettuare il dispiegamento della rete stessa. I secondi possono sfruttare i risultati dei test per verificare la validità di modelli teorici attraverso esperimenti pratici. Idealmente, i test dovrebbero avere luogo in condizioni molto simili a quelle reali. Tuttavia, spesso ciò significa iniettare traffico generato artificialmente e potenzialmente dannoso in una vera rete. Una valida alternativa ai test su rete reale consiste nell'implementazione di una configurazione di rete di interesse all'interno di un ambiente software sicuro ed isolato, che riproduca il più verosimilmente possibile la reale configurazione che si intende realizzare. Ciò si può ottenere in due modi:

- **ambienti simulati**: permettono all'utente di prevedere il risultato di un insieme di dispositivi in una rete complessa, utilizzando un modello interno che è specifico del simulatore. Con la rete in ingresso ed il risultato (lo stato della rete, ad esempio) in uscita, i simulatori non riproducono necessariamente la stessa sequenza di eventi che avrebbe luogo in un sistema reale, ma applicano un insieme di routine di trasformazione che portano la rete ad uno stato finale che è quanto più possibile vicino a quello a cui arriverebbe il sistema reale. La rete simulata risulta scalabile e gestibile anche se di dimensioni consistenti. Lo

svantaggio è che i dispositivi simulati possono avere funzionalità limitate ed il loro comportamento può non essere vicino a quello dei dispositivi reali. Un esempio di simulatore di rete è NS-2 [11].

- **ambienti emulati**: l'obiettivo è riprodurre le funzionalità ed il comportamento dei dispositivi del mondo reale. Per questo motivo, spesso consistono di un software da utilizzare su dispositivi reali. Differentemente dai simulatori, in un emulatore la rete che è in fase di test è sottoposta agli stessi scambi di pacchetti ed agli stessi cambi di stato che avverrebbero nel mondo reale.

Il vantaggio dell'approccio emulativo emerge quando l'emulatore stesso è un software modulare, il che permette una maggiore flessibilità nell'effettuare i test di rete. Poiché si fa uso di software di instradamento reale, ogni aspetto della rete può essere influenzato e monitorato come nella realtà. Se da un lato ciò comporta grande accuratezza, le risorse computazionali necessarie per far funzionare un dispositivo emulato sono tipicamente maggiori di quelle disponibili sul dispositivo stesso. Quindi, le prestazioni di un dispositivo emulato saranno, in generale, inferiori a quello reale, e ciò pone spesso limiti di scalabilità sulle dimensioni di una rete emulata.

Per il nostro lavoro abbiamo utilizzato l'emulatore Netkit [13], basato su software open source e creato da un gruppo di persone dell'Università Roma Tre con la collaborazione del Linux User Group Roma Tre per realizzare esperimenti di rete su personal computer. L'emulatore è illustrato nel dettaglio in [44], sul quale si basa la sezione successiva.

Unitamente a Netkit, abbiamo utilizzato Net-SNMP per l'implementazione nello scenario emulato del protocollo SNMP, Quagga per la gestione dell'instradamento OSPF, Iperf per simulare le diverse situazioni di carico della rete, ed infine vnStat per avere delle misure sull'utilizzo e sul carico di rete. Tutti i software sono illustrati nella sezione successiva.

5.1 Ambiente di emulazione ed architettura di Netkit

Un *emulatore* è un ambiente software o hardware che mira a riprodurre le funzionalità ed il comportamento di dispositivi o sistemi reali, solitamente rendendo possibile l'utilizzo di prodotti software inalterati all'interno di un sistema che non sarebbe progettato per supportarli.

Una *macchina virtuale* è un software che crea un livello di astrazione tra la piattaforma hardware/software ed altro software, ad esempio un sistema operativo. Una macchina virtuale può anche implementare dispositivi virtuali, come dischi ed interfacce di rete, differenti da quelli disponibili per la piattaforma sulla quale viene eseguito l'emulatore.

Netkit consiste di due componenti principali: un insieme di script e strumenti che facilitano la messa a punto di complessi scenari di rete con un buon numero di dispositivi, ed un insieme di scenari preconfigurati pronti all'uso che possono servire ad analizzare tipici casi di studio. Netkit ha un utilizzo ed un'installazione piuttosto semplici e non richiede i privilegi di amministratore per nessuna delle due operazioni.

Col passare del tempo, i bug sono stati corretti e nuove funzionalità sono state introdotte in Netkit. È stata utilizzata la versione 2.7, con versione del filesystem 5.1 e kernel 2.8 (basato sullo User-Mode Linux kernel 2.6.26.5). L'approccio di emulazione adottato in Netkit è semplice. Fondamentalmente, ogni dispositivo di rete viene implementato in Netkit come una macchina virtuale. Ognuna di esse possiede un insieme di risorse virtuali, mappate in porzioni delle corrispondenti risorse sull'host. La Figura 5.1 mostra come ha luogo la mappatura. Le macchine virtuali sono equipaggiate con un disco, la cui immagine grezza è un file sulla macchina host; ognuna ha la sua regione di memoria, la cui dimensione può essere stabilita all'avvio. Inoltre, le macchine possono essere configurate con un numero arbitrario di interfacce di rete che sono connesse ad un hub virtuale. Il rettangolo tratteggiato in Figura 5.1 racchiude le risorse virtualizzate, mentre tutto ciò che ne è al di fuori

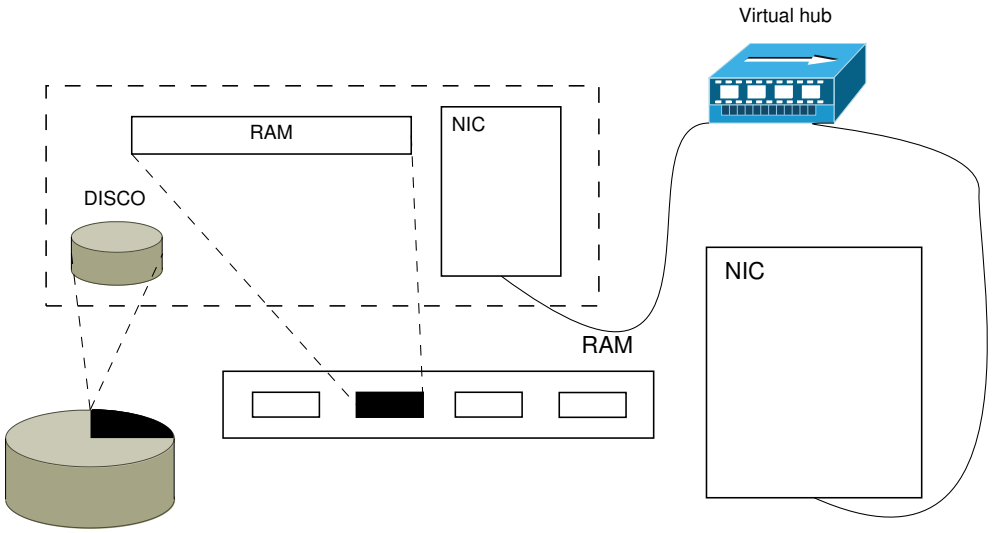

Figura 5.1: *Le entità racchiuse tra linee tratteggiate sono virtualizzate, mentre tutte le altre corrispondono a dispositivi o processi sulla macchina host.*

è un dispositivo o un processo sulla macchina host. Si può osservare che l'hub virtuale risiede nell'host reale: infatti, è un processo speciale che replica i pacchetti su tutte le interfacce connesse. Se richiesto, l'hub virtuale può essere connesso ad una interfaccia di rete sulla macchina host, in modo tale che la macchina virtuale possa raggiungere una rete esterna, come Internet.

Le macchine virtuali di Netkit possono essere collegate tramite hub virtuali. In pratica, un hub lavora come una sorta di cavo che collega più *ospiti*. Un *ospite* deve sempre essere connesso ad un hub virtuale e non può essere connesso direttamente ad un altro *ospite*. La Figura 5.2 mostra come appare una tipica rete di Netkit. VM1, VM2 e VM3 sono macchine virtuali e formano una semplice topologia costituita da due domini di collisione: uno include VM1 e VM2 e l'altro include VM2 e VM3. In questa topologia, VM2 può operare come uno switch virtuale che inoltra traffico solo sulle porte connesse al segmento LAN contenente l'host di destinazione. A questo proposito, VM2 è stato equipaggiato con due interfacce di rete.

L'esempio in Figura 5.2 introduce ad un altro principio dell'approccio di Netkit: le funzionalità di un dispositivo emulato dipendono dal software installato nella macchina virtuale che lo implementa. Per esempio, VM2 può essere trasformata in

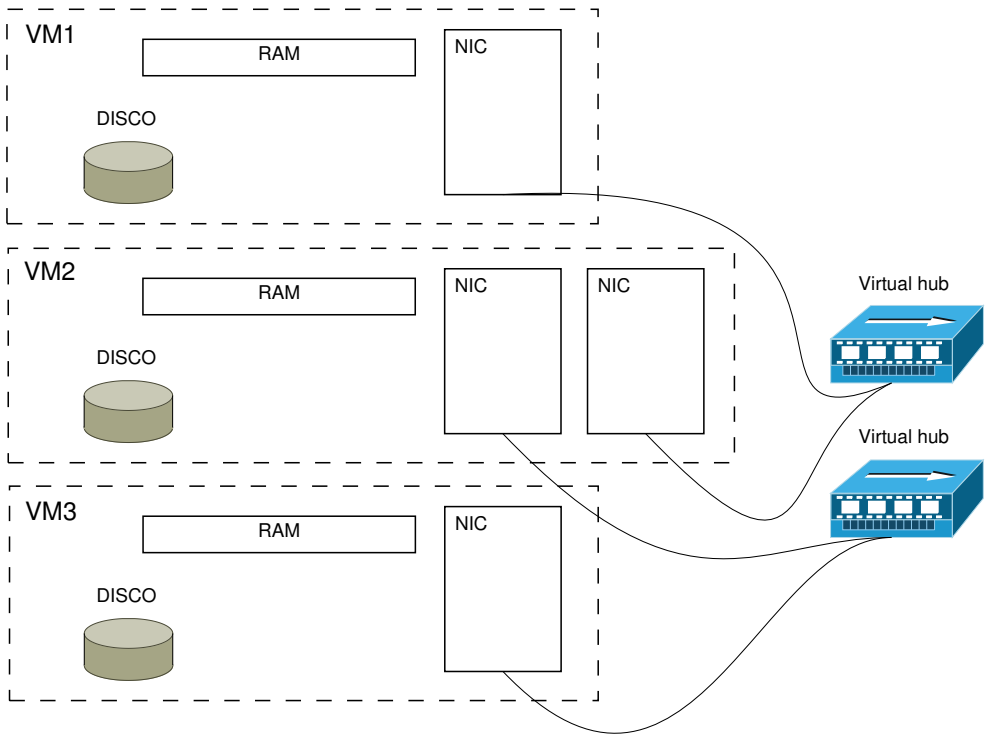

Figura 5.2: *Esempio di rete emulata con Netkit.*

uno switch eseguendo il software standard di amministrazione di un bridge ethernet o in un router settando propriamente la sua tabella di instradamento IP.

Le macchine virtuali di Netkit sono basate sul User-Mode Linux kernel [15], descritto più nel dettaglio nel paragrafo seguente. Eseguire una macchina virtuale significa avviare un'istanza UML, che spesso richiede l'utilizzo di complesse istruzioni da riga di comando. Per questo motivo, Netkit fornisce una semplice configurazione e gestione delle macchine virtuali attraverso un'interfaccia intuitiva costituita da diversi script.

La Figura 5.3 descrive sinteticamente l'architettura di Netkit, costituita dai blocchi all'interno dei rettangoli tratteggiati. Ogni blocco rappresenta un pezzo di software che viene eseguito su quelli negli strati sottostanti ed è controllato dagli strumenti alla sua sinistra. Le macchine virtuali sono istanze UML che girano direttamente sul kernel dell'*host* e vengono gestite da un insieme di comandi i cui nomi hanno il prefisso *l* (ltools) e *v* (vtools). Le macchine virtuali possono eseguire

software di routing. Gli hub virtuali sono implementati come processi avviati nel kernel dell'*host*.

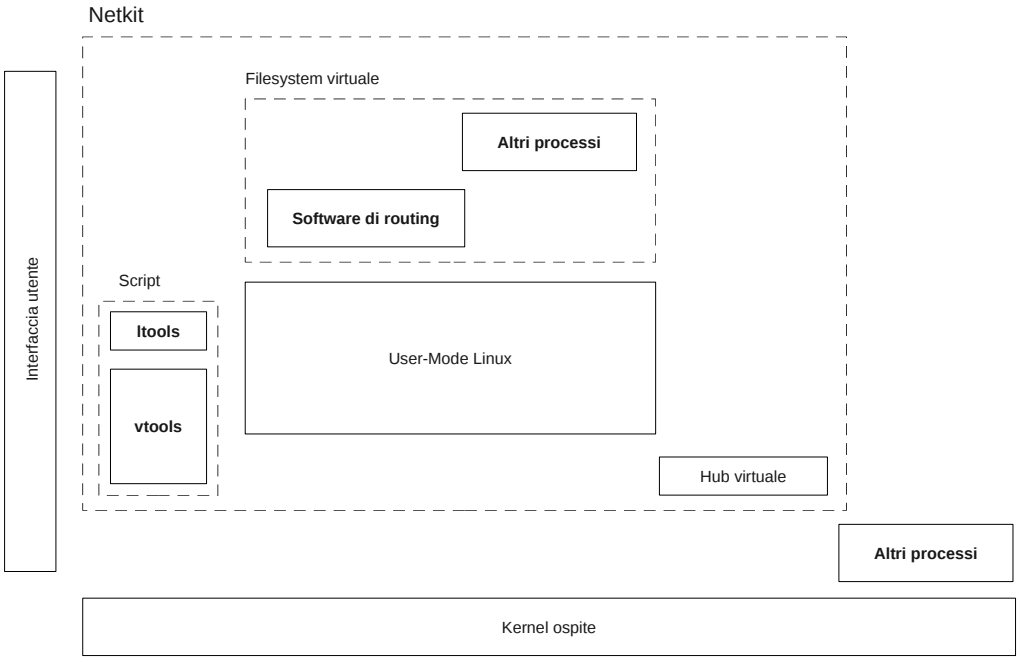

Figura 5.3: *Ogni software si appoggia su quelli di livello inferiore ed è controllato dagli strumenti alla sua sinistra.*

I blocchi con testo in grassetto sono i soli componenti presentati all'utente finale, che non ha la necessità di interagire direttamente con i kernel UML o gli hub virtuali. I blocchi ltools e vtools sono accessibili all'utente: i primi forniscono un'interfaccia di livello più alto alle macchine virtuali e sfruttano i vtools per implementare le loro funzionalità.

Per via della sua architettura, è facile trarre vantaggio dall'uso di Netkit in molti contesti. L'applicazione più comune è di tipo didattico: poiché dà agli studenti la sensazione di ciò che sta realmente accadendo all'interno della rete, Netkit è stato utilizzato con successo per insegnare i protocolli in uso nelle reti in corsi di livello universitario. Inoltre, Netkit fornisce un insieme di prove di rete che implementano

casi di studio che spaziano dai protocolli di instradamento, come TCP, RIP, BGP, OSPF, ai servizi di livello applicativo, come DNS ed e-mail [14]. Inoltre, la possibilità di effettuare esperimenti in un ambiente sicuro, combinata con la semplicità del mantenimento di una riproduzione uno-a-uno di una rete reale, torna utile per testare una configurazione, prima di implementarla. Per esempio, Netkit è stato utilizzato per aiutare a determinare i pesi OSPF assegnati all'interno di porzioni della GARR Italian Academic Research Network [2].

Rispetto ad altri emulatori di rete, Netkit ha il vantaggio di essere leggero e facile da installare ed utilizzare. Ad esempio, si può lanciare una complessa rete da 200 macchine virtuali in 30 minuti su una tipica workstation (Pentium IV 3.2 GHz, 2 MB cache, 2GB RAM). Netkit non ha dipendenze da altri software; inoltre, qualsiasi scenario emulato può essere facilmente redistribuito in forma preconfigurata, pronta all'uso e senza la necessità di trasporto dell'immagine del filesystem.

Netkit lavora solamente su Linux ed emula macchine virtuali Linux. Attualmente, Netkit fornisce un'implementazione del livello 2 dello stack protocollare di rete che non supporta l'emulazione del livello fisico (latenza, perdita di pacchetti, riordino, ecc.) o il comportamento delle stazioni mobili senza fili. Essendo un emulatore, Netkit non è fatto per riprodurre prestazioni in tempo reale dei protocolli e servizi di rete.

5.2 User-Mode Linux kernel

In un ambiente di emulazione, le macchine virtuali hanno caratteristiche molto simili a quelle reali e possiedono un loro kernel: in Netkit è stato utilizzato lo User-Mode Linux. Esso è ampiamente sfruttato, ad esempio, dagli sviluppatori hardware per testare nuovi dispositivi nel software. È spesso impiegato anche per eseguire server virtuali [5], [4]. Le parti fondamentali di UML sono illustrate dal suo progettista Jeff Dike in alcune pubblicazioni [24], [21].

User-Mode Linux [21], [36], [24], [23], [22], [15] deriva dal Linux kernel standard [10] ed è progettato per essere eseguito come un processo dell'utente. UML possiede

i suoi sottosistemi del kernel, inclusi uno scheduler, un gestore della memoria ed un filesystem. Un'istanza UML fornisce un ambiente virtualizzato nel quale ogni cosa (processi, memoria, filesystem, ecc.) è controllata da sé stessa anziché dal kernel dell'*host*. In pratica, UML appare come un processo dell'utente sulla macchina *host*, mentre agisce come un kernel per i suoi processi.

I settaggi delle macchine virtuali possono essere passati ad UML attraverso un'interfaccia a riga di comando. Pur trattandosi di un metodo efficace per specificare i parametri di configurazione, spesso gli utenti interessati solamente all'emulazione delle reti non vogliono avere a che fare con complessi comandi di invocazione del kernel. Per questo motivo, Netkit fornisce una interfaccia utente di livello più alto ad UML. La virtualizzazione ha luogo all'interno del kernel dell'*host* anziché a livello hardware: in altre parole, UML fornisce un hardware simulato costruito sulla base di servizi dati dal kernel dell'*host*. Il codice utente gira senza emulazione, mentre i processi in modalità kernel vedono un ambiente particolare che limita l'accesso alle risorse dell'*host*. Ciò rende l'emulazione più veloce, ed è la ragione per cui Netkit è considerato leggero. Lo svantaggio di questo approccio è che si possono emulare solamente macchine Linux.

In Netkit si possono implementare configurazioni complesse con molte macchine virtuali. Ogni macchina virtuale scrive sul suo filesystem, il che potenzialmente implica l'utilizzo di molti file di supporto, di dimensioni non trascurabili (centinaia di megabyte). Comunque, è presente la possibilità di condividere un filesystem tra macchine virtuali diverse: ciò viene ottenuto tramite una tecnica conosciuta come Copy-On-Write (COW). Quindi, una tipica configurazione di un complesso scenario emulato consiste di un solo grande file di supporto che contiene il modello del filesystem e diversi piccoli file COW (tipicamente meno di 10 MB) che immagazzinano i cambiamenti al filesystem. Questi ultimi possono essere ricostruiti, per una macchina virtuale, basandosi sia sul suo file COW che sul file di supporto. Ogni file COW può infatti essere utilizzato insieme col file di supporto dal quale è stato creato. Comunque le due parti possono anche essere unite attraverso uml_moo [12], [32] per

ottenere tutte le informazioni del filesystem per una data macchina virtuale.

L'avvio di una macchina virtuale comporta la creazione sul disco della macchina *host* di un file di estensione *.disk*. Sebbene apparentemente le dimensioni di questo file si aggirino su qualche GB, in realtà esse dipendono dal numero di modifiche effettuate al filesystem, il che significa che una macchina appena avviata occupa appena qualche centinaio di KB. Una volta che la macchina virtuale è stata avviata, è possibile comunicarci attraverso un terminale: può essere spenta, riavviata, se ne possono configurare i dispositivi emulati e si può mettere in pausa e riprenderne l'esecuzione in un secondo momento. Non è prevista la possibilità di utilizzare software con interfaccia grafica.

A partire dal 2002, Paolo Giarrusso [31] mantiene un insieme di patch chiamato SKAS che può essere opzionalmente applicato al kernel dell'host per cambiare il comportamento di UML. Queste portano benefici sia in termini di sicurezza che di prestazioni. Per quanto riguarda la sicurezza, SKAS fa sì che UML venga eseguito in uno spazio di indirizzamento diverso da quello dei processi che controlla. Ciò migliora anche le prestazioni, rendendo l'avvio di complessi scenari da emulare assai più veloce. Comunque, Netkit mantiene una buona scalabilità anche senza l'utilizzo di SKAS.

5.3 Supporto di rete in Netkit

La comunicazione tra macchine virtuali diverse è resa possibile in Netkit attraverso l'emulazione delle funzionalità di rete. La Figura 5.4 descrive graficamente come questa abbia luogo. UML permette di configurare le macchine virtuali con un numero arbitrario di interfacce di rete. Utilizzando gli appropriati comandi, queste interfacce possono essere collegate ad un processo uml_switch [32], [12] eseguito sull'*host*, che simula il comportamento di uno switch o hub. In questo modo, diverse macchine virtuali collegate allo stesso switch possono scambiare dati tra loro. Le interfacce di rete virtuali di UML possono essere collegate ad un socket UNIX. Un uml_switch può essere collegato allo stesso socket ed inoltrare pacchetti tra le macchine virtuali

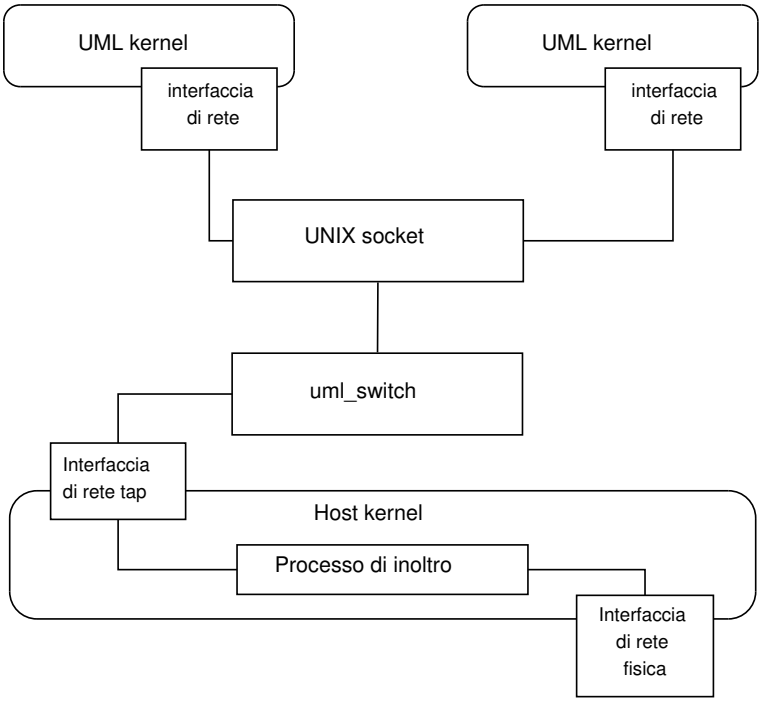

Figura 5.4: *Questa figura mostra come sono collegate le macchine virtuali di Netkit, con anche una connessione verso una rete esterna tramite tap*

che vi sono connesse. Gli script di Netkit si prendono cura di avviare in automatico i processi uml_switch in accordo con le necessità dell'utente. Dal punto di vista

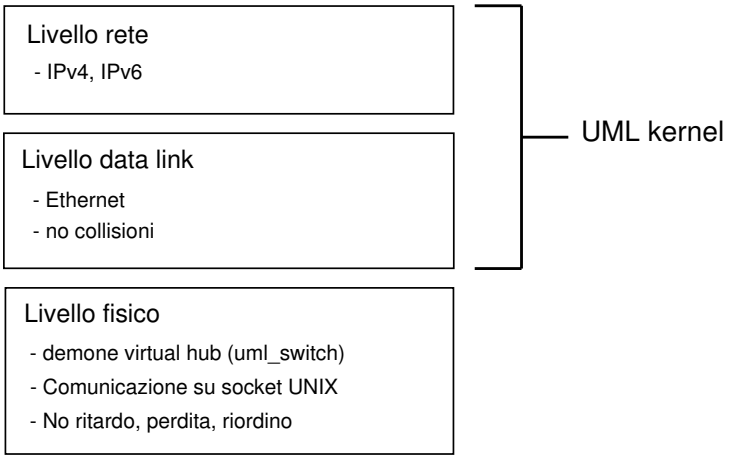

Figura 5.5: *Stack protocollare di rete emulata in Netkit.*

dello stack di rete, Netkit fornisce l'implementazione dei livelli ISO-OSI illustrati in Figura 5.5.

- Il livello fisico è implementato da un insieme di processi uml_switch eseguito sull'*host*. Questi sono configurati per comportarsi come hub e i pacchetti vengono inoltrati alle interfacce di altre macchine virtuali utilizzando i socket UNIX. Al momento non è invece fornito il supporto per simulare ritardi, perdite di pacchetti e riordino.

- Il livello data link supporta il protocollo Ethernet, ma le collisioni non possono avvenire perché l'uml_switch le evita. Se non diversamente specificato, alle interfacce di rete emulate sono assegnati degli indirizzi MAC generati automaticamente.

- Il livello di rete supporta sia IPv4 che IPv6.

- Quello che accade nei livelli superiori dipende dallo specifico software eseguito all'interno delle macchine virtuali.

I livelli dal data link al trasporto sono implementati, almeno in parte, nel UML kernel. Quindi, modificare il kernel può rendere disponibili nuove funzionalità.

La configurazione delle interfacce di rete in Netkit è molto semplice, grazie all'esistenza di script configurati appositamente. Netkit presenta gli uml_switch come domini di collisione virtuali. Ogni interfaccia di rete virtuale deve essere collegata ad un dominio di collisione che è identificato da un nome arbitrario. Di conseguenza, connettere macchine virtuali tra loro equivale a collegare le loro interfacce al medesimo dominio di collisione. Gli indirizzi IP per le interfacce possono quindi essere configurati nella maniera canonica, utilizzando il comando *ifconfig* all'interno delle macchine virtuali.

Di particolare interesse è la possibilità di configurare una macchina virtuale in maniera tale da raggiungere reti esterne, come Internet. Per farlo, si utilizza un dispositivo detto TAP/TUN. Esso è contenuto nel kernel di Linux e configura una particolare interfaccia di rete collegata ad un'applicazione utente invece che ad un mezzo fisico. I pacchetti inviati all'interfaccia TAP/TUN vengono scritti dall'applicazione sottostante, mentre l'informazione generata dall'applicazione appare come

se fosse ricevuta dall'interfaccia. La sola differenza tra TUN e TAP è che la prima si aspetta che sia l'applicazione a gestire i pacchetti IP, mentre la seconda permette di trasferire direttamente trame Ethernet.

Come mostrato in Figura 5.4, la configurazione di una connessione esterna implica la configurazione di un dispositivo TAP sulla macchina host e l'utilizzo di uml_switch come applicazione che invii e riceva i dati da essa. Poiché uml_switch può essere connesso a macchine virtuali attraverso socket UNIX, ciò permette di connettere un segmento di rete virtuale con una reale. Per evitare che gli indirizzi IP utilizzati per gli esperimenti vengano presi dalla rete reale, Netkit permette il routing anziché il bridging, ed utilizza il mascheramento per nascondere gli indirizzi assegnati alle interfacce di rete emulata. Il mascheramento è una variante del NAT basato su porta nel quale i pacchetti uscenti sono modificati per apparire come se fossero originati dall'interfaccia attraverso la quale sono stati inviati. Il mascheramento può essere abilitato in Linux traendo vantaggio dalla piattaforma netfilter [7] all'interno del kernel.

La configurazione di una TAP prevede l'utilizzo di due indirizzi IP che vengono assegnati automaticamente all'interfaccia TAP sulla macchina *host* e all'interfaccia emulata all'interno della macchina virtuale. Questi indirizzi sono anche utilizzati nella tabella di instradamento al suo interno. Occorre fornire entrambi, in maniera tale che la macchina virtuale, una volta avviata, sia già in grado di raggiungere l'esterno. La configurazione di interfacce TAP richiede i privilegi di amministratore, il che comporta la richiesta, da parte di Netkit, della password di root.

5.4 Net-SNMP

5.4.1 Cenni storici

Le radici del progetto Net-SNMP [6] risalgono al 1992 presso la Carnegie-Mellon University (CMU). Il Network Group alla CMU, guidato da Steve Waldbusser, sviluppò un'implementazione del allora nascente protocollo di gestione SNMP. Questa suite

di applicativi includeva una libreria, una selezione di semplici comandi di gestione e un *agente* che includeva la maggior parte delle informazioni di gestione definite nel RFC 1213 [27].

Wes Harddaker, facente parte del gruppo sopracitato e amministratore di sistema della University of California (UCD), estese il progetto per dare accesso a più informazioni e notificare certe situazioni di errore. Questo portò nel 1995 a rendere il codice disponibile al pubblico sotto il nome di UCD-SNMP. A questo progetto contribuì anche un altro amministratore di sistema della University of Liverpool, Dave Shield. Il suo interesse era dovuto alla ricerca di un sistema in grado di monitorare il sistema di stampa locale, il che lo spinse a creare diverse migliorie per il progetto UCD-SNMP.

Contemporaneamente anche Niels Baggesen, un amministratore di sistema danese, contribuì al progetto migliorando il suo comportamento con sistemi Solaris e dando maggiore stabilità e solidità al codice. Con la collaborazione di Joe Marzot che sviluppò un'interfaccia Perl per la libreria SNMP e di Mike Slifcak che lavorò sulle API, UCD-SNMP diventò a tutti gli effetti un progetto Open Source collaborativo. UCD-SNMP cominciò ad essere usato e migliorato da un numero sempre maggiore di persone.

Nel 1998, la IETF incaricò Wes Harddaker dello sviluppo di un'implementazione che avesse le specifiche di SNMPv3. Il risultato di questo lavoro portò al rilascio della nuova versione UCD-SNMP v4.0 nell'agosto del 1999.

Nel 2000 il nome del progetto cambiò in Net-SNMP e venne trasferito sul portale Sourceforge [9], permettendo di suddividere i compiti amministrativi. Grazie ad un grande numero di collaboratori dovuto anche al supporto per Microsoft Windows, il codice vide una ristrutturazione della piattaforma che venne dotata di un modulo MIB più flessibile. L'architettura del codice diventava sempre più modulare facilitando l'accorpamento di nuove funzionalità. Il frutto di questo lavoro si ebbe con la nuova versione di Net-SNMP v5 rilasciata nel 2002.

Contemporaneamente il progetto venne reso disponibile per differenti sistemi

embedded come quelli basati su μCLinux, nonché adottato come parte di default di molte distribuzioni Linux e sistemi basati su BSD. Successivamente vennero sviluppati molti pacchetti basati sulla piattaforma Net-SNMP, sancendo quindi il successo del progetto.

5.4.2 Funzionalità

Per ottenere una piattaforma che permetta il controllo e la gestione dei dispositivi di rete abbiamo riscontrato la necessità dell'utilizzo del protocollo SNMP. La suite di applicativi Net-SNMP mette a disposizione le principali funzionalità del protocollo in tutte le sue versioni, supportando sia IPv4 che IPv6. Tra le principali ricordiamo quelle più frequentemente utilizzate che permettono di:

- ottenere informazioni dai dispositivi remoti come *snmpget*, *snmpgetnext* e *snmpwalk*, il quale permette di ottenere informazioni multiple sui dispositivi attraverso *snmpgetnext* consecutive;

- manipolare i valori presenti nei MIB dei dispositivi come *snmpset*;

- gestire l'invio e la ricezione di *snmptrap* e *snmpinform* che permettono la reazione a determinate notifiche, reso disponibile dal demone *snmptrapd*;

- includere il supporto a molti moduli MIB standard attraverso il demone *snmpd* che può essere esteso usando moduli caricati dinamicamente come *SNMP multiplexing* (SMUX).

Net-SNMP è integrato in Netkit: tra le macchine virtuali è già disponibile la comunicazione SNMP, essendo già presenti al loro interno i file di configurazione del demone *snmpd*. In una rete OSPF come quella considerato, questa comunicazione è resa possibile grazie all'interazione di Net-SNMP e Quagga, descritta in modo più dettagliato in seguito.

5.4.3 Modifiche al codice sorgente

I moduli MIB precaricati attraverso l'installazione di Net-SNMP sono standard. Il modulo di interesse al fine della nostra trattazione è *IF-MIB*, che per quanto riguarda gli stati amministrativi delle interfacce prevede tre possibili situazioni come già mostrato nel Codice 3.1. Per poter gestire degli stati di basso consumo energetico, è necessario intervenire su questo modulo. Bisognerà introdurre un nuovo stato che rappresenti questa situazione, lo stato di *sleeping*. L'introduzione di questo cambiamento rende necessario intervenire su diversi codici sorgente di Net-SNMP, affinché la modifica sia operativa e utilizzabile dalle macchine emulate. I codici sorgente di seguito descritti fanno riferimento alla versione 5.4.1 di Net-SNMP.

In primo luogo il lavoro di variazione ha riguardato il modulo MIB sopracitato. Abbiamo dovuto individuare il file testuale contenente le informazioni sulle interfacce dei dispositivi, capire quale fosse l'informazione la cui modifica portasse al risultato desiderato e inserire il nuovo stato. Gli oggetti del MIB che potrebbero prestarsi a questi cambiamenti sono *ifAdminStatus* e *ifOperStatus*. La scelta è ricaduta sul primo. Si vuole infatti poter intervenire sullo stato attuale di un'interfaccia e *ifOperStatus*, essendo gestita esclusivamente tramite interazioni con *ifAdminStatus*, non rende disponibile la funzione *snmpset* da parte della stazione di gestione. Il cambiamento necessario è stato quello mostrato nel Codice 5.1.

Codice 5.1: *Oggetto ifAdminStatus contenuto in IF-MIB con l'aggiunta del nuovo stato*

```
ifAdminStatus OBJECT-TYPE
SYNTAX  INTEGER {
            up(1),       -- ready to pass packets
            down(2),
            testing(3), -- in some test mode
            sleeping(4) -- in low consumption mode
        }
```

Ovviamente questa sola modifica non permette di effettuare la *snmpset* al nuovo valore sull'oggetto desiderato. La causa di questo comportamento è insita nel codice sorgente di Net-SNMP, scritto in linguaggio *C++*. Sono infatti presenti dei controlli

che non permettono di effettuare delle *snmpset* a valori non riconosciuti dal codice. Per questo motivo è stato necessario modificare i seguenti file:

- ifTable_constants.h: il percorso di questo file all'interno della cartella di installazione di Net-SNMP è: `/agent/mibgroup/if-mib/ifTable`. Si tratta di un file *header* in cui sono definite variabili e costanti utilizzate dai file con estensione *.c*. Abbiamo definito qui la costante relativa allo stato di *sleeping* dell'oggetto *ifAdminStatus*, assegnandogli il valore intero *4*;

- ifTable_interface.c: dislocato nella stessa cartella del file precedente. Una sezione di questo codice ha il compito di verificare che il valore passato dalla *snmpset* sia ammissibile per l'oggetto obiettivo. La modifica ha permesso al nuovo stato di passare i controlli di ammissibilità;

- interface.c: il percorso di questo file all'interno della cartella di Net-SNMP è: `/agent/mibgroup/if-mib/data_access`. La funzione della sezione di codice modificata, come sopra, ha funzioni di ammissibilità di valori assegnati agli oggetti, con la peculiarità di fare i controlli in fase di precompilazione.

Con l'applicazione di queste modifiche abbiamo reso possibile l'operazione di *snmpset* sull'oggetto *ifAdminStatus* al nuovo stato di *sleeping*.

Verificando le funzionalità di Net-SNMP, abbiamo però notato un'instabilità legata a questo stato ed al legame tra stato amministrativo e stato operativo delle interfacce. Come già detto, lo stato operativo di un'interfaccia non è gestibile da eventi esterni come le *snmpset* e quindi la sua variazione dipende dallo stato amministrativo e dalle condizioni dei dispositivi di rete. Lo stato operativo dovrà essere *down* se quello amministrativo è *down* ma anche nel caso in cui ci siano situazioni di guasto. Nel caso in cui lo stato amministrativo sia invece posto in *up*, allora lo stato operativo dovrà essere *up* solo nel caso in cui l'interfaccia sia realmente pronta per trasmettere e ricevere pacchetti. Il problema riscontrato è dovuto proprio a queste definizioni. Ponendo infatti lo stato amministrativo in *sleeping* tramite *snmpset*,

Net-SNMP non riconosce la relazione di questo valore con lo stato operativo e lo pone quindi *down*. Al successivo controllo, il codice si accorge dell'anomalia, e per riportare la situazione alla normalità modifica lo stato amministrativo per renderlo compatibile con quello operativo. Lo stato amministrativo risulta quindi *down* benché sia stato posto in *sleeping*. L'idea per porre rimedio a questa situazione è di rendere gli stati amministrativi *sleeping* e *down* analoghi per quanto riguarda le interazioni con lo stato operativo. Questa assunzione non crea problemi nello scenario emulato perché quello che interessa è il valore logico. In una situazione reale occorrerà gestire con più attenzione questo punto in quanto dovrà essere pensato anche uno stato operativo a basso consumo che lo mantenga tale anche a livello fisico. Le ulteriori modifiche al codice sorgente per ottenere quanto spiegato sopra hanno riguardato:

- ifTable_constants.h: file già citato per le modifiche precedenti. Per dare stabilità al codice abbiamo utilizzato un vettore (`sleepID[ ]`) per mantenere memoria della situazione delle interfacce di un nodo riguardo allo stato di *sleeping*. Il vettore sarà riempito con il valore *1* se l'interfaccia corrispondente alla posizione è stata posta in *sleeping*, con lo *0* altrimenti. La dichiarazione di questo vettore va inclusa in questo file, in quanto di tipo *header*;

- ifTable.c: localizzato all'interno della cartella di Net-SNMP con lo stesso percorso del file precedente. Le modifiche incluse in questo file sono quelle che permettono di intervenire sui valori di `sleepID[ ]` e quindi di discriminare un'interfaccia in *sleeping* dalle altre;

- ifTable_constants.h: il percorso di questo file all'interno della cartella di installazione di Net-SNMP è: `/agent/mibgroup/if-mib/data_access`. In questo file è presente la porzione di codice responsabile dell'interazione tra stato operativo e stato amministrativo dell'interfaccia. Per questo motivo è stato necessario attuare diverse modifiche a questo file per far sì che il comportamento di Net-SNMP fosse quello desiderato.

L'ultimo passo è stato rendere operative le modifiche all'interno delle macchine virtuali di Netkit. Abbiamo dotato ogni macchina virtuale della nuova versione della suite Net-SNMP, sostituendo la versione preinstallata. Lo scenario emulato in questo modo può disporre a pieno del nuovo modulo MIB comprensivo dello stato a basso consumo.

5.5 Quagga

Per poter lavorare con i protocolli di instradamento Netkit include una versione del software di routing Quagga [8], derivante dal più datato progetto GNU Zebra [33] sviluppato da Kunihiro Ishiguro e ormai abbandonato sin dal 2005.

I protocolli di routing si occupano della diffusione delle informazioni riguardanti le destinazioni disponibili in una rete, al fine di aggiornare automaticamente le tabelle di instradamento di ciascun dispositivo. Quagga, a differenza dei software tradizionali composti da un unico processo che fornisce tutte le funzionalità di routing, è una suite di demoni che forniscono il supporto per diversi protocolli.

Sono presenti diversi demoni specifici per i vari protocolli e *zebra*, un demone che svolge il ruolo di kernel routing manager. Il demone *ripd* gestisce il protocollo RIP, mentre *ospfd* è il demone che supporta OSPFv2, *bgpd* invece è responsabile di BGP-4.

In Figura 5.6 [44] viene riportata un'astrazione dell'architettura di Quagga e del modo in cui vengono propagate le informazioni nella tabella di routing del kernel. Ogni demone, attraverso un file di configurazione, gestisce uno specifico protocollo. Ognuno di essi ascolta su differenti porte TCP, in modo che i messaggi riferiti ad un particolare protocollo di routing possano essere inviati al demone appropriato. Per ogni protocollo di instradamento vengono mantenute una *Routing Information Base* (RIB) e una *Forwarding Information Base* (FIB). La RIB è l'insieme di tutte le destinazioni note ad un determinato protocollo, unite al percorso per raggiungerle e alcune informazioni aggiuntive riguardanti la raggiungibilità. La FIB contiene, per ogni possibile destinazione nella rete, solo il miglior percorso per raggiungerla. Zebra

in sé non è altro che un demone di routing: riceve informazioni dalle FIB degli altri demoni e, per ogni destinazione, seleziona il miglior percorso reso disponibile dai vari protocolli di routing. Le migliori rotte selezionate vengono poi immesse nella tabella di instradamento del kernel, la quale verrà usata per trasmettere effettivamente i pacchetti.

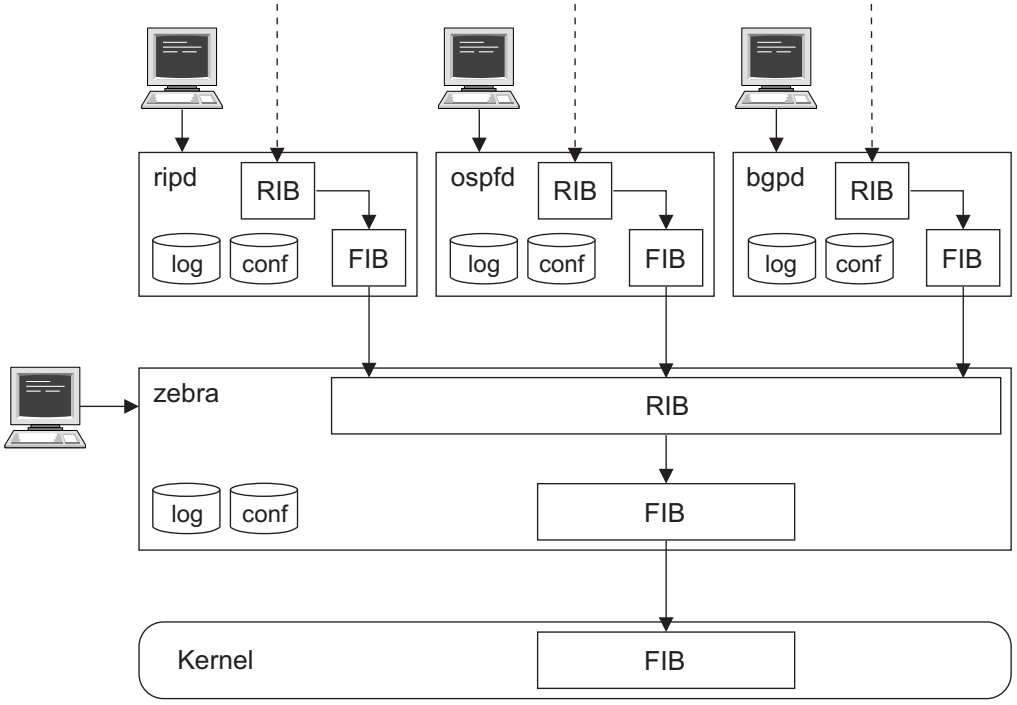

Figura 5.6: *La figura descrive l'architettura del software di routing Quagga. Ogni dispositivo può accedere alle informazioni dei demoni tramite telnet. Le frecce tratteggiate rappresentano le connessioni che permettono lo scambio di informazioni fra i demoni eseguiti sui diversi dispositivi. Le frecce continue rappresentano la propagazione degli aggiornamenti sul routing.*

Tutti i demoni, incluso zebra, possono essere interrogati tramite telnet su una porta dedicata per controllarne lo stato di funzionamento o effettuare configurazioni "on the fly". I demoni forniscono un'interfaccia a linea di comando molto simile a quella dei router Cisco e la maggior parte dei comandi disponibili sui dispositi reali viene riprodotta. Tuttavia ogni demone è in grado di gestire i soli comandi riferiti al protocollo che implementa.

Ogni demone che gestisce un protocollo di routing va configurato attraverso uno

specifico file, nel caso di OSPF si tratta di *ospfd.conf* al cui interno vanno specificati il processo OSPF, il router-ID, i timer, la metrica, le interfacce con relativa area di appartenenza ecc.

Quagga in sé non supporta le funzionalità di gestione delle MIB degli agenti SNMP ma è in grado di connettersi ad un agente SNMP attraverso il protocollo SMUX (SNMP multiplexing protocol) [45] e rendere così disponibile il routing di tali informazioni. Ci sono diversi agenti SNMP che supportano SMUX ma gli autori di Quagga consigliano di usare Net-SNMP, il quale deve essere compilato con l'opzione `--with-mib-modules=smux` affinché possa accettare le connessioni da Quagga. Inoltre, per abilitare il protocollo SMUX in Quagga è necessario usare in fase di compilazione l'opzione `--enable-snmp`. Bisogna poi creare una connessione tra l'agente SNMP identificato dal demone *snmpd* e ognuno dei demoni di Quagga. Queste connessioni usano differenti OID e password. Gli OID in questione vengono usati esclusivamente per interconnettere i demoni e non sono utilizzabili per le interrogazioni da parte dei client.

I seguenti OID sono usati nelle comunicazioni tra i demoni snmpd e Quagga:

```
    (OID .iso.org.dod.internet.private.enterprises.gnome)
zebra .1.3.6.1.4.1.3317.1.2.1 .gnomeProducts.zebra.zserv
bgpd .1.3.6.1.4.1.3317.1.2.2 .gnomeProducts.zebra.bgpd
ripd .1.3.6.1.4.1.3317.1.2.3 .gnomeProducts.zebra.ripd
ospfd .1.3.6.1.4.1.3317.1.2.5 .gnomeProducts.zebra.ospfd
ospf6d .1.3.6.1.4.1.3317.1.2.6 .gnomeProducts.zebra.ospf6d
```

Questi OID devono essere inseriti nei file di configurazione *snmpd.conf* e *ospfd.conf* attraverso il comando `smux peer oid`.

5.6 vnStat

Monitorare il traffico di rete su una macchina è un'operazione vitale per la salvaguardia di qualunque attività che si appoggi all'uso della rete. A tale scopo sono nati svariati programmi di monitoring, alcuni anche molto complessi. Esistono poi strumenti più leggeri e più semplici da usare, ma ugualmente efficaci, come vnStat [16].

VnStat nasce principalmente come strumento testuale, ma riesce anche a produrre risultati quali grafici e immagini. L'installazione del programma, che è rilasciato sotto licenza GPL, non richiede complessi passaggi. Può essere installato sia come amministratore che come semplice utente ed il suo utilizzo non è di difficile comprensione. Come detto in precedenza, la filosofia di vnStat è la semplicità d'uso.

VnStat è disponibile solo per piattaforme Linux e BSD poiché la sua attività si basa essenzialmente sulla lettura delle informazioni fornite dal kernel. In pratica, nel tentativo di migliorare le performance del programma evitando di gravare sul tempo di elaborazione della macchina su cui è installato, vnStat non esegue alcuna operazione di *sniffing* sulle interfacce di rete. Le informazioni così raccolte vengono memorizzate in una base di dati testuale, che serve da archivio storico e archivio dei dati cui attingere per estrapolare i report.

Le statistiche e i report variano per dettaglio e tipologia. L'output di default è quello testuale in quanto, come già detto, lo strumento è nato per essere usato da riga di comando. Esiste infine una utile interfaccia in PHP, con la quale poter generare i vari report senza passare dalla riga di comando.

5.6.1 Utilizzo dei link

Tramite il protocollo SNMP è possibile calcolare l'utilizzo di un link [47]. Le variabili contenute in MIB-II sono dei contatori, è quindi possibile leggere questi valori in un intervallo temporale e calcolarne la differenza.

Nel caso di collegamenti half-duplex l'utilizzo di un link risulta:

$$u = \frac{(\Delta ifInOctets + \Delta ifOutOctets) \times 8 \times 100}{\Delta t \times ifSpeed}$$

- Δ`ifInOctets`: differenza tra due letture dell'oggetto ifInOctets, il quale rappresenta il numero di ottetti in ingresso;

- Δ`ifOutOctets`: differenza tra due letture dell'oggetto ifOutOctets, il quale rappresenta il numero di ottetti in uscita;

- `IfSpeed`: oggetto rappresentante la velocità dell'interfaccia;
- Δ`t`: rappresenta l'intervallo temporale di osservazione.

Questa formula tuttavia è semplificata perché non considera l'overhead associato al protocollo.

Implementando questo calcolo in Netkit abbiamo notato che gli oggetti in questione venivano aggiornati in un tempo dell'ordine della decina di secondi, tempo troppo elevato per poter ottenere dei valori affidabili sull'utilizzo delle interfacce. Questo probabilmente è dovuto in parte alla macchina host e in parte a Netkit, che per non gravare troppo sulle prestazioni del sistema, aggiorna questi valori con tempistiche rilassate rispetto a quanto invece avviene per le statistiche del kernel.

A tal proposito è stato necessario trovare un sistema alternativo per il calcolo dell'utilizzo di un link, una soluzione al problema è stata trovata usando il tool vnStat, sopra descritto.

5.7 Iperf

Iperf [3] è uno strumento, scritto in linguaggio *C++*, usato per testare le reti. È in grado di generare pacchetti di dati TCP e UDP e di misurare il throughput di una rete che li trasporta. Iperf consente all'utente di impostare vari parametri che possono essere utilizzati per la prova di una rete, o alternativamente, per ottimizzare la messa a punto della rete stessa. Iperf ha funzionalità sia di client che di server, e può misurare la velocità del traffico tra le due estremità.

Si tratta di un software open source che può essere eseguito su diverse piattaforme tra cui Linux/Unix e Windows ed è supportato dal National Laboratory for Applied Network Research.

Ci sono due versioni di Iperf al momento, Iperf 2.x e Iperf3. Iperf3 è una nuova applicazione, indipendente dal progetto originale, creata con l'obiettivo di ottenere un codice più semplice e performante e una versione della libreria delle funzionalità che possa essere utilizzata in altri programmi. Iperf 2.x è invece il proseguimento

del progetto iniziale ed è la versione da noi utilizzata. Iperf è utilizzabile tramite interfaccia a linea di comando, necessaria per il funzionamento in Netkit; è tuttavia disponibile anche un'interfaccia grafica scritta in Java chiamata Jperf. Per il suo funzionamento richiede che venga designato un server ed un client. Il client deve essere in grado di avviare una connessione verso il server su una specifica porta TCP o UDP. Il server, una volta eseguito, rimane in ascolto in attesa di stabilire una connessione con un client. Il server può essere anche eseguito come demone in background.

Quando viene utilizzato con datagrammi UDP, Iperf permette all'utente di specificare diversi parametri quali le dimensioni del datagramma, la quantità di dati da inviare, la durata della connessione e altri parametri, inoltre fornisce dati riguardo il throughput dei datagrammi e la perdita di pacchetti. Quando viene utilizzato con pacchetti TCP, Iperf permette di specificare diversi parametri tra cui MTU, windows size e durata della connessione e restituisce una misura del throughput del carico utile.

Capitolo 6

Scelte progettuali ed implementazione prototipale

6.1 Scelte progettuali

Il lavoro di tesi qui presentato si focalizza sui primi due scenari applicativi considerati. Per poterli implementare in un ambiente di lavoro basato su Netkit è stato necessario scegliere un linguaggio di programmazione con cui scrivere il codice per interagire con le macchine virtuali.

Tra i vari linguaggi considerati la nostra attenzione è ricaduta su *bash*. *Bash*, acronimo per "bourne again shell", è una shell testuale del progetto GNU usata principalmente nei sistemi operativi Unix. Si tratta di un interprete di comandi che permette all'utente di comunicare col sistema operativo attraverso una serie di funzioni predefinite o di eseguire programmi. *Bash* è in grado di eseguire i comandi che le vengono passati, utilizzando la redirezione dell'input e dell'output per eseguire più programmi in cascata in una pipeline software, passando l'output del comando precedente come input del comando successivo. Oltre a questo, *bash* mette a disposizione un semplice linguaggio di scripting che permette di svolgere compiti più complessi, non solo raccogliendo in uno script una serie di comandi, ma anche utilizzando variabili, funzioni e strutture di controllo del flusso. In informatica un linguaggio di scripting è un linguaggio di programmazione interpretato, destinato in genere a compiti di automazione del sistema o delle applicazioni, o ad essere usato

all'interno delle pagine web. I programmi sviluppati con questi linguaggi vengono detti script e, generalmente, sono semplici programmi il cui scopo è quello di interagire con altri programmi molto più complessi in cui avvengono le operazioni più significative. Gli script si distinguono dai programmi con cui interagiscono, i quali solitamente sono implementati in un linguaggio differente e non interpretato.

Il motivo principale per cui abbiamo scelto questo linguaggio di scripting rispetto ad un linguaggio di programmazione vero e proprio, come il *C* ad esempio, è che *bash* mette a disposizione una più semplice compatibilità con la suite di applicativi Net-SNMP. Consente infatti di manipolare i comandi SNMP (Get, Set e Trap). In particolare ci ha permesso di assegnare le risposte di tali messaggi a variabili utilizzabili all'interno di un codice, cosa che ad esempio *C* non permette di fare in modo così diretto e semplice.

Come già spiegato, l'obiettivo di questa tesi è di permettere il risparmio energetico in una rete, e per fare ciò è necessario spegnere i nodi in essa presenti. Quando un nodo viene spento, o meglio posto in uno stato di *sleeping*, risulta irraggiungibile dal resto della rete. Le uniche strade percorribili per far tornare attivo un nodo consistono in un risveglio programmato nel nodo stesso, o nell'invio di particolari segnali da parte di un'altra entità. Nella realtà quest'ultima possibilità di riattivazione di un dispositivo prende il nome di *Wake on LAN (WOL)* e *Wake on WAN* e permette il risveglio tramite l'invio di un particolare messaggio di livello 2. Per utilizzare questa tecnica è necessario utilizzare un hardware apposito. La scheda madre deve poter supportare tali segnali, le interfacce di rete devono avere un firmware in grado di riconoscere i messaggi e devono essere alimentate con una tensione di stand-by anche quando non sono attive.

Netkit, in quanto sistema emulato, non permette l'uso di tali messaggi e segnali di livello fisico. Pertanto non supporta nessuna forma di risveglio delle macchine virtuali.

Nel lavoro presentato vi è la necessità di spegnere/accendere un nodo di rete, ossia una macchina virtuale di Netkit. Non potendo sfruttare i comandi presenti

in Netkit per l'accensione e lo spegnimento di una macchina virtuale, in quanto richiedevano dei tempi troppo lunghi e non prevedevano nessuno stato di stand-by, abbiamo quindi deciso di agire a livello di interfaccia. Nella nostra implementazione consideriamo quindi un nodo spento quando tutte le sue interfacce sono inattive. Una volta spente tutte le sue interfacce, però, una macchina risulta irraggiungibile se non da sé stessa. Per ovviare a questo problema ci sono venute incontro alcune funzionalità di Netkit, in particolare le cosiddette interfacce tap. Tramite la tap è possibile collegare l'host reale con la macchina virtuale attraverso un collegamento diretto. In questo modo, un nodo con tutte le sue interfacce di rete *sleeping* rimane comunque accessibile tramite l'interfaccia tap. Attraverso questa interfaccia dedicata, tramite dei messaggi *snmpset*, abbiamo reso possibile la riattivazione delle interfacce, risvegliando così il nodo.

Dotando ogni nodo di una tale interfaccia di rete si viene a creare una rete overlay che collega ogni macchina virtuale con la macchina host reale. Le interfacce di questa rete possono essere mantenute sempre attive in quanto non influenzano il normale funzionamento di OSPF nel sistema emulato.

Nel secondo scenario, centralizzato dinamico, per poter far fronte ad un eventuale aumento del traffico in rete, abbiamo previsto che, durante il suo periodo di *sleeping*, un nodo possa risvegliarsi periodicamente per verificare l'eventuale necessità di un risveglio anticipato. Quando un nodo in *sleeping* si risveglia deve chiedere al nodo centrale (coordinatore), che conosce la situazione attuale della rete, se deve rimanere attivo o se può tornare in stato di *sleeping*. Il nodo centrale controllerà il carico della rete e in base ad una soglia prestabilita comunicherà al nodo di riattivarsi o meno.

La stima del carico di rete viene calcolata misurando l'utilizzo delle interfacce del nodo centrale. Questa informazione è locale, pertanto quella adottata può essere considerata una soluzione euristica. Non misuriamo direttamente il carico di tutta la rete perché coinvolgerebbe tutti i nodi e richiederebbe un ulteriore consumo di risorse da parte delle macchine emulate, nonché la trasmissione di molti pacchetti di controllo. Per le nostre prove abbiamo creato reti con un numero molto contenuto

di nodi e l'utilizzo presente su tale nodo rispecchiava il carico della rete.

Per implementare questo scambio di richieste in Netkit abbiamo utlizzato la suite di applicativi Net-SNMP ed in particolare i messaggi *snmptrap*. Net-SNMP mette a disposizione il demone *snmptrapd* che, se opportunamente configurato, permette di gestire le trap. Questo demone è inoltre in grado di eseguire dei programmi alla ricezione di una trap tramite le direttive *traphandle*. Il file da configurare per gestire questi eventi è *snmptrapd.conf*.

Fino alla versione 5.2 di Net-SNMP il demone *snmptrapd* accettava e processava qualunque notifica entrante. Dalla versione 5.3 in avanti, invece, è stato aggiunto un controllo di accesso sulle notifiche. Per far si che che in SNMPv2c vengano ricevute le trap bisogna quindi fornire al demone la community e alcuni parametri, aggiungendo nel file la linea: `authCommunity log,execute,net public`. In questo modo il demone processerà tutte le trap ricevute attraverso la community `public`. Inoltre queste trap potranno essere registrate, utilizzate per eseguire programmi e inviate in rete. A questo punto sarebbe possibile inviare e ricevere correttamente le trap ma non vi sarebbe ancora associato un programma da eseguire. Per poterlo fare, abbiamo dovuto aggiungere sempre nello stesso file di configurazione le direttive traphandle usando la sintassi: `traphandle OID command`. In questo modo viene associato uno specifico programma da eseguire ad un OID associato ad una trap. Nel nostro caso è stato definito:

```
traphandle .1.3.6.1.6.3.777 /caller_utilizzo.sh
traphandle .1.3.6.1.6.3.775 /handler_1.sh
```

in quanto nel sistema emulato abbiamo avuto la necessità di definire due di questi eventi. Un OID (scelto in modo casuale) è associato alla *snmptrap* che svolge il ruolo di richiesta inviata dal nodo in stato di *sleeping* verso il nodo centrale ed un altro è associato alla trap in direzione opposta, usata per mantenere il nodo in stato di *sleeping*.

6.2 Sviluppo e implementazione

Richiamando la breve descrizione dello scenario centralizzato dinamico data nel capitolo precedente, è utile descrivere in modo più dettagliato lo scambio di messaggi tra il router centrale e un router da esso controllato che chiameremo router agente. Come si può vedere in Figura 6.1, è l'host reale che da vita allo scambio di messaggi tra il router centrale ed il router agente. Il router prestabilito deve infatti entrare nello stato di *sleeping* in base al contenuto della tabella oraria che risiede nell'host reale. A questo punto, dopo un periodo prestabilito in fase di implementazione, il router interessato dovrà mandare una richiesta al router centrale per sapere se dovrà essere risvegliato o meno. Il router centrale attiverà un controllo del traffico sulle proprie interfacce, decidendo se la situazione richiede il risveglio degli elementi posti in *sleeping* in base al confronto con una soglia di carico prestabilita. Si possono ora verificare due situazioni. Nella situazione in cui il controllo dia esito negativo, verrà inviato un messaggio al router agente per avvisarlo di rieffettuare la richiesta di controllo. È chiaro come in questo modo possa essere garantito un controllo periodico attraverso il rimbalzo dei messaggi descritti. In caso di esito positivo, invece, sarà lo stesso router centrale che effettuerà il risveglio dei dispositivi in *sleeping*. Questo avverrà attraverso messaggi di *snmpset* e interromperà il rimbalzo dei messaggi di controllo.

La tabella oraria prevede, oltre a degli istanti in cui i dispositivi vanno posti in *sleeping*, anche degli istanti di risveglio. Questi istanti serviranno per riportare la rete a pieno regime nel momento prestabilito, nel caso in cui non siano state riscontrate situazioni di alto carico.

6.2.1 Codici sviluppati

Per la creazione dello scenario descritto nell'ambiente emulato con Netkit, abbiamo affrontato lo sviluppo e l'implementazione di diversi codici che garantissero la compatibilità tra Net-SNMP e l'ambiente emulato implementato con Netkit.

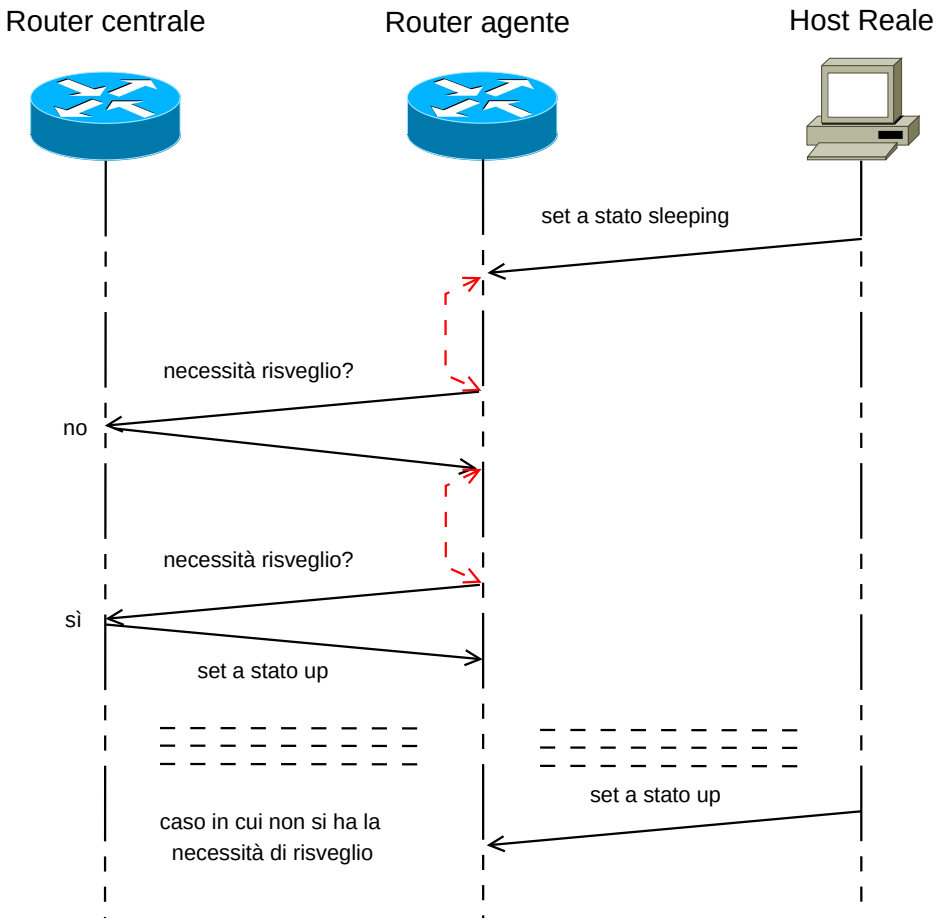

Figura 6.1: *Scambi tra router centrale e router agente*

Il primo programma che entra in gioco è quello responsabile della lettura della tabella oraria, quindi delle operazioni di *snmpset* che permettono alle interfacce dei dispositivi virtuali di entrare nello stato di *sleeping* ed eventualmente di uscirne negli orari prestabiliti. Abbiamo nominato questo file *centralizzato.sh*. La prima parte di *centralizzato.sh* è responsabile dello riempimento delle strutture dati, che ci permette di far capire alla macchina che esegue lo script di capire quali siano gli istanti di transizione delle proprie interfacce. È di estrema importanza includere in questa porzione di codice delle istruzioni che permettano alla macchina virtuale di conoscere la propria identità nella rete overlay, fondamentale per gli scambi di messaggi Net-SNMP. Il codice verrà eseguito in *background* in quanto è necessario che contemporaneamente la macchina esegua altri programmi relativi agli altri elementi

di rete. Con la periodicità del minuto, verrà effettuato il controllo dell'ora locale, confrontata con i dati letti dalla tabella per decidere se eseguire o meno operazioni Net-SNMP. Facciamo notare come la necessità di un'operazione di *snmpset* allo stato *sleeping* attivi allo stesso tempo una *snmptrap* (contenente parametri fondamentali per fare riferimento alla corretta interfaccia del giusto router in fase di ricezione) consente l'inizio del rimbalzo di messaggi visto in precedenza. È necessaria la presenza di questo codice per ciascuna interfaccia di ciascun router della rete emulata per poter gestire la presenza della tabella oraria centralizzata.

Attraverso il meccanismo di *trap handling* fornito da Net-SNMP ci è stato possibile reagire ad una *snmptrap* mandando in esecuzione un altro codice: *handler_1.sh*. Abbiamo avuto la necessità di utilizzare questa struttura di codici annidati per poter stare al passo con i tempi di esecuzione. Infatti, abbiamo verificato l'impossibilità di effettuare operazioni che richiedessero un certo tempo di calcolo all'interno dei codici gestiti dal *trap handler*, che causavano una situazione di stallo del demone *snmptrapd*. La conseguenza sarebbe la nascita di una coda delle *snmptrap* che sarebbe fonte di problemi di sincronismo tra i codici. L'esecuzione di questo codice permette al router agente di conoscere esattamente quale sia la sua interfaccia che è stata messa in stato di *sleeping*. La funzione principale di questa porzione di codice è il richiamo in *background* di *scenario2.sh* con l'opportuno passaggio di parametri.

È in *scenario2.sh* che abbiamo incluso l'impostazione del tempo di attesa che precede l'invio della messaggistica di controllo per le decisioni di riaccensione, data dall'invio della *snmptrap* al router centrale. Questo parametro, come mostrato nella sezione dei risultati, si dimostra di notevole importanza per quanto riguarda le prestazioni dello scenario implementato.

Come per la situazione vista in precedenza, anche in questo caso, abbiamo sfruttato il *trap handling* di Net-SNMP per la reazione alla *snmptrap* di richiesta di controllo del carico sulla rete. Abbiamo quindi sviluppato presente un programma, denominato *caller_utilizzo.sh*, che legge i parametri della *snmptrap* per passarli al

codice che esegua i controlli veri e propri.

Come fulcro decisionale di questo scenario, abbiamo scelto di implementare il programma *handler_utilizzo.sh*, eseguito in *background* dal router centrale. Dopo lo riempimento delle strutture dati con i parametri passati, il codice procede con la valutazione delle condizioni della rete monitorando lo stato delle interfacce del router centrale. Abbiamo pensato di effettuare il controllo attraverso la lettura dei file prodotti da vnStat, che contengono il traffico, in kbits, che fluisce attraverso le interfacce sopracitate. L'implementazione prevede quindi il calcolo dell'utilizzo medio riscontrato per queste interfacce, dato che servirà per il confronto con la soglia prestabilita (altro parametro di fondamentale importanza per un risultato apprezzabile). Il passaggio di questo controllo determinerà il risveglio dei dispositivi in stato di *sleeping* tramite *snmpset*. Il caso opposto rimanderà all'invio di una *snmptrap*, utile per continuare il processo di rimbalzo periodico delle messaggistica di controllo.

6.2.2 Esempio di funzionamento

Per comprendere meglio il funzionamento dello scenario in questione, coinvolgendo l'interazione tra i diversi codici implementati nelle macchine reali e virtuali, è utile affidarsi ad un semplice esempio. Facendo riferimento a Figura 6.2, verranno di seguito elencati tutti gli eventi provocati da una *snmpset* allo stato *sleeping* del router virtuale 2 (R2).

1. Il processo comincia con l'esecuzione del codice *centralizzato_R2.sh*, che risiede nella macchina reale, che in base alla tabella oraria ordina la *snmpset* allo stato *sleeping* su R2. Contemporaneamente viene inviata una *snmptrap* a R2 per ogni interfaccia interessata. La ricezione della *snmptrap* attiva *handler_1.sh* che a sua volta causa l'esecuzione in *background* di *scenario2.sh*;

2. l'esecuzione di *scenario2.sh* prevede l'invio di un'ulteriore *snmptrap* al router virtuale manager (RM) per avvisarlo che R2 è pronto ad un risveglio in caso

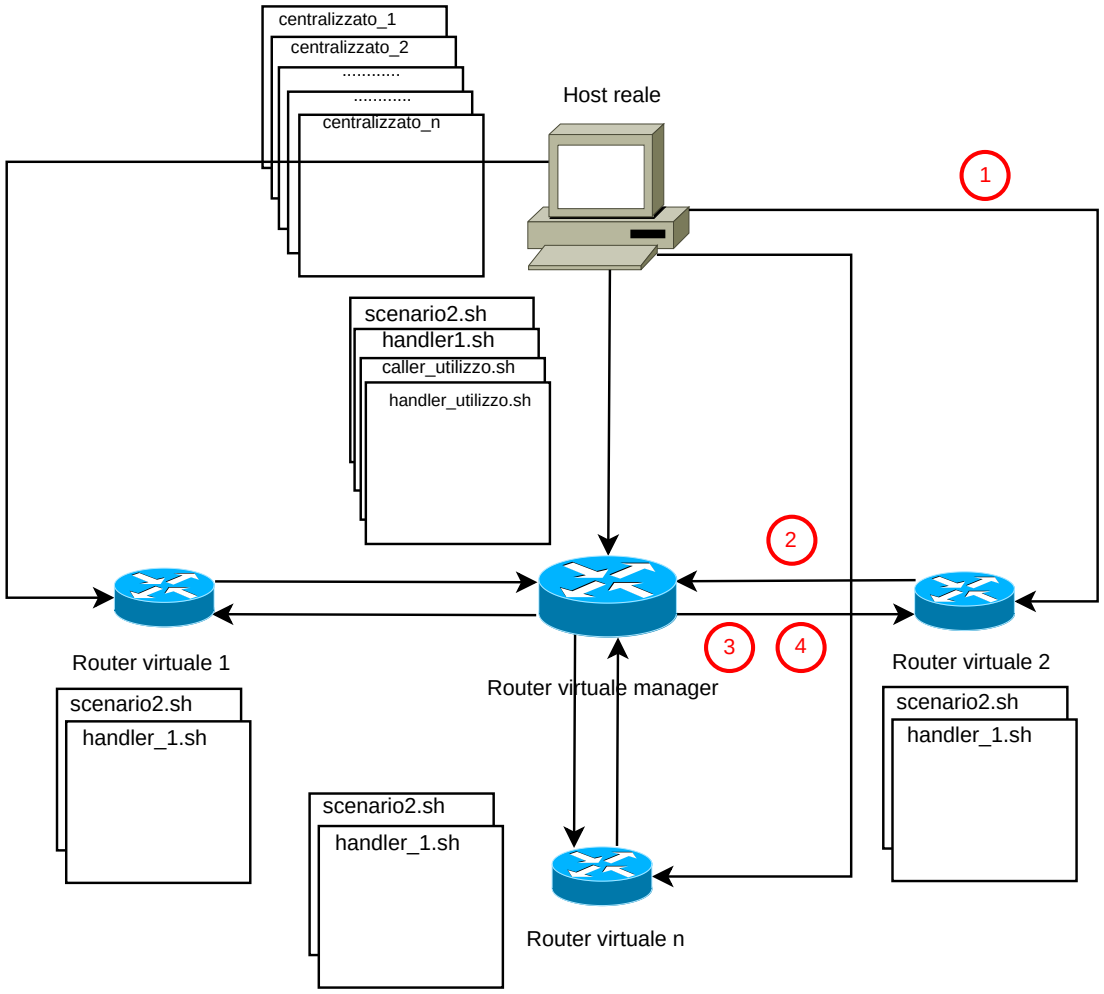

Figura 6.2: *Esempio di funzionamento dello scenario centralizzato dinamico*

di necessità. Vengono inoltrati i dati necessari per riconoscere il richiedente: indirizzo IP (TAP) e indice di interfaccia. La ricezione della *snmptrap* da parte del RM è responsabile dell'esecuzione di *caller_utilizzo.sh*, che a sua volta lancia in *background* il codice responsabile dei controlli e quindi delle decisioni, *handler_utilizzo.sh*. Quest'ultimo, in base agli esiti del controllo, porterà l'esecuzione al punto 3 oppure al punto 4 della descrizione;

3. il controllo da parte di *handler_utilizzo* sullo stato della rete, in questo caso, ha dato esito positivo. Sarà quindi effettuata un'operazione di *snmpset* per provvedere al risveglio dei dispositivi posti in stato di *sleeping* per far fronte

alla situazione di carico riscontrata. Arrivati a questo punto le interazioni automatiche tra i codici sono terminate fino al presentarsi di un nuovo evento di transizione nella tabella oraria centralizzata;

4. l'esito del controllo è stato negativo, R2 potrà quindi essere lasciato nello stato a basso consumo energetico fino ad un'ulteriore richiesta di controllo. Questa situazione è gestita tramite l'invio a R2 di una *snmptrap* che ha lo stesso effetto di quella descritta al punto 1, dando vita ad un nuovo ciclo di controllo della situazione di carico della rete.

6.3 Test e risultati

Per testare il funzionamento degli script e delle modifiche illustrate nella sezione precedente, abbiamo configurato in Netkit la topologia di rete illustrata in Figura 6.3. Sono evidenziati gli indirizzi dei domini di collisione e le tap utilizzate per ogni macchina virtuale. Il router B0 svolge il ruolo di host virtuale manager (v. Figura 4.4).

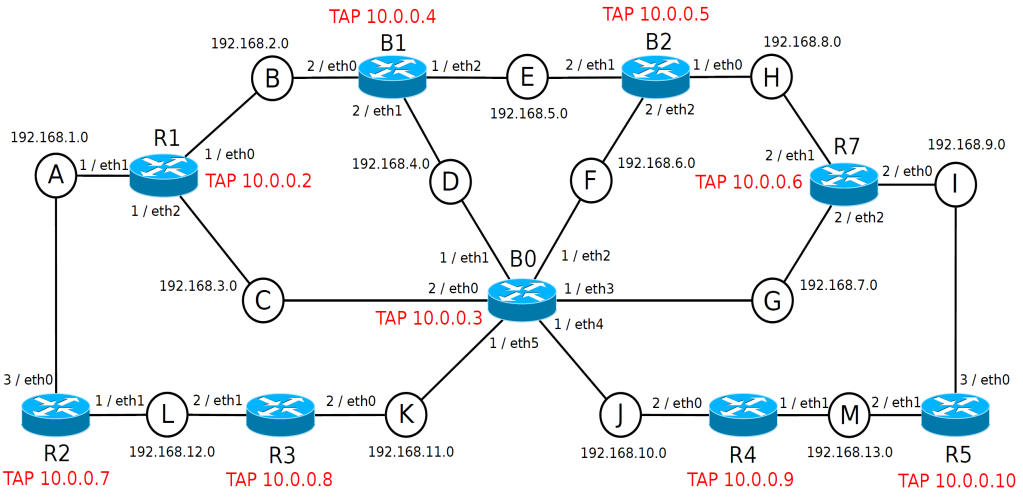

Figura 6.3: *Rete emulata in Netkit - Scenario centralizzato dinamico*

I route R2, R5 e B2 sono invece sorgenti e destinazioni di traffico. Tutti i router comunicano con l'host reale attraverso le tap, il cui funzionamento è stato descritto

nel paragrafo 4.3. La capacità di ogni collegamento è stata limitata via software ad 1 Mbit/s poiché in Netkit dipende dalle potenzialità della macchina reale e non è possibile sfruttare i valori standard. Vengono posti in sleeping i router R1, R7 e B1 e le interfacce di B2, R2 ed R5 ad essi collegate.

Il superamento del 50% di utilizzo di un link determina le riaccensioni: tale soglia distingue, nel nostro caso, le situazioni di basso carico da quelle di alto carico.

I test sono stati effettuati nel seguente modo:

- **condizione di basso carico**: abbiamo configurato la tabella oraria degli spegnimenti e delle riaccensioni sulla quale si basa il funzionamento degli script di gestione. Abbiamo realizzato test della durata di 6 minuti a rete totalmente scarica. In questo arco di tempo, abbiamo calcolato la banda occupata dal traffico di gestione SNMP. La valutazione della dimensione dei pacchetti è stata effettuata sfruttando lo *sniffer tcpdump* sulle interfacce tap, sulle quali vengono trasmessi e ricevuti sia i messaggi di *snmpget* ed *snmpset* che le *trap*. In Figura 6.4 vengono illustrati i risultati ottenuti, al variare della periodicità dei controlli dell'utilizzo dei collegamenti.

- **condizione di alto carico**: per riprodurre tale situazione abbiamo utilizzato iperf, che consente di generare ed iniettare pacchetti all'interno della rete; è prevista, inoltre, la possibilità di specificare il protocollo da utilizzare, la dimensione dei pacchetti e la portata dei flussi di traffico generati. Il procedimento seguito è stato il seguente:

  - **spegnimento interfacce**: vengono poste in sleeping tutte le interfacce di R1, R7 e B1 e quelle di B2, R5 ed R2 ad essi collegate.

  - **avvio di iperf**: successivamente agli spegnimenti viene avviato iperf sulle macchine virtuali B2, R2 ed R5 per caricare la rete.

  - **calcolo dei tempi di riaccensione**: viene infine calcolato il tempo

necessario a riattivare ciascuna delle interfacce poste in sleeping in precedenza.

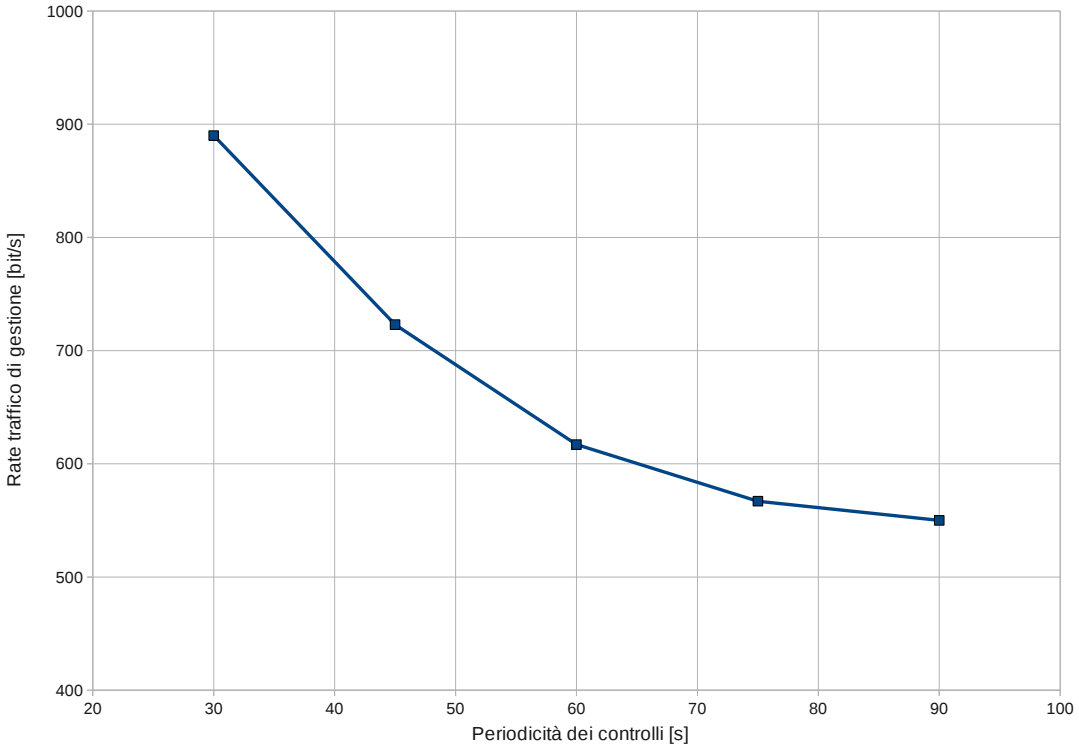

Figura 6.4: *Appare evidente come, all'aumentare dell'intervallo di controllo, si abbiamo una diminuzione della banda occupata dal traffico di gestione.*

L'aumento del traffico causa la riaccensione delle interfacce poste in stato di sleeping solamente quando l'utilizzo dei collegamenti supera il 50%. Abbiamo calcolato una media dei tempi di riaccensione delle varie interfacce e le prove sono state effettuate, come nel caso di basso carico, al variare della periodicità dei controlli. I risultati sono illustrati in Figura 6.5, mentre in Figura 6.6 viene mostrato il risparmio energetico a livello di nodi e link: è evidente come l'implementazione dello sleeping porti ad una notevole riduzione dei consumi, con apprezzabili risultati sia per i nodi che per i link. Il risparmio complessivo ottenuto è stato del 39% e sono stati utilizzati i dati di consumo del router Juniper M10i [1]. Nella Tabella 6.1 abbiamo riportato il consumo in watt del solo chassis, mentre nella Tabella 6.2 ci sono

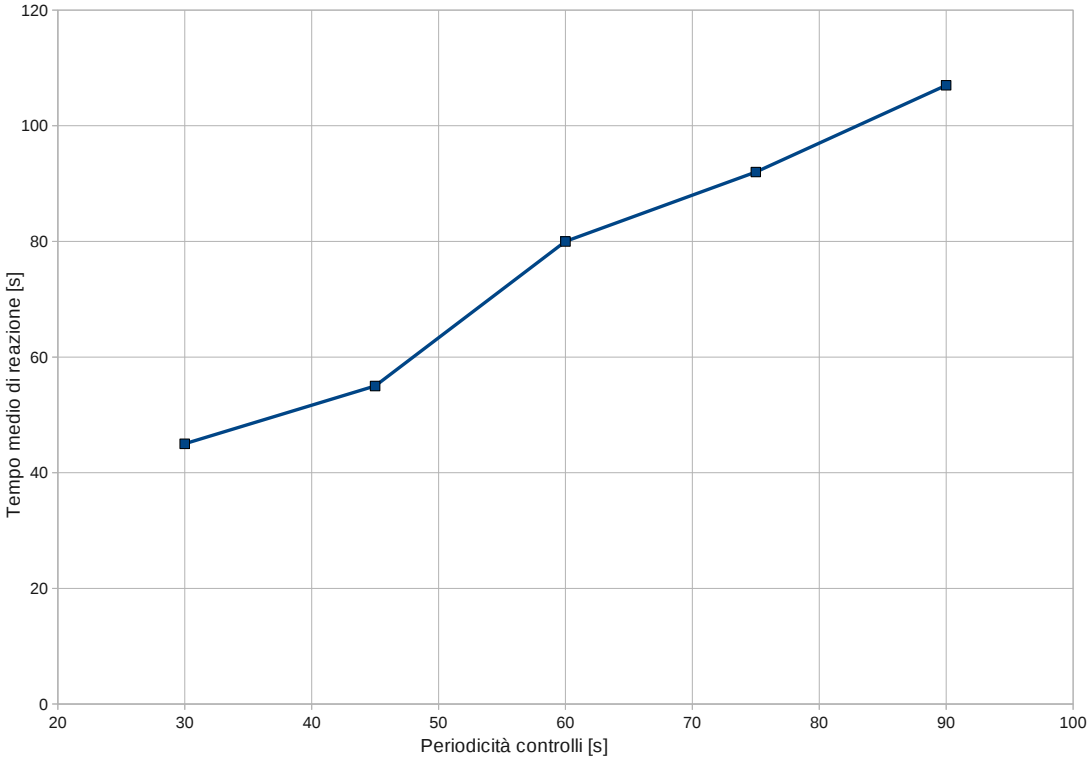

Figura 6.5: *Si nota come, all'aumentare dell'intervallo di controllo, si abbiano riaccensioni più lente.*

i consumi di alcune delle line card supportate. Per i test effettuati abbiamo scelto di utilizzare la scheda Gigabit Ethernet da 2 Gbps perché equipaggiata di una sola porta. Questo ci permette di calcolare in modo più accurato il consumo dei nodi, attribuendo una scheda per ogni interfaccia.

Chassis type	Capacity (Gbps)	# PIC slot	Chassis Power (W)
Juniper M10i	16	8	86.4

Tabella 6.1: *La tabella descrive le caratteristiche principali del router M10i, il dato di maggior interesse è il consumo in potenza.*

6.4 Considerazioni

Dai test effettuati è emerso come, al variare della periodicità dei controlli, cambino contemporaneamente sia il tempo di reazione in caso di aumento del carico della

Type	Capacity (Gbps)	# ports	# slots	Power (W)
Fast Ethernet	0.8	4	1	6.8
Fast Ethernet	1.6	8	1	12.5
Gigabit Ethernet	2	1	1	7.3
Fast Ethernet	2.4	12	1	11
Gigabit Ethernet	8	4	1	31

Tabella 6.2: *Line cards per il router M10i*

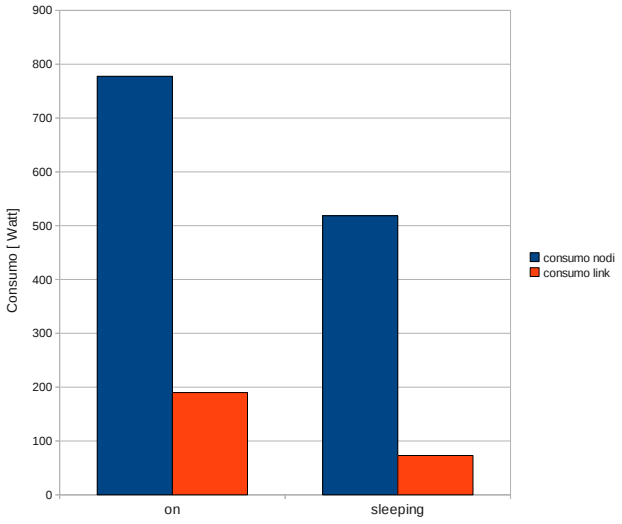

Figura 6.6: *Viene illustrato il risparmio energetico dato dall'implementazione dello sleeping sia a livello di nodi che di link.*

rete, sia la banda occupata dal traffico di gestione SNMP. Questo comportamento era facilmente prevedibile ed evidenzia la necessità di una soluzione di compromesso, in grado di coniugare una buona reattività ad un contenuto aumento del traffico di gestione.

Naturalmente, lo spegnimento dei dispositivi determina cambiamenti topologici all'interno della rete: questi causano un aumento degli scambi di messaggi del protocollo OSPF. La frequenza di questi messaggi è stata mantenuta sui valori originali del protocollo, e non è stata fatta distinzione tra sleeping e guasti per ragioni di semplicità. Una possibile implementazione dello stato di basso consumo energetico potrebbe essere fatta, per esempio, dedicando un bit all'interno dei pacchetti di

Hello che dica se un dispositivo è in sleeping oppure no. È evidente infatti, come già sottolineato, che un dispositivo in sleeping debba comunque mantenere le relazioni con gli altri nodi della rete inviando traffico protocollare, anche se meno frequentemente. Si potrebbero utilizzare degli OSPF Hello Interval più elevati. Questo permetterebbe ad un nodo di distinguere i vicini che sono attivi da quelli che sono stati posti in uno stato di basso consumo energetico: per via della struttura del protocollo, ciò richiederebbe una modifica della configurazione dei nodi in maniera tale da prevedere la possibilità che vi siano, all'interno del medesimo AS, pacchetti con Hello Interval differenti.

Il procedimento da noi seguito, inoltre, potrebbe essere migliorato in futuro: il comportamento attuale degli script determina, al superamento delle soglie di utilizzo dei collegamenti, la riaccensione di tutti i dispositivi in sleeping. Realizzare una procedura che riaccenda solo il necessario potrebbe migliorare di molto le prestazioni in termini di risparmio energetico.

Infine, allo stato attuale vengono inviate *trap* per valutare l'utilizzo di ogni singola interfaccia: per ridurre il traffico di gestione, sarebbe utile poter accorpare in singoli messaggi le informazioni relative ad un intero nodo.

Capitolo 7

Conclusioni

Il consumo di energia in Internet stà guadagnando sempre maggiore attenzione, e lo sleep è una delle tecniche più efficaci per risparmiare energia. Tuttavia, i router in una rete non possono sfruttare semplicemente la modalità di sleep per il risparmio energetico, perché si scambiano continuamente informazioni di routing attraverso i pacchetti non-dati, e mettere alcuni di essi in sleep può causare il blocco delle connessioni. In questa tesi abbiamo analizzato il problema dello sleep dei nodi di rete in uno dei principali protocolli di routing oggi utilizzati, OSPF.

Dopo una descrizione della struttura di rete e di come il consumo sia distribuito in essa, abbiamo analizzato le tecniche di risparmio energetico disponibili in letteratura, le tecniche di sleeping e le procedure di adattamento del rate.

Ci siamo poi occupati della descrizione del protocollo OSPF e del suo funzionamento, necessaria per capire come poter sfruttare all'interno di esso un nuovo stato di basso consumo, illustrando poi anche il protocollo SNMP, usato per gestire il nuovo stato di sleep introdotto nel MIB all'interno di Netkit.

Partendo dal problema posto da Gupta e Singh nel loro articolo [29] e analizzando i loro suggerimenti, siamo arrivati a proporre quella che riteniamo essere una buona soluzione al risparmio energetico in reti OSPF in grado di sfruttare lo stato di sleep. Per cercare di mantenere il più possibile inalterato il protocollo abbiamo demandato la distinzione logica tra nodo down e sleep ad un livello superiore, introducendo que-

sto nuovo stato all'interno della IF-MIB standard, responsabile della gestione delle interfacce di rete dei dispositivi. In questo modo, ai fini dell'instradamento, OSPF tratta un nodo down ed uno sleep nello stesso modo. Questo comportamento ha lo svantaggio di non rendere disponibile ad OSPF l'informazione sullo sleep di un nodo. Per ovviare a questo abbiamo previsto che il nodo in stato di sleep (ricordiamo che si tratta di uno stato di basso consumo in cui il nodo non è fisicamente spento), possa inviare dei messaggi simili ai normali pacchetti *Hello* ai suoi vicini per informarli del suo particolare stato. Questa informazione dovrà poi essere distribuita nei nodi in modo che il/i coordinatori possano agire di conseguenza in caso di variazioni del traffico.

Congiuntamente alla soluzione abbiamo proposto anche alcuni scenari applicativi basati su due diversi approcci: uno centralizzato ed uno distribuito, il primo dei quali è discusso in questa tesi. Questi scenari sono stati implementati in un ambiente di lavoro basato sull'emulatore di reti Netkit.

Abbiamo sviluppato una piattaforma che fosse in grado di gestire il basso consumo energetico in una rete OSPF attraverso il protocollo SNMP. Per farlo, abbiamo creato una serie di script che permettessero, in una rete emulata in Netkit, lo spegnimento e l'eventuale riaccensione delle interfacce di rete delle macchine virtuali.

Infine, abbiamo illustrato i risultati ottenuti con una piccola rete OSPF. Dai test effettuati si apprezzano buone prestazioni in termini di risparmio energetico, intorno al 35%. Per poter reagire ai cambiamenti di stato della rete, è necessario un continuo scambio di messaggi SNMP per monitorare lo stato dei collegamenti. È emerso come, al variare della frequenza di tali messaggi, si abbiano cambiamenti relativi sia ai tempi di reazione che alla banda occupata dal traffico di gestione. Occorre trovare una soluzione di compromesso che combini un contenuto scambio di messaggi SNMP ad una buona rapidità di reazione in caso di variazioni dello stato della rete.

Abbiamo optato per una soluzione che, in caso di aumento del traffico, riaccen-

da tutte le interfacce, e di conseguenza i nodi, posti in *sleeping*. Abbiamo inoltre assunto che lo spegnimento di tutte le interfacce di rete di un router equivalga allo spegnimento del router stesso. Per migliorare la nostra piattaforma, si potrebbero effettuare modifiche tali da garantire la riaccensione, in caso di variazioni dello stato di rete, solamente dei dispositivi strettamente necessari al mantenimento della connettività e della QoS. In una rete reale, infine, si potrebbero porre in *sleep* gli interi nodi, sfruttando la WOL in caso di necessità di riaccensioni.

Bibliografia

[1] Datasheet router Juniper. http://wwww.juniper.net/.

[2] GARR - The Italian Academic and Research Network. http://www.garr.it/.

[3] Iperf. http://iperf.sourceforge.net/.

[4] Kyron. Linux Virtual Server. http://www.kyron.it/virtualserver.asp.

[5] Linode.com. Virtual Private Server. http://linode.com/.

[6] Net-snmp. http://www.net-snmp.org/.

[7] Netfilter – Firewalling, NAT, and Packet Mangling for Linux. http://www.netfilter.org/.

[8] Quagga Routing Suite. http://www.quagga.net/.

[9] Sourceforge. http://sourceforge.net/.

[10] The Linux Kernel Archives. http://www.kernel.org/.

[11] The Network Simulator - NS-2. http://www.isi.edu/nsnam/ns/.

[12] UML Utilities. http://user-mode-linux.sourceforge.net/dl-sf.html.

[13] University of Roma Tre Computer Networks Research Group. Netkit. http://www.netkit.org/.

[14] University of Roma Tre Computer Networks Research Group. Netkit ready to use Labs. http://www.netkit.org/labs.html.

[15] User-mode Linux Kernel. http://user-mode-linux.sourceforge.net/.

[16] vnStat. http://humdi.net/vnstat/.

[17] B. Addis, A. Capone, G. Carello e B. Sansò. *Dynamic and Coordinated Energy Management through Optimized Routing and Device Powering for Greener Communication Networks*. 2010.

[18] L. Berger, I. Bryskin, A. Zinin, and R. Coltun. *The OSPF Opaque LSA Option*. RFC 5250, 2008.

[19] C. Gunaratne, K. Christensen e B. Nordman. *Managing energy consumption costs in desktop PCs and LAN switches with proxying, split TCP connections, and scaling of link speed*. International Journal of Network Management, Settembre 2005.

[20] C. Kalmanek, M. Rumsewicz, J. Yates, Y. Zhang, M. Roughan e A. Reenberg. *Experience in measuring internet backbone traffic variability: Models, metric, measurements and meaning*. In Proceeding of the 2nd ACM SIGCOMM Workshop on Internet Measurement, Marseille, France, 2002.

[21] Jeff Dike. *A User-Mode Port of the Linux Kernel*. In Proc. 4th Annual Linux Showcase and Conference, pagine 63–72, Ottobre 2000.

[22] Jeff Dike. *Double your Fun with User-Mode Linux*. Novembre 2004.

[23] Jeff Dike. *User-Mode Linux*. Talk at the 2001 Ottawa Linux Symposium, Luglio 2001.

[24] Jeff Dike. *User Mode Linux*. Prentice Hall, 2006.

[25] S. Abeck, B. Neumaier e H. Hegering. *Integrated Management of Networked System*. Morgan Kaufmann, 1999.

[26] J. F. Kurose e K. W. Ross. *Internet e Reti di Calcolatori, seconda edizione*. McGraw-Hill, 2003.

[27] K. McCloghrie e M. Rose. *Management Information Base for Network Management of TCP/IP-based internets:MIB-II*. RFC 1213 (Standard), 1991. Aggiornata da RFC 2011, 2012, 2013.

[28] H. Morikawa e M.Minami. *Some Open Challenges for Improving the Energy Efficiency of the Internet*. Proc. 3rd International Conference on Future Internet (CFI 2008), Giugno 2008.

[29] M. Gupta e S. Singh. *Greening of the Internet*. In Proceedings of the ACM conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM 2003),(Karlsrhue, Germany), Agosto 2003.

[30] Thomas M. e Thomas II. *OSPF Network Design Solution*. Cisco Press, 2003.

[31] Paolo Giarrusso. SKAS patches and UML updates. http://www.user-mode-linux.org/blaisorblade/.

[32] Paolo Giarrusso. UML Utilities. http://www.user-mode-linux.org/blaisorblade/uml-utilities/.

[33] Kunihiro Ishiguro. GNU Zebra Routing Software. http://www.zebra.org/.

[34] K. Hinton, J. Baliga, R. S. Tucker. *Energy Consumption of the Internet*. In Proceedings of the Joint International Conference on Optical Internet and the 32nd Australian Conference on Optical Fibre Technology. (COIN-ACOFT 2007), Melbourne, Australia, Giugno 2007.

[35] K.J. Christensen e P.C. Gunaratne. *Predictive power management method for network devices*. IEEE 2005 29 March 2005, Electronics Letters online no: 20051149.

[36] Brad Marshall. User Mode Linux, VMWare and Wine – Virtual Machines under Linux. http://quark.humbug.org.au/publications/linux/uml.pdf, Aprile 2003.

[37] K. McCloghrie, D. Perkins, and J. Schoenwaelder. *Structure of Management Information Version 2 (SMIv2).* RFC 2578 (Standard), 1999.

[38] J. Moy. *Multicast Extensions to OSPF.* RFC 1584, 1994.

[39] J. Moy. *OSPF Version 2.* RFC 2328 (Standard), 1998.

[40] John T. Moy. *Anatomy of an Internet Routing Protocol.* Addison-Wesley, 1998.

[41] P.C. Gunaratne, K.J. Christensen, B. Nordman e S. Suen. *Reducing the Energy Consumption of Ethernet with Adaptive Link Rate (ALR).* IEEE Transactions on Computers 57, Aprile 2008.

[42] R. Presuhn. *Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP).* RFC 3416 (Standard), 2002.

[43] R. Jain e I. Chlamtac. *The P2 algorithm for dynamic calculation of quantiles and histograms without storing observations.* 1985.

[44] Massimo Rimondini. *Emulation of Computer Networks with Netkit.* Gennaio 2007.

[45] M.T. Rose. *SNMP MUX protocol and MIB.* RFC 1227, 1991.

[46] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy e D. Wetherall. *Reducing Network Energy Consumption via Sleeping and Rate-Adaptation.* In Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NDSI 2008) (USENIX Association, ed.), (San Francisco, California, USA), Aprile 2008.

[47] Cisco Systems. *How To Calculate Bandwidth Utilization Using SNMP.* Documento 8141, 2005.

Ringraziamenti

Vogliamo in primo luogo ringraziare il professor Antonio Capone per averci dato la possibilità e gli spunti necessari per portare avanti questo progetto.

Un ringraziamento va a tutti i professori ed i collaboratori del DEI che hanno risposto alle nostre richieste di aiuto.

Inoltre vogliamo ringraziare Emanuele Colombo (MobiMESH), la mailing list del progetto Netkit e il nostro collega Luca Giovanni Gianoli.

Un grande grazie va ai nostri genitori e familiari che ci hanno supportato durante tutto il nostro percorso universitario. Con grande fiducia ed affetto si sono sempre dimostrati disponibili ad aiutarci ed incoraggiarci anche nei momenti di maggiore difficoltà.

Ringraziamo tutti i colleghi che hanno condiviso con noi le interminabili giornate universitarie, tra lezioni e svaghi: Bazza, Casi, Dario cipo, J-ovà, Fede, Giusso, Dario big, Bucc, Nic, Luca, Roby, Ste, il Cerri, Teo e tutti quelli che hanno scelto un'altra strada.

Un ringraziamento particolare va a Luca, membro importante del nostro team senza il quale questo lavoro non sarebbe stato possibile.

Da parte di Emanuele, un ringraziamento speciale a Mari, per l'affetto e la comprensione con i quali mi ha seguito durante questi ultimi tre anni, supportandomi nei momenti di difficoltà e rendendo ancor più piacevoli quelli di gioia. Un saluto particolare anche a mia nonna Lina, venuta mancare due anni fa, grazie di tutto!

Infine, un ringraziamento sincero agli amici di sempre con i quali abbiamo condiviso gioie e difficoltà anche in questi ultimi cinque anni, dimostrando che su di loro si può sempre contare.

Grazie a tutti, Emanuele & Riccardo