

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 DISTRIBUITO

Relatore: Prof. Antonio Capone

Tesi di Laurea di:  
Luca Fusari  
Matr. 734485

Anno Accademico 2009 – 2010



# Indice

<b>Sommario</b>	<b>VII</b>
<b>Introduzione</b>	<b>1</b>
<b>1 Tecniche di risparmio energetico</b>	<b>5</b>
1.1 Struttura di rete e consumo energetico . . . . .	6
1.2 Tecniche di sleeping . . . . .	7
1.3 ALR - Adaptive Link Rate . . . . .	12
<b>2 Open Shortest Path First (OSPF)</b>	<b>15</b>
2.1 Base di dati link-state . . . . .	16
2.2 Suddivisione in aree . . . . .	17
2.3 Protocollo di Hello e creazione di adiacenze . . . . .	20
2.4 Propagazione delle informazioni, i pacchetti LSA . . . . .	21
2.5 Stati delle interfacce . . . . .	24
<b>3 Network Management</b>	<b>27</b>
3.1 SNMP . . . . .	27
3.2 Architettura . . . . .	29
3.3 Management Information Base . . . . .	31
3.3.1 Structure Management Information . . . . .	32
3.4 Messaggi SNMP . . . . .	34

---

<b>4</b>	<b>Soluzione proposta</b>	<b>39</b>
4.1	Scenari di risparmio energetico . . . . .	44
4.1.1	Scenario centralizzato statico . . . . .	45
4.1.2	Scenario centralizzato dinamico . . . . .	46
4.1.3	Scenario distribuito . . . . .	47
<b>5</b>	<b>Ambiente di lavoro</b>	<b>49</b>
5.1	Ambiente di emulazione ed architettura di Netkit . . . . .	51
5.2	User-Mode Linux kernel . . . . .	55
5.3	Supporto di rete in Netkit . . . . .	57
5.4	Net-SNMP . . . . .	60
5.4.1	Cenni storici . . . . .	60
5.4.2	Funzionalità . . . . .	62
5.4.3	Modifiche al codice sorgente . . . . .	63
5.5	Quagga . . . . .	66
5.6	vnStat . . . . .	68
5.7	Iperf . . . . .	69
<b>6</b>	<b>Scelte progettuali ed implementazione prototipale</b>	<b>71</b>
6.1	Scelte progettuali . . . . .	71
6.2	Algoritmo . . . . .	73
6.3	Implementazione . . . . .	76
6.4	Generazione dei dati di test . . . . .	78
6.4.1	Topologie di rete . . . . .	78
6.4.2	Generazione delle matrici di traffico . . . . .	81
6.4.3	Consumi . . . . .	83
6.5	Risultati computazionali . . . . .	83
<b>7</b>	<b>Conclusioni</b>	<b>91</b>
	<b>Bibliografia</b>	<b>95</b>

# Elenco delle figure

1.1	Consumi totali di rete . . . . .	8
1.2	Diagramma di traffico settimanale . . . . .	8
1.3	Chassis semplificato di un router, utilizzato in [19] . . . . .	10
1.4	Struttura di un periodo di inattività . . . . .	11
1.5	Schema dell'algoritmo di sleeping predittivo presentato in [38] . . . . .	11
1.6	Scambio di trame MAC nel meccanismo ALR [47] . . . . .	13
2.1	Esempio di AS monoarea . . . . .	17
2.2	Stati delle interfacce OSPF . . . . .	25
3.1	Architettura di un sistema SNMP . . . . .	30
3.2	Albero delle MIB . . . . .	34
3.3	Formato dei messaggi SNMP . . . . .	38
4.1	Scambio di <i>Hello packet</i> . . . . .	43
4.2	Opaque-LSA . . . . .	43
4.3	Scenario centralizzato statico . . . . .	45
4.4	Scenario centralizzato dinamico . . . . .	46
4.5	Scenario distribuito . . . . .	47
5.1	Risorse di una macchina virtuale di Netkit . . . . .	52
5.2	Esempio di rete emulata con Netkit . . . . .	53
5.3	Architettura di Netkit . . . . .	54
5.4	Connessione tra macchine virtuali in Netkit . . . . .	58

5.5	Stack protocollare di rete emulata in Netkit . . . . .	58
5.6	Architettura di Quagga . . . . .	67
6.1	Topologie di rete . . . . .	80
6.2	Comparazione dei risparmi nelle reti(1) . . . . .	87
6.3	Comparazione dei risparmi nelle reti(2) . . . . .	87
6.4	Comparazione dei risparmi nelle reti(3) . . . . .	88
6.5	Consumo energetico delle reti . . . . .	89

# Elenco delle tabelle

2.1	Grafo della rete OSPF . . . . .	18
6.1	Topologie di reti bacbone, progetto Rocketfuel . . . . .	79
6.2	Suddivisione tra nodi edge e core . . . . .	81
6.3	Specifiche del router M10i . . . . .	83
6.4	Line cards per il router M10i . . . . .	83
6.5	Risultati rete Ebone (1) . . . . .	85
6.6	Risultati rete Exodus (1) . . . . .	85
6.7	Risultati rete Telstra (1) . . . . .	86
6.8	Risultati rete Ebone (2) . . . . .	88
6.9	Risultati rete Exodus (2) . . . . .	89
6.10	Risultati rete Telstra (2) . . . . .	89





# Sommario

## Italiano

Fin dai primi anni novanta, si è potuta osservare una crescita costante dell'utilizzo di Internet. Oggi Internet è presente nella vita di tutti i giorni di molte persone, diventando uno strumento di lavoro insostituibile per molte aziende. Si prevede che nei prossimi anni il suo utilizzo aumenterà ulteriormente, facendo diventare quello dell'ICT uno dei settori leader per quanto riguarda il consumo energetico. Per questo motivo si è sviluppato un grande interesse nella limitazione dei consumi energetici di questo settore.

In questa tesi presentiamo un metodo che permette di porre in *sleeping* i dispositivi di rete poco utilizzati e di gestirne il risveglio. L'approccio mostrato è di tipo distribuito. Tutti i dispositivi di rete eseguono un algoritmo che permette loro di prendere decisioni sul proprio stato, in base alla conoscenza della situazione della rete stessa e mantenendo l'instradabilità delle domande di traffico. Lo scenario considerato è di tipo OSPF nel quale abbiamo sfruttato le potenzialità del protocollo SNMP per ottenere le caratteristiche desiderate.

I risultati mostrano come il risparmio energetico dipenda dalla rete considerata e dal suo carico, permettendo nel caso ottimo di ottenere risparmi superiori al 40%.

## English

Since the early nineties, we can observe a constant growth of Internet utilization. Today, Internet is present in everyday life of a lot of people, becoming a needful working tool for many companies. 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. For this reason a great interest to limit energy consumption in this sector has been arising.

In this work, we present a method which allows to set in *sleeping* mode the underutilized network devices and manage their awakening. The illustrated approach is of distributed type. Every network device performs an algorithm that allows the device to take decisions about its state, checking the network situation and maintaining the traffic demands routing feasible. We consider an OSPF scenario in which we have exploited the SNMP protocol capabilities to obtain desired features.

The results show that energy saving depends on the considered network and its load, allowing at best to achieve a saving higher than 40%.

# 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 [31], 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 [20], 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 del terzo scenario, sviluppando l'algoritmo eseguito dalle singole macchine. L'emulazione in Netkit ci ha permesso di valutare la bontà dell'interazione dell'algoritmo con le modifiche apportate in reti di piccole dimensioni. Il numero di macchine virtuali gestibili con Netkit non è molto elevato, quindi, per una valutazione più obiettiva delle prestazioni dell'algoritmo, abbiamo eseguito delle prove con reti reali, come le reti di backbone di alcuni ISP. I risultati hanno mostrato come, con l'impiego di questa soluzione, sia possibile ottenere un buon risparmio energetico.

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.

Il Capitolo 6 descrive le scelte progettuali e l'implementazione di un algoritmo distribuito ideato per effettuare lo spegnimento di nodi ed interfacce all'interno di reti OSPF. Sono descritti i test effettuati ed i risultati ottenuti per le reti backbone di alcuni ISP.

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

# Capitolo 1

## Tecniche di risparmio energetico

Nel 2003, Gupta e Singh [31] 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 device 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 [30], 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 multiplatforma 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



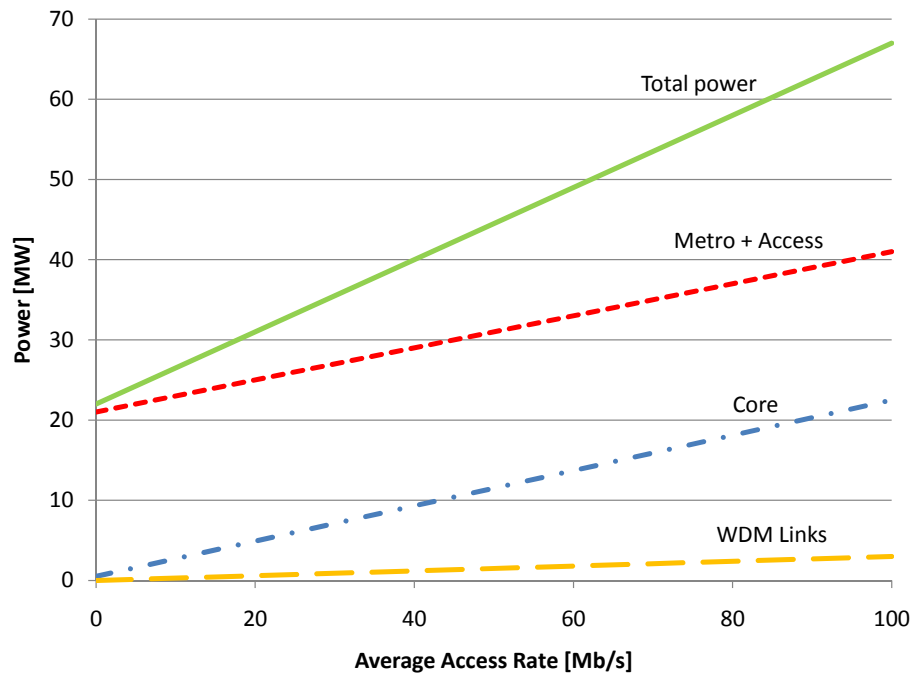
inferiore al totale degli abbonati. Con la recente diffusione di applicazioni quali, per esempio, l'IPTV, il peer-to-peer e lo streaming, tale valore è stato abbassato a causa delle maggiori richieste di banda e dei più stringenti requisiti in termini di QoS.

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 moltiplicazione 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.

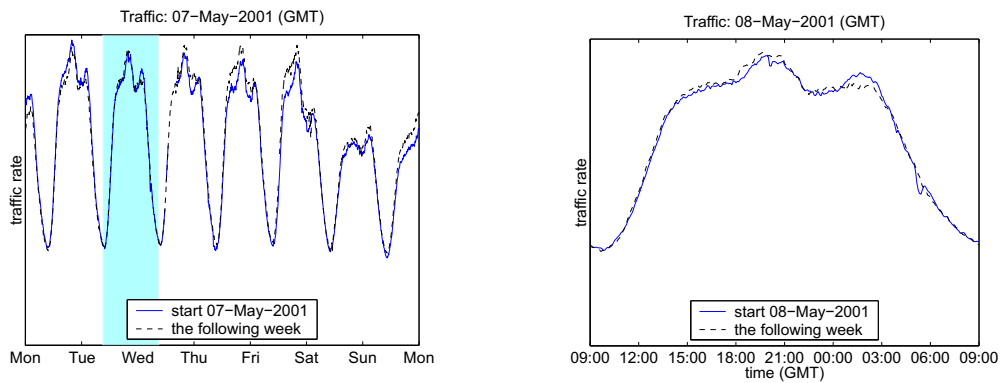
In [36] viene presentato un modello network-based per la misurazione del consumo energetico di Internet. Vengono calcolati i consumi della metro-access network e dei core routers, utilizzando dati relativi ad apparecchiature Cisco, e quelli relativi ai link WDM (Wavelength Division Multiplexing). Il rate di accesso medio è dato da  $R(\rho + 1)$ , dove  $R$  è il picco del rate di accesso e  $\rho$  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 [22] (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 [31], 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 in un intervallo di 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

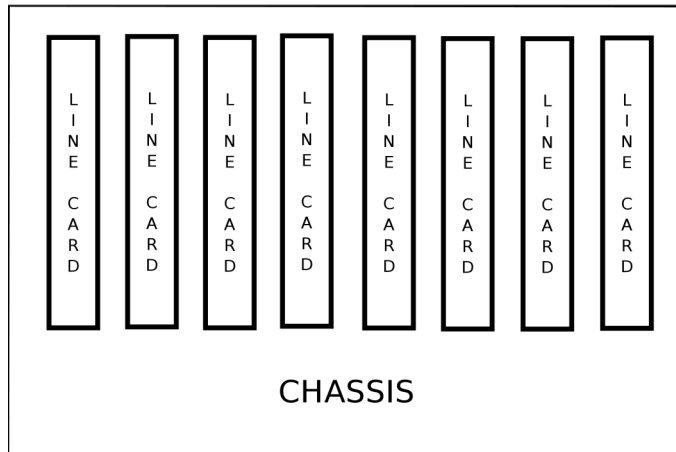
sarebbe opportuno avere una durata del periodo di sleeping sufficientemente lunga da compensarlo. Tale periodo deve anche essere superiore a quello necessario per il risveglio, e la sua massima durata è condizionata dal carico della rete e dalle caratteristiche del protocollo di routing utilizzato.

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 [19], 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 ognuna 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 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).

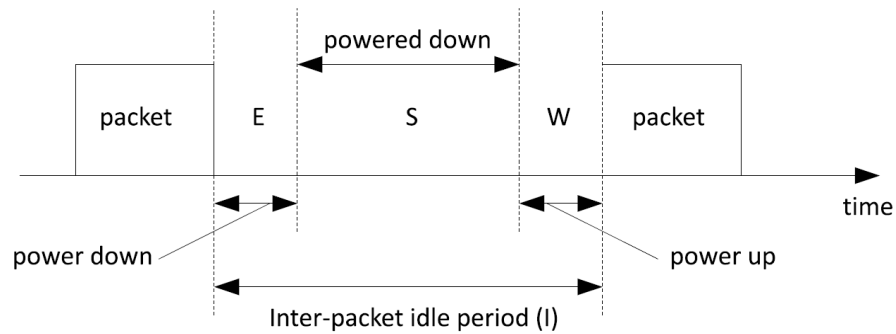
I test sono stati effettuati su una rete da 9 nodi ed una da 20. Ciò che è emerso in



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

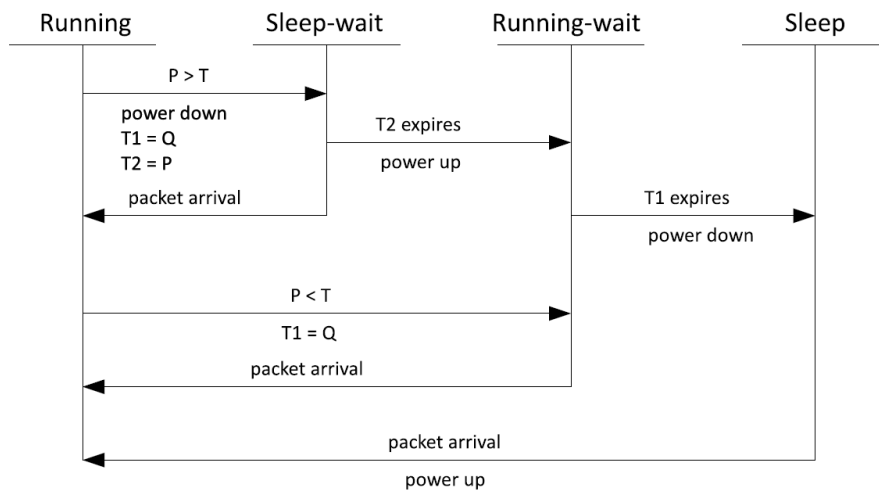
entrambi i casi è che l'utilizzo di un routing fisso limita la capacità del meccanismo di gestione dell'energia di reagire alle variazioni di traffico cambiando lo stato dei router e delle line cards. Vi è comunque un'apprezzabile riduzione dei consumi, che risulta però più marcata con il routing variabile: 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 [38], 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 finché 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 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  $\alpha$  è una costante ( $0 \leq \alpha \leq 1$ ). La procedura completa



**Figura 1.4:** Periodo di inattività tra due pacchetti

è illustrata in Figura 1.5 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). Il valore di  $Q$  è calcolato con un algoritmo specifico [49].



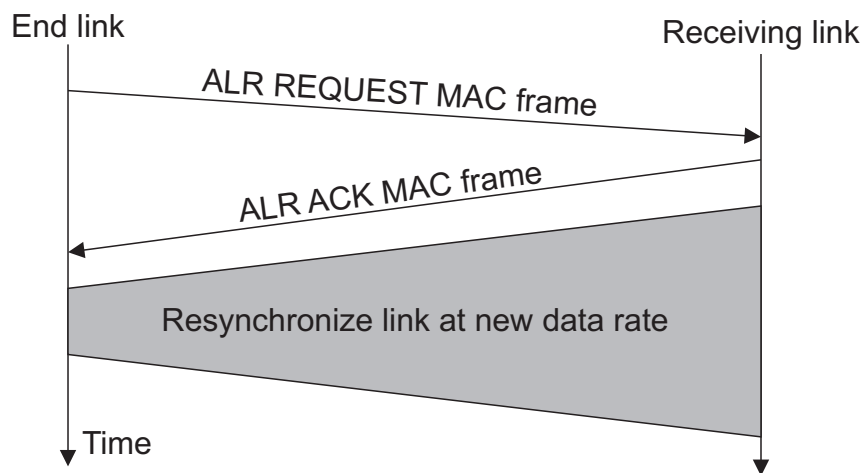
**Figura 1.5:** Schema dell'algoritmo di sleeping predittivo presentato in [38]

In [37] viene proposto un protocollo di routing distribuito per coordinare lo spegnimento dei nodi in una rete, senza degradarne la qualità del servizio. Un router in grado di entrare in uno stato di risparmio energetico ha due stati: *working*, in cui opera come un normale router, e *sleeping*, in cui non processerà nessun pacchetto fino alla sua riaccensione.

Nella rete sono presenti due tipi di router, router normali e router in grado di entrare in sleep. Se un router è sleeping e il carico di rete è basso rimarrà in tale stato e lo stesso accadrà se il carico di rete è alto e si trova nello stato working. Un router rileva il carico di rete misurando l'utilizzo massimo sui suoi link ( $U_{max}$ ); questo valore non rispecchia il reale carico di rete ma è ad esso correlato. Quando in un router attivo  $U_{max}$  scende al di sotto di una determinata soglia, se è possibile mantenere la connettività di rete, invia un messaggio chiedendo il permesso per andare in sleeping al coordinatore (precedentemente eletto tra i router). Se il router riceve una risposta affermativa, ricostruisce ed invia in broadcast una tabella di routing e successivamente entra in sleep per un certo periodo. Al termine di tale periodo si risveglia e si riconnette nuovamente alla rete. A questo punto se  $U_{max}$  del coordinatore supera una certa soglia, il router riceverà il permesso di passare allo stato working, altrimenti ritornerà in sleeping.

### 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 [52], [21] e [47] gli autori si concentrano sulla possibilità di risparmiare considerevoli quantità di energia facendo lavorare i link ad un rate più basso durante i periodi 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:



**Figura 1.6:** Scambio di trame MAC nel meccanismo ALR [47]

- 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 [47] 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.



## Capitolo 2

# Open Shortest Path First (OSPF)

OSPF [44] 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 sì 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.

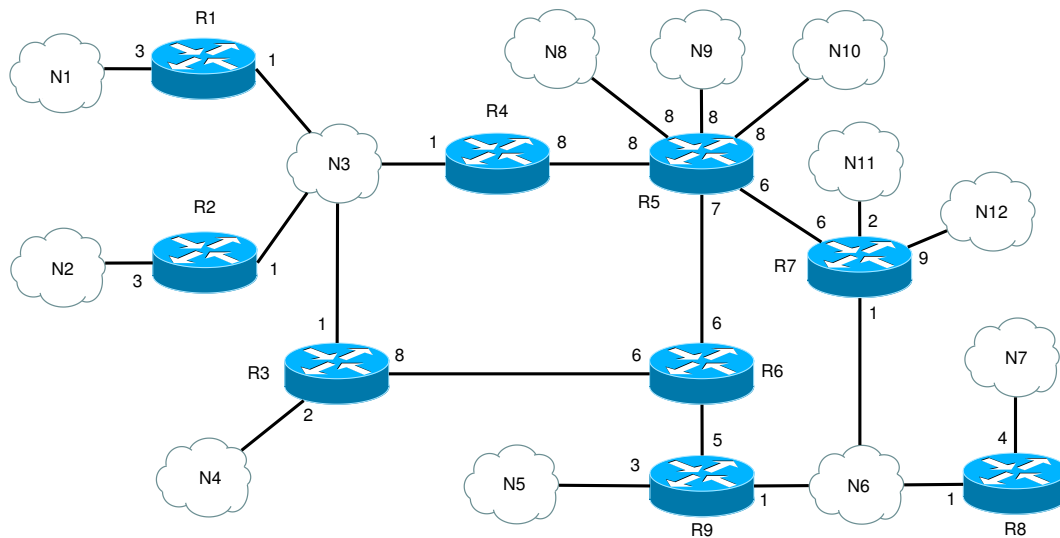
Vediamo ora in modo più dettagliato gli aspetti importanti di questo protocollo:

- 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 della 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 questa 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 [43]: definito per le estensioni multicast di OSPF, non molto usato. Potrà essere riassegnato in futuro;
7. Not So Stubby Area LSA [32]: 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 [45]: 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 [20]: 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.

È 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 un ambiente basato 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 operativo, 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.





# 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 [27]. 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 dei dati.

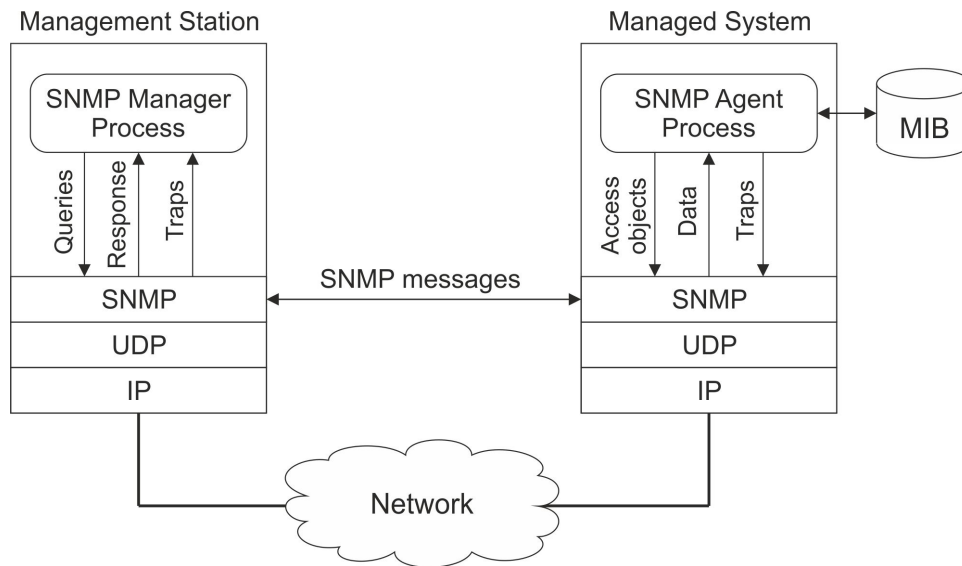
Lo standard finale tuttavia portò alla definizione di **SNMPv2c**, dove la lettera “c”, solitamente omessa, sta per community e di SNMPv2p. Questo sta a significare che il primo standard utilizza lo stesso meccanismo di autenticazione di SNMPv1 mantenendo le nuove funzionalità grazie all’adozione della 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 [28]: un’**entità di gestione**, i **dispositivi da gestire** e un **protocollo di gestione** della rete.

Un sistema SNMP è composto da una o più **Stazioni 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 cosiddette 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 stazione di gestione 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 definito un database detto MIB (Management Information Base), il 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 database 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 [42] 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 [28]. 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.
```

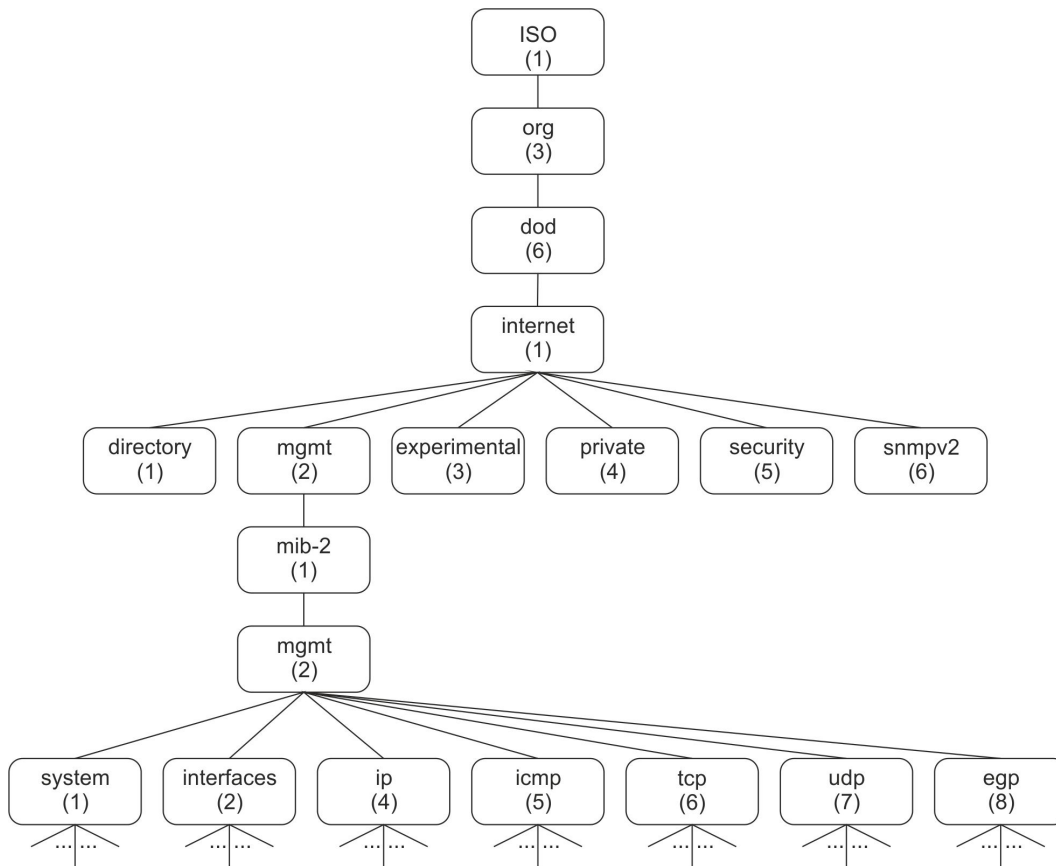


Figura 3.2: Albero delle MIB

### 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'agente 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 sì che il destinatario della notifica invii un messaggio di risposta.

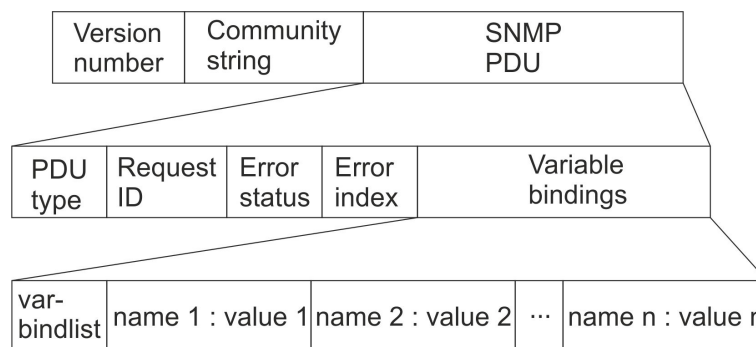
Le varie versioni di SNMP si differenziano anche per il formato di PDU utilizzato. Il formato di una PDU SNMPv2, mostrato in Figura 3.3, è descritto in [48] ed è simile al formato definito in SNMPv1.

- **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 suo 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 [31] 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 basate 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 anziché 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.

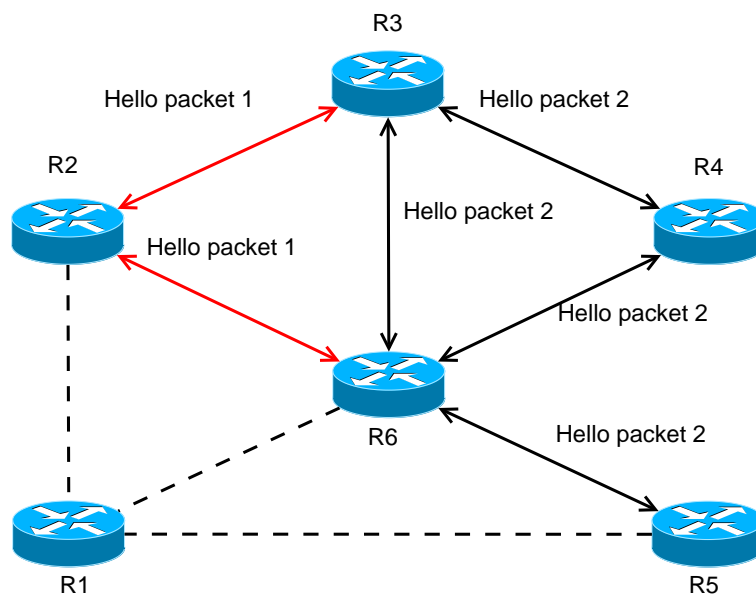
Dalla lettura di questo articolo è nato il nostro interesse verso lo sviluppo di una soluzione al problema del risparmio energetico in reti OSPF.

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, 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 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 [20] 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 un ambiente 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

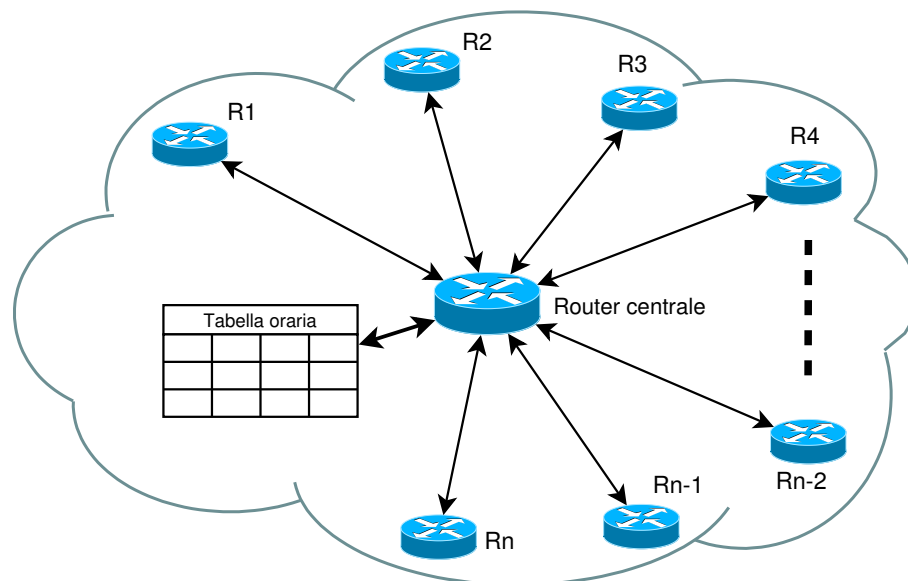
Come già detto, con questo lavoro, si vuole dimostrare che in una rete OSPF è possibile applicare dei cambiamenti per garantire un minore consumo energetico delle apparecchiature di rete. Sono quindi state valutate le diverse possibilità per il

raggiungimento di questo obiettivo attraverso due diversi approcci: soluzione **centralizzata** e soluzione **distribuita**. 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

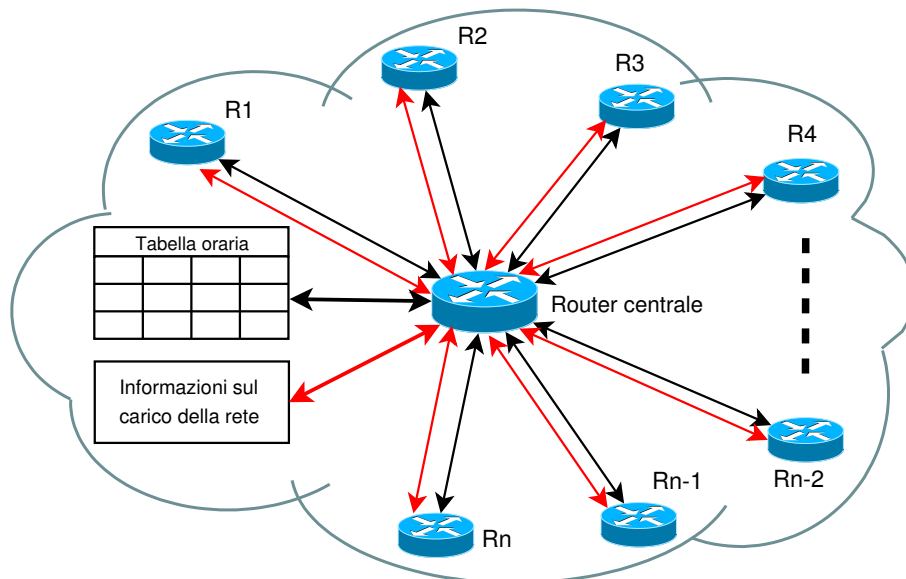
decide quali interfacce di quali router porre in stato di *sleeping*. Ovviamente la tabella oraria sarà costruita in modo di garantire che in orari prestabiliti (ottenuti attraverso osservazioni delle normali condizioni di carico della rete), in cui la rete

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, il software di management Net-SNMP e gli stati delle interfacce.

#### 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 inatten-

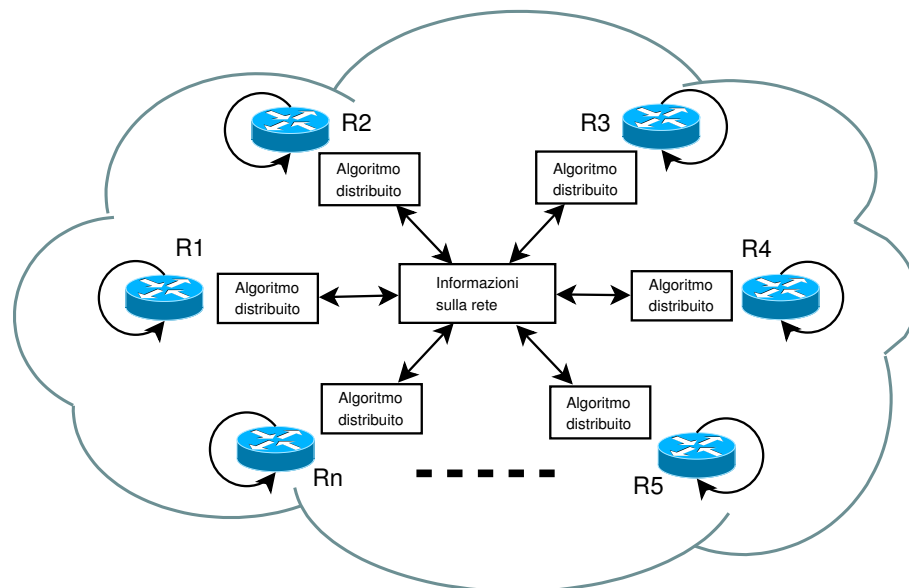


dibile a causa di un aumento di traffico non previsto o di una situazione di guasto. É quindi utile disporre di uno scenario, che permetta ai nodi in stato di *sleeping* di risvegliarsi periodicamente per controllare la situazione della rete attraverso il router centrale. Ciò può essere reso possibile attraverso l'uso di messaggistica SNMP come *snmpset* e *snmptrap*.

Rispetto allo scenario precedente si è introdotta 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

Un approccio di tipo centralizzato prevede l'esistenza di una entità centrale di coordinamento e risulta essere una configurazione piuttosto statica. Si è quindi pensato ad una possibile implementazione distribuita degli stati a basso consumo. 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 ha lo stesso ruolo ed esegue lo stesso algoritmo, convergendo alla stessa soluzione. In questo modo, dopo



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

l'esecuzione dell'algoritmo, ogni elemento di rete sarà a conoscenza della situazione della rete, decidendo indipendentemente se mettersi o meno in stato di *sleeping*.

Il lavoro di tesi qui presentato si focalizza su quest'ultimo approccio presentato. Poichè non era nostra intenzione sviluppare un algoritmo che avesse come punto di forza l'efficienza, ci siamo accontentati di trovare un'euristica che permettesse di valutare la possibilità di spegnere gli elementi di rete, nodi e link, conoscendo alcune informazioni sulla rete e che potesse sfruttare lo stato di basso consumo presentato nel paragrafo 5.4.3, in un ambiente emulato come Netkit.

# 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 [12].

- **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 [14], 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 [50], 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.

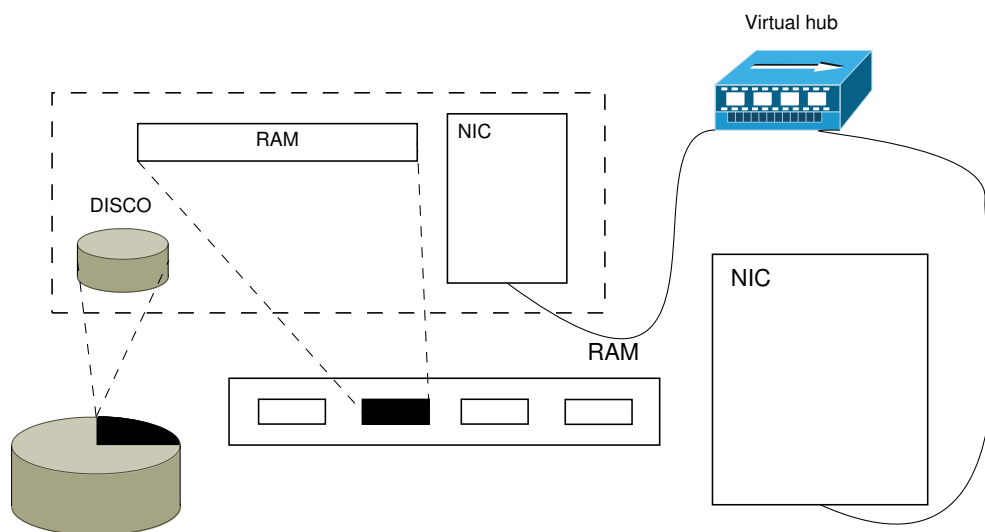
## 5.1 Ambiente di emulazione ed architettura di Netkit

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

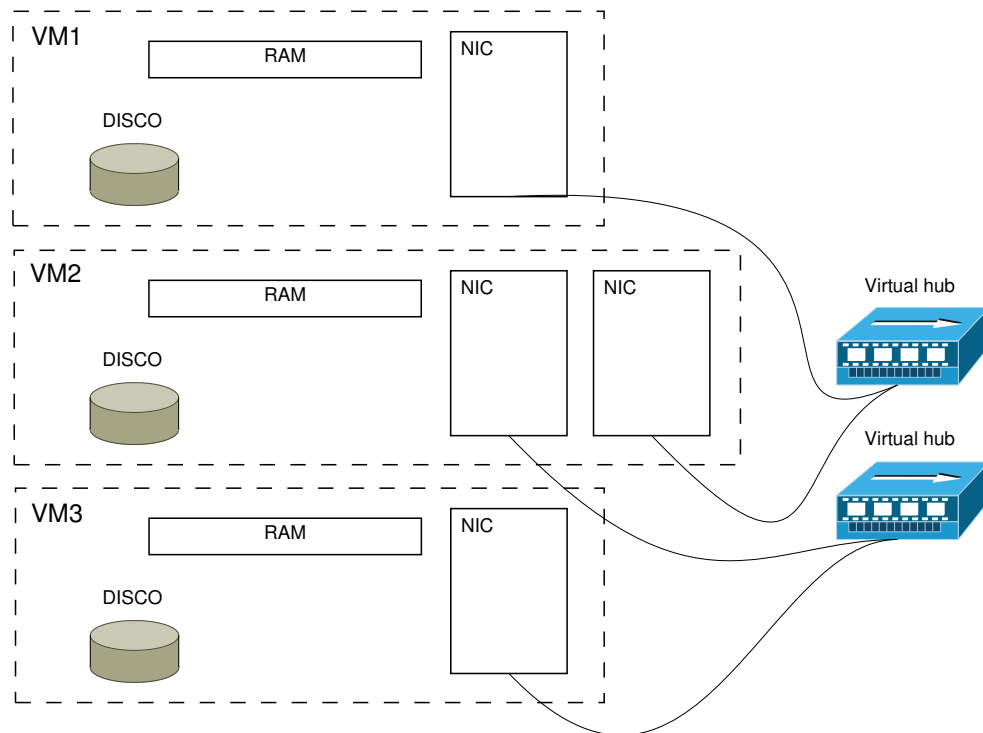


**Figura 5.1:** Le entità racchiuse tra linee tratteggiate sono virtualizzate, mentre tutte le altre corrispondono a dispositivi o processi sulla macchina host.

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



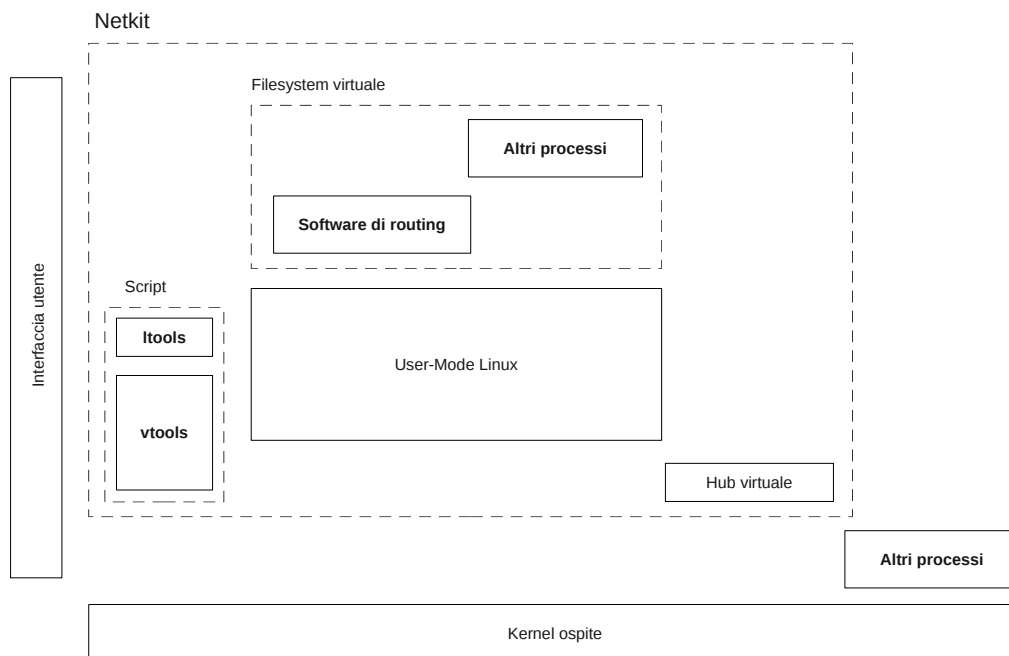
**Figura 5.2:** Esempio di rete emulata con Netkit.

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 [16], 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 [15]. 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 [26], [23].

User-Mode Linux [23], [41], [26], [25], [24], [16] deriva dal Linux kernel standard [11] 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.

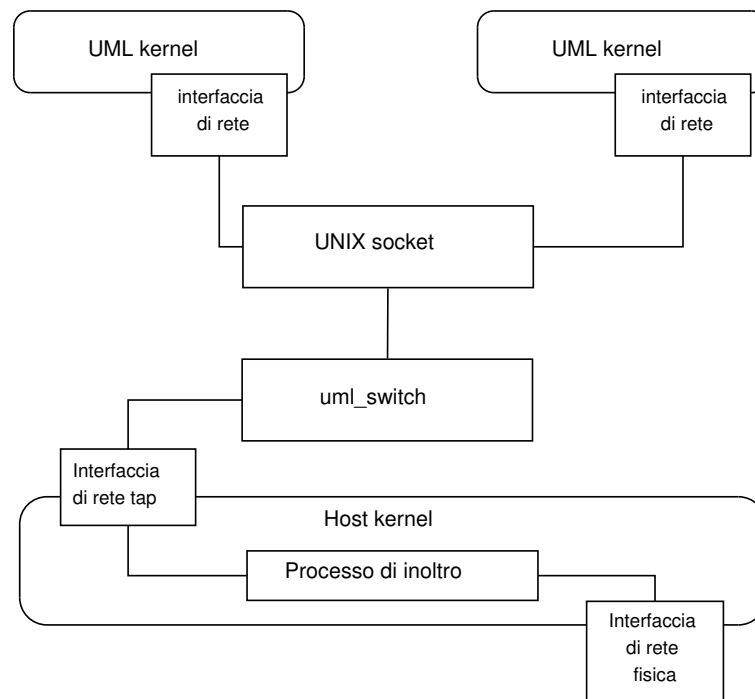
In Netkit si possono implementare configurazioni complesse con molte macchine virtuali. Ogni macchina virtuale scrive sul suo filesystem, il che potenzialmente implica l'utilizzo di molti file di supporto, di dimensioni non trascurabili (centinaia di megabyte). Comunque, è presente la possibilità di 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` [13], [34] 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 [33] 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.

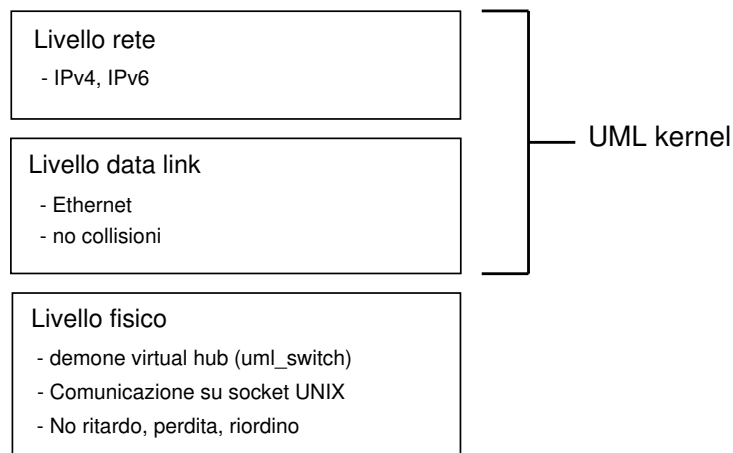
### 5.3 Supporto di rete in Netkit

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` [34], [13] 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 che vi sono connesse. Gli script di Netkit si prendono cura di avviare in automatico



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

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 invia e riceve 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 [29].

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 [10], 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  $\mu$ CLinux, 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 uno scenario di tipo OSPF come quello 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 sopraccitato. 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

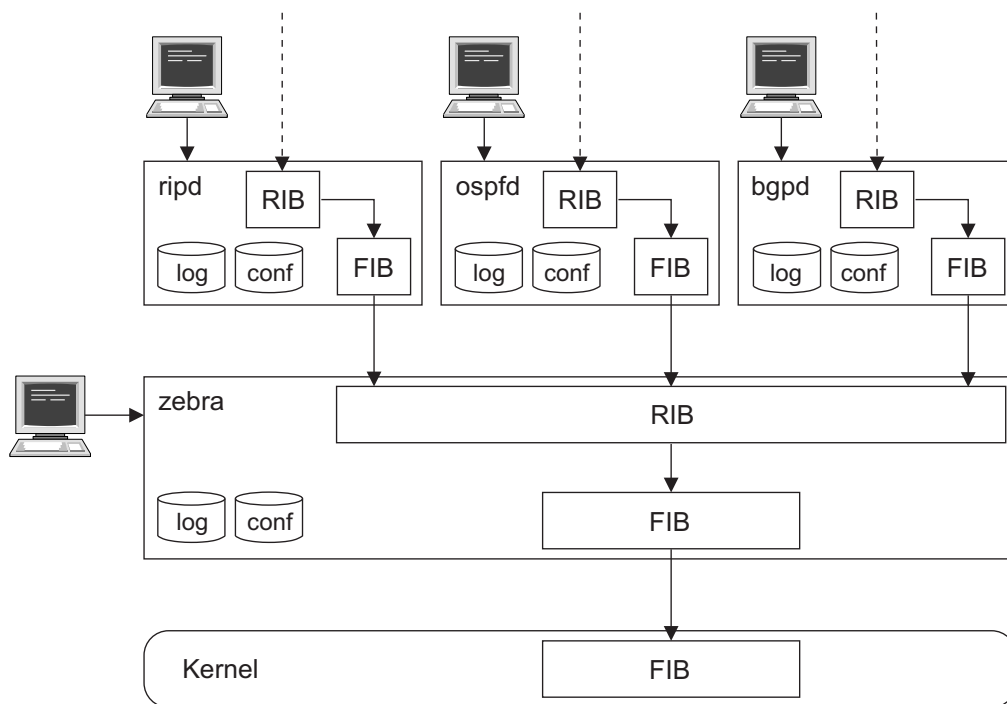
Per poter lavorare con i protocolli di instradamento Netkit include una versione del software di routing Quagga [8], derivante dal più datato progetto GNU Zebra [35] sviluppato da Kunihiro Ishiguro e ormai abbandonato sin dal 2005.

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 [50] 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 dispositivi 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) [51] 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-smnp`. 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
ospfd .1.3.6.1.4.1.3317.1.2.5 .gnomeProducts.zebra.ospfd
```

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 [17].

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 co-

me 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.

## 5.7 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. 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 sull'ultimo scenario applicativo presentato nel paragrafo 4.1. Per poterlo 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.

I linguaggi di programmazione da noi considerati sono due, *bash* e *C*. 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 redirectione 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.

Il motivo principale per cui abbiamo preso in considerazione questo linguaggio di scripting rispetto ad un linguaggio di programmazione vero e proprio, 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. Questa caratteristica è stata utile soprattutto per i primi due scenari.

Tuttavia, bash è mancante di alcune funzionalità presenti in un linguaggio di programmazione classico, come ad esempio la possibilità di gestire array bidimensionali, presente invece nel linguaggio C. Per l'implementazione dell'ultimo scenario presentato, in cui si prevede l'uso di un algoritmo distribuito, è risultata molto più agevole la scrittura di tali codici in C.

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 se 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.

Uno scenario distribuito prevede che ogni nodo sia indipendente dagli altri nel prendere decisioni. Pertanto ognuno di essi deve eseguire le stesse operazioni. Sono stati pubblicati diversi lavori per una gestione dinamica degli elementi di rete, in [39], [40] e [18] vengono presentati algoritmi in grado di realizzarli. Abbiamo quindi pensato di utilizzare un algoritmo che, dati una serie di parametri di ingresso, con una semplice euristica potesse selezionare il maggior numero di elementi di rete da spegnere. Abbiamo identificato nella topologia di rete e nella distribuzione del traffico gli elementi base per poterlo realizzare.

## 6.2 Algoritmo

Per sviluppare il nostro algoritmo ci siamo trovati di fronte ad una scelta fondamentale: creare un algoritmo che, indipendentemente dal nodo che lo esegue, converga sempre allo stesso risultato o creare un algoritmo che in ogni nodo possa convergere a risultati diversi e prevedere poi un meccanismo di coordinamento che selezioni il migliore. La prima soluzione è sicuramente più difficile da realizzare in quanto richie-

derebbe un maggior numero di informazioni iniziali, tuttavia avrebbe il vantaggio di essere molto più veloce e soprattutto eviterebbe la segnalazione necessaria alla selezione di uno dei tanti risultati trovati. La seconda soluzione invece risulterebbe molto più semplice a livello algoritmico ma meno gestibile a livello di segnalazione. Per queste motivazioni si è scelto di seguire un metodo basato sulla prima soluzione.

Abbiamo analizzato diversi algoritmi molto semplici che potevano addattarsi al nostro scopo. Considerando che il consumo di un intero router (chassis e line cards) è molto superiore a quello di una interfaccia di rete, si avrebbe un maggior risparmio energetico nello spegnere interi nodi rispetto ai singoli link. Spegnere interi nodi però è molto più difficile che spegnere singoli link. Seguendo queste assunzioni ci siamo inizialmente focalizzati su un algoritmo che spegnesse selettivamente i nodi. Successivamente lo abbiamo ampliato ed ottimizzato aggiungendo una parte che si occupasse dello spegnimento dei link. Una descrizione semplicistica del problema è la seguente: **dati** i) una topologia di rete (descritta da una matrice delle adiacenze), ii) l'ID univoco di ogni nodo, iii) le domande di traffico tra nodi sorgente/destinazione (descritta da una matrice di traffico), iv) il consumo dei nodi; **trovare** il maggior numero di nodi e link da poter spegnere in modo da ridurre il consumo totale della rete; **garantendo** la connettività tra i nodi sorgente/destinazione e l'instradabilità del traffico rispettando i vincoli di capacità.

Il Codice 6.1 riporta in modo schematico la descrizione delle euristiche da noi proposte, una per l'ottimizzazione dei nodi e una per i link.

**Codice 6.1:** *Pseudo-codice dell'algoritmo proposto*

```
ordinamento_nodi(vettore_nodi)
//spegnimento nodi
for(i=0;i<N;i++) {
    spegnimento_nodo(vettore_nodi[i]);
    calcolo_shortest_path();
    if (controllo_connettività() == true)
        continua;
    else
        break; //riaccensione nodo
    reinstradamento_traffico();
    if (controllo_link() == true)
```

```
        continua;
    else
        break; //riaccensione nodo
}
//spegnimento link
for(i=0;i<L;i++) {
    spegnimento_link(i);
    calcolo_shortest_path();
    if (controllo_connettività() == true)
        continua;
    else
        break; //riaccensione link
    reinstradamento_traffico();
    if (controllo_link() == true)
        continua;
    else
        break; //riaccensione link
}
```

Inizia considerando una rete in cui tutti i suoi elementi sono attivi. I nodi vengono ordinati, secondo uno dei criteri che verranno spiegati in seguito, in base al loro consumo, il quale è assunto proporzionale al numero di interfacce presenti. Per semplicità abbiamo considerato che tutti i router fossero uguali e quindi avessero il medesimo consumo, il problema dell'ordinamento si riduce quindi al calcolo del grado. Se sono presenti più nodi di pari grado il discriminante risulterà l'ID, nodi con ID più alto avranno la precedenza su nodi con ID inferiore. L'algoritmo a questo punto selezionerà iterativamente un nodo da spegnere seguendo l'ordinamento. Successivamente ad ogni passo si ricalcola la nuova topologia senza il nodo selezionato. Si verifica che questo nodo non sia sorgente/destinazione di traffico e si ricalcolano tutti i percorsi utilizzando l'algoritmo di Dijkstra. Se la connettività tra tutte le coppie di nodi sorgente/destinazione è rispettata si procede a reinstradare sui nuovi percorsi la matrice di traffico. Se non vengono violati i vincoli di capacità il nodo selezionato viene spento. Una volta scanditi tutti i nodi si passa ad un eventuale spegnimento dei singoli link rieseguendo gli stessi passi sopra citati per verificare la fattibilità dell'instradamento del traffico anche senza questo elemento di rete.

L'algoritmo proposto è quindi suddiviso in due parti, un primo blocco che si

occupa dello spegnimento dei nodi ed un secondo blocco che si occupa di un eventuale spegnimento dei link.

Sono state pensate diverse politiche che permettessero l'ordinamento dei nodi. Nelle nostre prove abbiamo implementato un ordinamento casuale, un ordinamento crescente in base al grado ed uno decrescente. Con il primo ordinamento selezioniamo i nodi casualmente senza seguire nessun criterio. Con un ordinamento crescente facciamo sì che l'algoritmo provi a spegnere prima i nodi con pochi link, quindi quelli che consumano meno in termini di potenza. Con un ordinamento decrescente invece diamo la precedenza ai nodi con più link e quindi con un maggior consumo.

### 6.3 Implementazione

L'esecuzione di questo algoritmo da parte di ogni nodo della rete fa sì che ognuno di essi converga alla stessa soluzione e possa quindi adottare le procedure necessarie all'eventuale spegnimento. Implementare questa soluzione in una rete reale richiede che ogni nodo conosca l'attuale topologia di rete completa e le domande di traffico presenti tra i nodi. Queste informazioni sono misurabili all'interno della rete e devono quindi essere distribuite ai nodi o risiedere in un elemento di rete che sia accessibile a tutti come il DR in OSPF. Oltre a ridurre il numero di aggiornamenti inviati attraverso la rete, questo processo assicura che tutti i router ricevano le stesse informazioni nello stesso momento e da una singola fonte. Per questo motivo e per le conseguenze che si avrebbero sul protocollo OSPF è bene che il DR non venga mai posto in sleep.

Poiché il traffico in rete non è statico ma variabile è necessario che la rete possa adattarsi ad esso. Se ad esempio dovesse aumentare il traffico si dovrà reagire riaccendendo dei nodi. Tuttavia, soltanto un nodo attivo può accorgersi del cambiamento e rieseguire l'algoritmo per trovare una nuova soluzione di rete che si adatti alle mutate condizioni. Inoltre, sarebbe opportuno mantenere il più possibile inalterata la topologia attuale, aggiungendovi quindi nodi senza ricalcolarne una

nuova. Questo potrebbe ridurre il massimo risparmio energetico, ma eviterebbe di stravolgere la rete e di caricarla con un eccessivo traffico di segnalazione.

Riaccendere dei nodi risulta essere particolarmente problematico, in quanto l'obiettivo è creare nuovi percorsi tra i nodi sorgente/destinazione di traffico, e la riaccensione di singoli nodi senza un criterio opportuno non lo permetterebbe. A livello algoritmico, per gestire la riaccensione dei nodi basterebbe rieseguire gli stessi passi fatti per lo spegnimento vincolando però i nodi attivi a non poter essere spenti. In questo modo otteniamo una nuova topologia che permette di instradare il nuovo traffico e mantiene attivi i nodi che già lo erano.

Implementare quanto appena descritto in una rete reale non è semplice. Ogni nodo sleep non è a conoscenza della situazione attuale e non può eseguire l'algoritmo con i nuovi parametri. Stando a queste assunzioni saranno i nodi attivi che, in seguito ad un ricalcolo della topologia, dovranno attuare una procedura in grado di risvegliare i nodi. Poiché abbiamo detto che ogni nodo è a conoscenza della topologia completa, sarà anche in grado di determinare se è connesso ad uno dei nodi da risvegliare e tramite tecniche quali WOL può agire di conseguenza. Il discorso è leggermente diverso se invece assumiamo che oltre ai nodi completi possano essere spenti anche i singoli collegamenti: in questo caso il nodo di interesse sarà attivo su altre interfacce e potrà agire localmente per risvegliare l'interfaccia richiesta.

Se un nodo, durante il suo periodo di sleep, dovesse per qualche motivo guastarsi, è necessario che gli altri nodi della rete ne vengano a conoscenza. Questo perché il nodo in questione deve essere escluso da un eventuale ricalcolo della topologia, in quanto non più utilizzabile. È necessario dunque che ogni nodo in sleep informi la rete sul suo attuale stato, così come viene fatto normalmente in OSPF tramite i messaggi di Hello. Ogni nodo sleep potrebbe risvegliarsi periodicamente e scambiare messaggi simili agli Hello, con tempistiche superiori ai normali Hello packet in modo da non appesantire il protocollo: i suoi vicini sarebbero così informati della sua presenza e del suo stato di sleep. In questo modo si può mantenere aggiornato un database contenente lo stato di tutti i nodi nella rete. Se entro un determinato

intervallo di tempo non si riceve una comunicazione da un nodo si ritiene che questo sia guasto, così come avviene normalmente in OSPF attraverso il router dead interval.

L'algoritmo, con il nuovo stato di sleep introdotto, è stato implementato tramite Netkit in una rete OSPF monoarea di dieci nodi. Abbiamo applicato a tale rete una matrice di traffico statica, considerando il traffico costante tra le coppie di nodi per verificarne il funzionamento. Le informazioni riguardo la topologia di rete, per semplicità, sono state preconfigurate in ogni nodo. Inoltre, poiché in Netkit non è possibile simulare un guasto ad un link non abbiamo implementato lo scambio di messaggi riguardo lo stato dei nodi, considerandoli quindi nei due soli stati possibili, up e sleep. Una rete di dimensioni ridotte come quella provata non ci permetteva di valutare le prestazioni dell'algoritmo in termini di efficienza. Non potendo implementare in Netkit reti di dimensioni maggiori, a causa della complessità che ne deriva, abbiamo preferito testare il solo algoritmo con reti reali e matrici di traffico diverse. Nella sezione seguente abbiamo riportato le topologie di rete utilizzate e il modello di generazione delle matrici di traffico.

## 6.4 Generazione dei dati di test

### 6.4.1 Topologie di rete

In ambito "green", per i ricercatori è molto importante avere a che fare con topologie di rete reali. Queste infatti influenzano le prestazioni dei vari approcci sul risparmio energetico. Purtroppo le topologie di rete degli Internet Service Provider non sono comunemente disponibili, poiché tali informazioni sono considerate strettamente confidenziali. In [46] vengono presentate nuove tecniche adottate dai ricercatori, in grado di dedurre le mappe di alcuni ISP. Essi sono stati in grado di mappare dieci diversi ISP – Abovenet, AT&T, Ebone, Exodus, Level3, Sprint, Telstra, Tiscali (Europe), Verio e VSNL (India) – sfruttando oltre 750 fonti di traffico catturato a disposizione del pubblico. In Tabella 6.1 vengono riportate le caratteristiche di sei

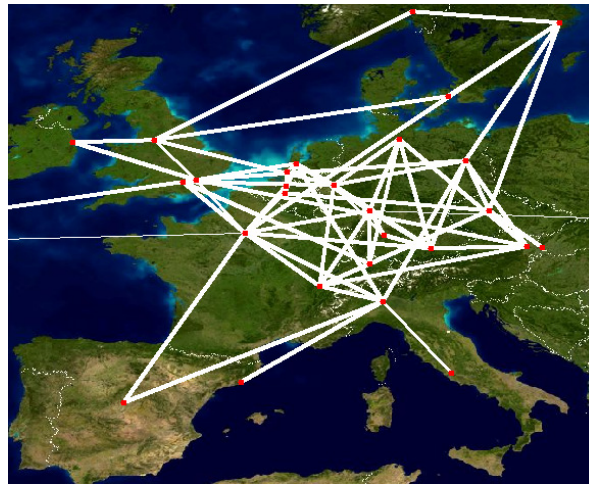


reti di backbone tra quelle precedentemente elencate; i documenti relativi a tali mappe sono disponibili presso [9]. Queste mappe includono sia i nodi di accesso che di backbone. Tre dei dieci provider sopra elencati hanno confrontato le mappe ottenute dai ricercatori con la topologia reale, confermando la loro estrema precisione.

I test sono stati effettuati su tre delle sei topologie di rete riportate nella Tabella 6.1. In particolare si è lavorato con le reti Ebone, Exodus e Telstra, le cui topologie sono mostrate in Figura 6.1. Purtroppo, anche se le topologie di rete sono note, non esistono informazioni precise circa la capacità dei link delle reti. Abbiamo quindi assunto, per ogni rete, che i link avessero tutti la stessa capacità scelta in modo arbitrario. Infine, abbiamo avuto la necessità di selezionare un insieme di nodi edge e core. I nodi edge potrebbero essere sia di origine che di destinazione per le domande di traffico. I nodi core, invece, sono di transito, quindi sono gli unici che possono essere spenti. Nelle topologie usate non abbiamo nessuna informazione circa lo stato del nodo (core/edge): abbiamo quindi scelto a caso una serie di nodi edge per ciascuna delle tre topologie considerate (Tabella 6.2). Ovviamente abbiamo considerato tutti i nodi foglia come nodi di edge. In questo modo si evitano i casi banali in cui tutti i nodi foglia di core potrebbero essere facilmente spenti: una simile situazione non sarebbe realistica. Abbiamo inoltre deciso di selezionare almeno un nodo edge per ogni singola città.

Network	AS number	Routers	Links	Cities	Inter-city Links
Telstra (Australia)	1221	104	302	53	55
Sprintlink (US)	1239	315	1944	44	83
Ebone (Europe)	1755	87	322	23	38
Tiscali (Europe)	3257	161	656	50	88
Exodus (US)	3967	79	294	22	37
Abovenet (US)	6461	141	748	22	42

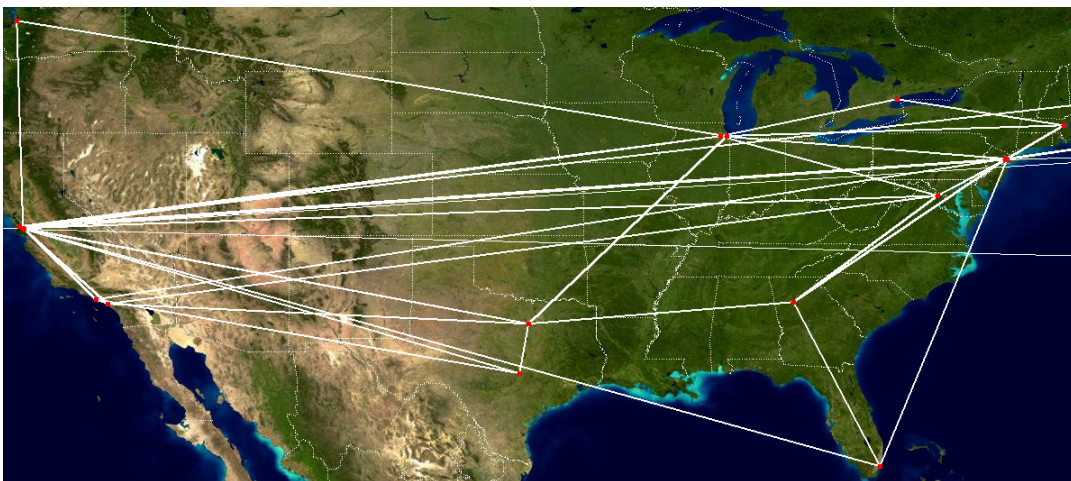
**Tabella 6.1:** *Topologie di backbone del progetto Rocketfuel. La Tabella mostra il numero di link e router per ogni topologia. Vengono riportate anche alcune informazioni sulle città per dare un'idea delle reali dimensioni delle reti considerate.*



(a) Ebone



(b) Telstra



(c) Exodus

**Figura 6.1:** Rappresentazione schematica delle topologie del backbone di rete. Immagini di sfondo fornite da Visible Earth, NASA.

Network	Total Routers	Edge Routers	Core Routers	% Edge Routers
Telstra (Australia)	104	65	39	62.5
Ebone (Europe)	87	31	56	35.6
Exodus (US)	79	38	41	48.1

**Tabella 6.2:** Suddivisione tra router di edge e di core considerata nelle prove. La percentuale di router di edge nella topologia Telstra risulta molto alta perché sono presenti 51 nodi foglia.

### 6.4.2 Generazione delle matrici di traffico

Come nel caso di topologie di reti reali, anche le informazioni relative alle matrici di traffico non sono facilmente reperibili. A causa dell'assenza di dati abbiamo deciso di usare una formulazione in programmazione lineare, in modo da ottenere, per ognuna delle tre topologie, una matrice di massimo traffico sostenibile. Per poterla instradare occorre che tutti i collegamenti di rete siano attivi. La formulazione utilizzata è illustrata di seguito.

*Funzione obiettivo:*

$$\max \sum_{i=1}^N \sum_{j=1}^N M * t_{ij} + a_{ij} \quad (6.1)$$

*Vincoli:*

$$\sum_{s=1}^N f_{ijs} \leq c_{ij} * e_{ij} \quad \forall i, j \in V \quad (6.2)$$

$$\sum_{j=1}^N f_{iji} = \sum_{d=1}^N t_{id} + a_{id} \quad \forall i \in V \quad (6.3)$$

$$\sum_{j=1}^N f_{jis} - \sum_{j=1}^N f_{ijs} = t_{si} + a_{si} \quad \forall i, s \in V : s \neq i \quad (6.4)$$

$$t_{ii} + a_{ii} = 0 \quad \forall i \in V \quad (6.5)$$

$$t_{ij} \leq \sum_{h=1}^N \frac{e_{ih} * c_{ih}}{N - \alpha} \quad \forall i, j \in V \quad (6.6)$$

$$t_{ij} \geq \beta \quad \forall i, j \in V : j \neq i \quad (6.7)$$

$$t_{ij}, a_{ij} \geq 0 \quad \forall i, j \in V, \quad f_{ijs} \geq 0 \quad \forall i, j, s \in V \quad (6.8)$$

$$t_{ij} + a_{ij} \leq M * G_i G_j \quad \forall i, j \in V \quad (6.9)$$

L'equazione (6.8) esprime le variabili: siano  $t_{ij}, a_{ij}$  variabili reali positive che rappresentano le domande di traffico tra il nodo  $i$  e il nodo  $j$  ( $t_{ij} + a_{ij}$  indica le totali domande di traffico tra  $i$  e  $j$ ), e siano  $f_{ijs}$  le variabili di flusso che mostrano il totale traffico instradato attraverso il collegamento  $(i, j)$  ed originato dal nodo  $s$ . Usiamo due variabili (la principale  $t_{ij}$  e l'ausiliaria  $a_{ij}$ ) per rappresentare le domande di traffico ed evitare soluzioni banali in cui il traffico tra due nodi coincida con la capacità. Introduciamo perciò i vincoli (6.6) e (6.7), che pongono alcune limitazioni ai valori accettabili di  $t_{ij}$ ; in particolare l'equazione (6.6) e l'equazione (6.7) definiscono rispettivamente il massimo ed il minimo valore tollerato per ogni domanda di traffico. I parametri  $\alpha$  e  $\beta$  possono essere variati per ottenere matrici più o meno bilanciate. Nella funzione obiettivo (6.1),  $t_{ij}$  è moltiplicato per  $M$  (parametro di valore molto grande) ed ha quindi un peso maggiore di  $a_{ij}$ ; in questo modo si cerca di massimizzare i valori di  $t_{ij}$ , rispettando i vincoli (6.6) e (6.7). L'uso della variabile ausiliaria  $a_{ij}$  nasce dal fatto che i vincoli (6.6) e (6.7) rendono quasi impossibile, per la maggior parte delle topologie, ottenere una matrice di traffico completa con i soli valori ammissibili di  $t_{ij}$ . L'equazione (6.2) è il vincolo di capacità dei link, mentre (6.3) e (6.4) sono i consueti vincoli di conservazione di flusso. L'equazione (6.5) permette solo le domande di traffico tra nodi distinti. Ovviamente è necessario sommare  $t_{ij}$  e  $a_{ij}$  per ottenere la matrice di traffico finale. Per garantire l'assenza di domande di traffico tra nodi core introduciamo l'equazione (6.9), che sfrutta la tecnica del big-M. Il parametro binario  $G_i$  è uguale ad 1 quando il nodo  $i$  è un nodo edge.

Le matrici di traffico ottenute con il metodo appena descritto possono essere instradate senza eccedere la capacità dei link solamente se si utilizza un instradamento libero, con la possibilità di dividere i flussi multicommodity. Il protocollo OSPF potrebbe non essere in grado di gestire lo stesso instradamento.

Le matrici di traffico massimo ottenute non sono in grado di caricare completamente la rete. Ciò non è dovuto alla formulazione ma ad altri due elementi: la struttura della topologia e l'impossibilità di generare domande di traffico in ogni no-

do. Abbiamo assunto che le matrici per basso carico di traffico fossero proporzionali a quella ottenuta per il pieno carico: le abbiamo pertanto ricavate moltiplicando la matrice originale per un parametro compreso tra 0 e 1.

### 6.4.3 Consumi

Per misurare con dei valori concreti l'effettivo risparmio energetico, sono stati sfruttati dati relativi ai consumi di dispositivi di rete reali. In particolare, sono stati considerati i dati di consumo riportati da Juniper riguardo il router M10i [1]. Nella Tabella 6.3 abbiamo riportato il consumo in watt del solo chassis, mentre nella Tabella 6.4 ci sono 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 6.3:** La tabella descrive le caratteristiche principali del router M10i, il dato di maggior interesse è il consumo in potenza.

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 6.4:** Line cards per il router M10i

## 6.5 Risultati computazionali

Per valutare le performance dell'algoritmo proposto in condizioni reali lo abbiamo testato su tre topologie di rete di backbone. Non potendo implementare queste reti

in Netkit a causa delle loro eccessive dimensioni ed alla mancanza di alcune informazioni su di esse abbiamo preferito valutare semplicemente quale fosse il possibile risparmio in termini di energia applicando l'algoritmo a tali topologie di rete.

Abbiamo applicato l'algoritmo ad ognuna delle tre reti in situazioni di carico diverse e con i criteri di ordinamento presentati nel paragrafo 6.2. Per ogni prova abbiamo raccolto i risultati forniti dall'esecuzione dell'algoritmo verificando quale fosse il numero di nodi e link da mantenere accesi affinché si potesse instradare il traffico. Abbiamo poi calcolato i valori in termini di potenza risparmiata, assumendo che i nodi fossero tutti uguali fra loro, usando i parametri del router M10i.

Le Tabelle 6.5, 6.6 e 6.7, riportano i risultati ottenuti. Nella prima colonna viene riportato il carico della rete: un valore del 10%, ad esempio, rappresenta la rete in cui la matrice di traffico instradata è stata ottenuta moltiplicando per 0,1 la matrice completa. La seconda colonna riporta il tipo di ordinamento applicato ai nodi. Le colonne *nodi off* e *link off* riportano rispettivamente il numero di nodi spenti con la relativa percentuale rispetto al totale dei nodi presenti nella rete ed il numero totale di link spenti.

Le matrici di traffico utilizzate per testare l'algoritmo erano state pensate per caricare la rete senza tener conto di una effettiva instradabilità con protocolli come OSPF, il quale prevede un solo percorso tra i nodi sorgente e destinazione. Per questo motivo nella rete Exodus abbiamo potuto testare solo un carico di rete del 10% e 20%, poichè per valori superiori le matrici di traffico non erano instradabili. Questo rende i suoi risultati poco interessanti.

Analizzando i risultati si osserva subito il divario in performance dell'algoritmo applicato alla rete Ebone rispetto alla rete Telstra. Con un carico della rete pari al 10% infatti si passa da un 42% di nodi spenti ad un 17%. Questo è giustificato dalla struttura della rete Telstra che, nonostante abbia un maggior numero di nodi, ha, in media, un numero inferiore di link per nodo rispetto alla rete Ebone. Inoltre, sono presenti ben 51 nodi foglia che abbiamo dovuto classificare come nodi edge. Questo fa sì che il numero di nodi core che possono essere spenti in questa rete si

Ebone: 322 link, 87 nodi - 56 nodi core, 31 nodi edge				
Carico (%)	Ordinamento	Nodi off (#)	(%)	Link off (#)
10%	Crescente	37	42,53	224
10%	Decrescente	30	34,48	210
10%	Casuale	31	35,63	212
20%	Crescente	37	42,53	220
20%	Decrescente	31	35,63	208
20%	Casuale	30	34,48	206
30%	Crescente	35	40,23	208
30%	Decrescente	29	33,33	194
30%	Casuale	28	32,18	192
40%	Crescente	30	34,48	190
40%	Decrescente	31	35,63	192
40%	Casuale	30	34,48	192
50%	Crescente	27	31,03	176
50%	Decrescente	26	29,89	172
50%	Casuale	27	31,03	176

**Tabella 6.5:** In questa tabella sono riportati i risultati dei test effettuati sulla rete Ebone in funzione di diversi carichi di rete

Exodus: 294 link, 79 nodi - 41 nodi core, 38 nodi edge				
Carico (%)	Ordinamento	Nodi off (#)	(%)	Link off (#)
10%	Crescente	37	46,84	208
10%	Decrescente	37	46,84	208
10%	Casuale	37	46,84	208
20%	Crescente	20	25,32	148
20%	Decrescente	20	25,32	148
20%	Casuale	20	25,32	148

**Tabella 6.6:** In questa tabella sono riportati i risultati dei test effettuati sulla rete Exodus in funzione di diversi carichi di rete

riduca molto. Osservando i risultati ottenuti per la rete Ebone si nota come per valori bassi di traffico il risparmio in termini di elementi di rete spenti è elevato. Siamo infatti intorno al 40% dei nodi, valore che tende a diminuire sensibilmente all'aumentare del carico di rete. Questa rete con ben 56 nodi core e i suoi 322 link risulta essere molto magliata, il che permette diversi gradi di libertà nella scelta dei possibili percorsi. Nelle reti Exodus e Telstra, invece, il risparmio è inferiore.

Osservando i risultati ottenuti in funzione dei diversi criteri di ordinamento usati nell'algoritmo, si nota che nelle reti Telstra ed Exodus l'esito è pressoché il mede-

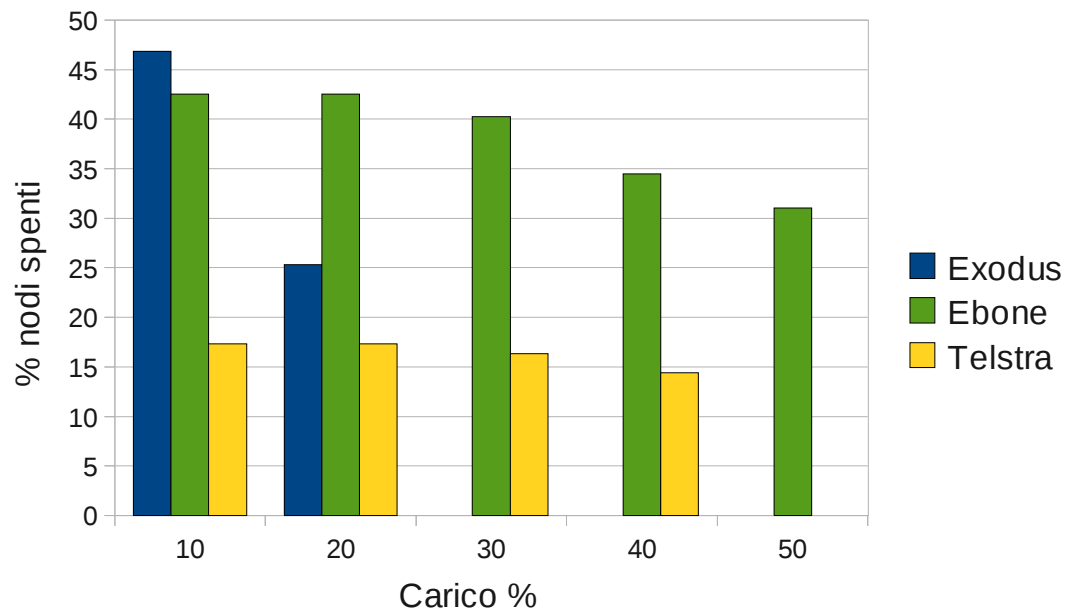
Telstra: 302 link, 104 nodi - 39 nodi core, 65 nodi edge				
Carico (%)	Ordinamento	Nodi off (#)	(%)	Link off (#)
10%	Crescente	18	17,31	132
10%	Decrescente	18	17,31	132
10%	Casuale	18	17,31	132
20%	Crescente	18	17,31	132
20%	Decrescente	18	17,31	132
20%	Casuale	18	17,31	132
30%	Crescente	17	16,35	128
30%	Decrescente	17	16,35	128
30%	Casuale	16	15,38	126
40%	Crescente	15	14,42	118
40%	Decrescente	14	13,46	118
40%	Casuale	14	13,46	116

**Tabella 6.7:** In questa tabella sono riportati i risultati dei test effettuati sulla rete Telstra in funzione di diversi carichi di rete

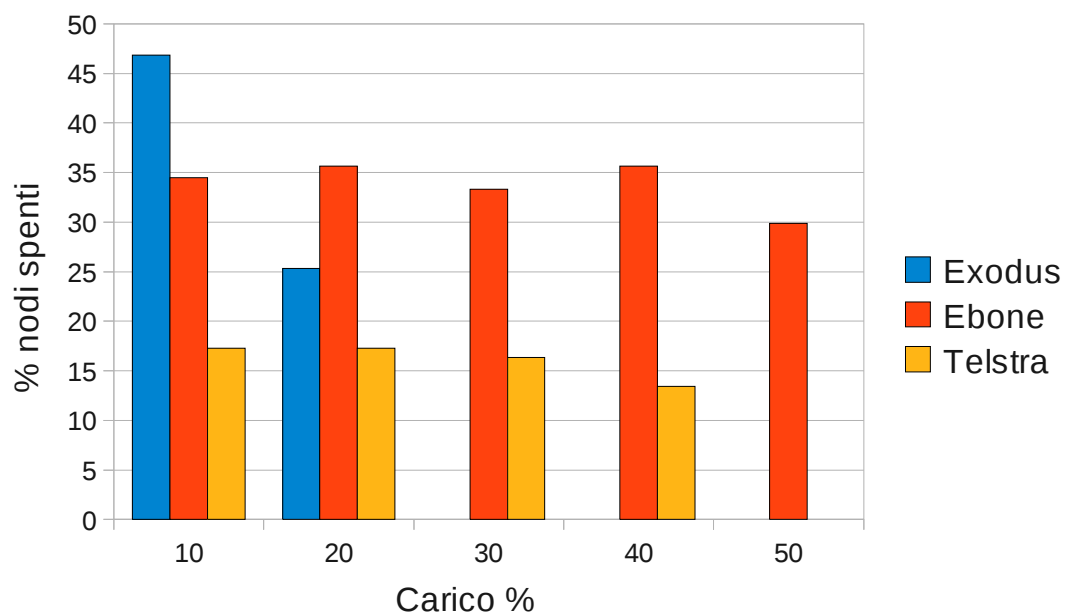
simo. Osservando, invece, i risultati ottenuti con la rete Ebone si nota come un ordinamento dei nodi di tipo crescente, che prova a spegnere prima i nodi con un minor numero di link, dia risultati migliori. Questo potrebbe essere un controsenso al nostro obiettivo, cioè massimizzare il risparmio energetico, che farebbe pensare che sia meglio spegnere prima i nodi che consumano di più. Tuttavia, quanto descritto sopra era intuibile, infatti, è molto più probabile che si riesca a spegnere un nodo con pochi link, poiché su di esso passerà in media meno traffico, rispetto ad un nodo con molti link.

Riportiamo di seguito dei grafici comparativi dei risultati ottenuti. Ogni grafico mostra, per ogni tipo di ordinamento eseguito, la percentuale dei nodi spenti rispetto al totale dei nodi di cui sono formate le reti, in funzione dei diversi carichi. In Figura 6.2 abbiamo riportato i risultati ottenuti nelle tre reti applicando un ordinamento dei nodi di tipo crescente. In Figura 6.3, invece, abbiamo riportato i risultati ottenuti con un ordinamento di tipo decrescente. Infine, in Figura 6.4 i risultati adottando un ordinamento casuale. Dai grafici si evidenzia quanto descritto precedentemente, in particolare si osserva come un ordinamento crescente dia i risultati migliori.



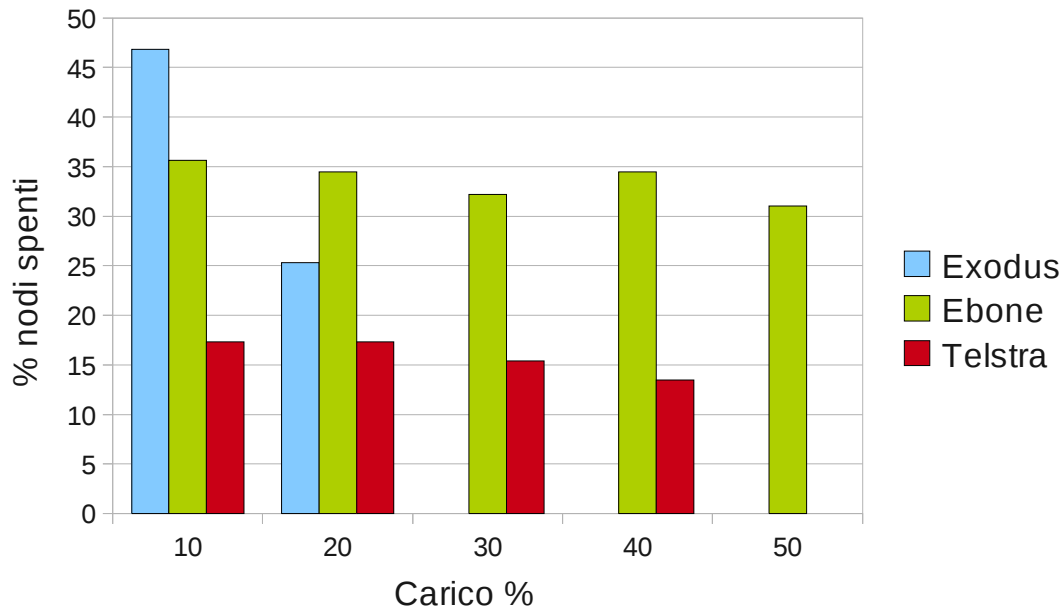


**Figura 6.2:** Il grafico riporta la percentuale dei nodi spenti dall' algoritmo nelle reti Ebone, Exodus e Telstra, al variare del carico di rete, adottando un ordinamento crescente.



**Figura 6.3:** Il grafico riporta la percentuale dei nodi spenti dall' algoritmo nelle reti Ebone, Exodus e Telstra, al variare del carico di rete, adottando un ordinamento decrescente.

Nelle Tabelle 6.8, 6.9 e 6.10, abbiamo riportato dei valori numerici dei possibili risparmi, in termini di potenza, usando i consumi di un router reale (vedi paragrafo 6.4.3). Nell'intestazione delle tabelle si trova il consumo con tutti gli elementi



**Figura 6.4:** Il grafico riporta la percentuale dei nodi spenti dall' algoritmo nelle reti Ebone, Exodus e Telstra, al variare del carico di rete, adottando un ordinamento casuale.

di rete attivi, mentre nella colonna *Risparmio energia* si può vedere la potenza in watt risparmiata (chassis e line cards,  $(86,4 W + n \times 7,3 W)$ ), riferita ai casi di ordinamento crescente dei nodi precedentemente descritti. Questi valori dipendono dal tipo di line card utilizzata, i valori qui presentati sono riferiti a schede Gigabit Ethernet aventi ciascuna una sola porta, quindi una scheda per ogni link. Nell'ultima colonna abbiamo infine riportato il valore percentuale del risparmio energetico che si avrebbe nella rete calcolato come  $(P - P_{off}) / P \times 100$ , dove  $P$  e  $P_{off}$  sono rispettivamente il consumo della rete con tutti gli elementi attivi e il consumo con i nodi in sleep.

Ebone: 322 link, 87 nodi - consumo totale=10126.6 W				
Carico	Nodi off (#)	Link off (#)	Risparmio energia (W)	Risparmio (%)
10%	37	224	4832	47,7
20%	37	220	4802,8	47,4
30%	35	208	4542,4	44,8
40%	30	190	3979	39,2
50%	27	176	3617,6	35,7

**Tabella 6.8:** In questa tabella sono riportati i valori di energia risparmiata nel caso di ordinamento crescente dei nodi sulla rete Ebone.

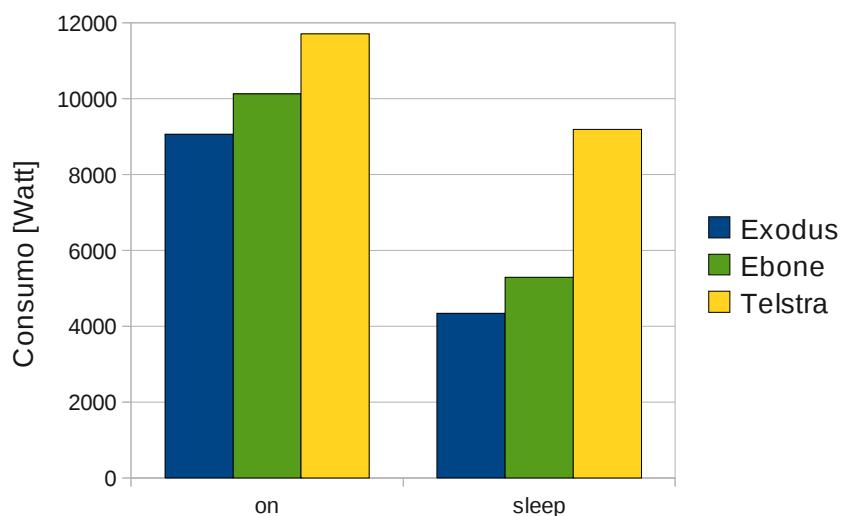
Exodus: 294 link, 79 nodi - consumo totale=9058.2 W				
Carico	Nodi off (#)	Link off (#)	Risparmio energia (W)	Risparmio (%)
10%	37	208	4715,2	52
20%	20	148	2808.4	31

**Tabella 6.9:** In questa tabella sono riportati i valori di energia risparmiata nel caso di ordinamento crescente dei nodi sulla rete Exodus.

Telstra: 302 link, 104 nodi - consumo totale=11708.6 W				
Carico	Nodi off (#)	Link off (#)	Risparmio energia (W)	Risparmio (%)
10%	18	132	2518,8	21,5
20%	18	132	2518,8	21,5
30%	17	128	2403,2	20,5
40%	15	118	2157,4	18,4

**Tabella 6.10:** In questa tabella sono riportati i valori di energia risparmiata nel caso di ordinamento crescente dei nodi sulla rete Telstra.

In Figura 6.5 viene mostrato un grafico del possibile risparmio energetico ottenibile nelle tre reti considerando un carico di rete del 10%. Nella parte di sinistra abbiamo riportato il consumo totale delle reti considerando attivi tutti i nodi e tutti link in esse presenti, mentre nella parte di destra abbiamo riportato i consumi delle reti calcolate dal nostro algoritmo. Come si può notare, per i motivi già citati, il risparmio maggiore si ottiene nelle reti Exodus ed Ebone, a seguire Telstra.



**Figura 6.5:** Il grafico riporta il consumo energetico nelle reti Ebone, Exodus e Telstra, con tutti i nodi attivi e con i nodi in sleep, con un carico del 10%.



# 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 [31] 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 questo 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, discusso in questa tesi. Questi scenari sono stati implementati in un ambiente di lavoro basato sull'emulatore di reti Netkit.

Seguendo l'affermazione di Gupta e Singh: "durante i periodi di sleep è necessario sostituire l'algoritmo SPF con uno in grado di selezionare il numero minimale di link in grado di garantire le richieste di traffico e soddisfare le QoS", abbiamo sviluppato un algoritmo in grado di effettuare una selezione dei nodi e dei link da spegnere.

Illustriamo infine le performance dell'algoritmo proposto applicato ad alcune topologie di rete reali. Dai risultati si evince che la topologia ha un grande impatto sul risparmio energetico, persino maggiore a quello del carico di traffico. Vediamo infatti come in reti diverse si passi da un possibile risparmio energetico superiore al 40% ad un 20%, come nel caso della rete Telstra.

I risultati confermano la validità della formulazione. Nonostante si tratti di una soluzione euristica, in situazioni di basso carico l'algoritmo ha permesso di spegnere nodi e link basandosi su un numero limitato di informazioni.

Il lavoro offre diversi spunti per possibili sviluppi futuri. Sarebbe infatti interessante eseguire dei nuovi test utilizzando matrici di traffico reali, oltre che l'utilizzo di reti emulate di maggiori dimensioni. Per migliorare l'algoritmo si potrebbe inoltre includere l'utilizzo dei link come possibile metrica. Infine, sarebbe interessante

effettuare un'analisi dell'impatto della soluzione proposta sulle connessioni TCP e sulla QoS.





# Bibliografia

- [1] Datasheet router Juniper. <http://www.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] Rocketfuel Project [Online]. [www.cs.washington.edu/research/networking/rocketfuel/](http://www.cs.washington.edu/research/networking/rocketfuel/).
- [10] Sourceforge. <http://sourceforge.net/>.
- [11] The Linux Kernel Archives. <http://www.kernel.org/>.
- [12] The Network Simulator - NS-2. <http://www.isi.edu/nsnam/ns/>.
- [13] UML Utilities. <http://user-mode-linux.sourceforge.net/dl-sf.html>.
- [14] University of Roma Tre Computer Networks Research Group. Netkit. <http://www.netkit.org/>.

- 
- [15] University of Roma Tre Computer Networks Research Group. Netkit ready to use Labs. <http://www.netkit.org/labs.html>.
- [16] User-mode Linux Kernel. <http://user-mode-linux.sourceforge.net/>.
- [17] vnStat. <http://humdi.net/vnstat/>.
- [18] A. Cianfrani, V. Eramo, M. Listanti, M. Marazza e E. Vittorini. *An Energy Saving Routing Algorithm for a Green OSPF Protocol*. IEEE INFOCOM 2010, San Diego (USA), Marzo.
- [19] 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.
- [20] L. Berger, I. Bryskin, A. Zinin, and R. Coltun. *The OSPF Opaque LSA Option*. RFC 5250, 2008.
- [21] 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.
- [22] 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.
- [23] Jeff Dike. *A User-Mode Port of the Linux Kernel*. In Proc. 4th Annual Linux Showcase and Conference, pagine 63–72, Ottobre 2000.
- [24] Jeff Dike. *Double your Fun with User-Mode Linux*. Novembre 2004.
- [25] Jeff Dike. *User-Mode Linux*. Talk at the 2001 Ottawa Linux Symposium, Luglio 2001.

- 
- [26] Jeff Dike. *User Mode Linux*. Prentice Hall, 2006.
- [27] S. Abeck, B. Neumaier e H. Hegering. *Integrated Management of Networked System*. Morgan Kaufmann, 1999.
- [28] J. F. Kurose e K. W. Ross. *Internet e Reti di Calcolatori, seconda edizione*. McGraw-Hill, 2003.
- [29] 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.
- [30] 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.
- [31] 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), (Karlsruhe, Germany), Agosto 2003.
- [32] Thomas M. e Thomas II. *OSPF Network Design Solution*. Cisco Press, 2003.
- [33] Paolo Giarrusso. SKAS patches and UML updates. <http://www.user-mode-linux.org/blaisorblade/>.
- [34] Paolo Giarrusso. UML Utilities. <http://www.user-mode-linux.org/blaisorblade/uml-utilities/>.
- [35] Kunihiro Ishiguro. GNU Zebra Routing Software. <http://www.zebra.org/>.
- [36] 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.

- [37] K. Ho e C. Cheung. *Green distributed routing protocol for sleep coordination in wired core networks*. In Networked Computing (INC), 2010 6th International Conference on, 2010.
- [38] K.J. Christensen e P.C. Gunaratne. *Predictive power management method for network devices*. IEEE 2005 29 March 2005, Electronics Letters online no: 20051149.
- [39] L. Chiaraviglio, M. Melia e F. Neri. *Energy-Aware Networks: Reducing Power Consumption By Switching Off Network Elements*. In FEDERICA-Phosphorus tutorial and workshop (TNC2008), Brugges, BE, 2008.
- [40] L. Chiaraviglio, M. Melia e F. Neri. *Reducing Power Consumption in Backbone Networks*. IEEE ICC 2009, Dresden, Germany.
- [41] Brad Marshall. User Mode Linux, VMWare and Wine – Virtual Machines under Linux. <http://quark.humbug.org.au/publications/linux/uml.pdf>, Aprile 2003.
- [42] K. McCloghrie, D. Perkins, and J. Schoenwaelder. *Structure of Management Information Version 2 (SMIv2)*. RFC 2578 (Standard), 1999.
- [43] J. Moy. *Multicast Extensions to OSPF*. RFC 1584, 1994.
- [44] J. Moy. *OSPF Version 2*. RFC 2328 (Standard), 1998.
- [45] John T. Moy. *Anatomy of an Internet Routing Protocol*. Addison-Wesley, 1998.
- [46] N. Spring, R. Mahajan e D. Wetherall. *Measuring ISP Topologies with Rocketfuel*. In Proceedings of the ACM SIGCOMM Conference, Agosto 2002.
- [47] 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.

- 
- [48] R. Presuhn. *Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)*. RFC 3416 (Standard), 2002.
- [49] R. Jain e I. Chlamtac. *The P2 algorithm for dynamic calculation of quantiles and histograms without storing observations*. 1985.
- [50] Massimo Rimondini. *Emulation of Computer Networks with Netkit*. Gennaio 2007.
- [51] M.T. Rose. *SNMP MUX protocol and MIB*. RFC 1227, 1991.
- [52] S. Nedeveschi, 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.



# Ringraziamenti

Questa tesi rappresenta il punto di arrivo di un lungo percorso scolastico iniziato tanto tempo fa e il coronamento della carriera universitaria iniziata nel 2005. Tante persone mi sono state vicine in questi anni, aiutandomi a raggiungere questo importante traguardo della mia vita.

Vorrei in primo luogo ringraziare il professor Antonio Capone per gli insegnamenti ricevuti e per avermi seguito durante questo lavoro di tesi trovando tempo da dedicarmi nonostante i molti impegni.

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

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

Un enorme grazie ai miei genitori, Rosanna e Claudio, a mio fratello Paolo e a tutti i familiari, per avermi sempre sostenuto ed incoraggiato in tutti questi anni ed in particolare durante questi mesi finali. Li ringrazio per aver sempre creduto in me ed avermi permesso di arrivare fin qui.

Rivolgo i dovuti ringraziamenti a tutti i colleghi che hanno condiviso con me 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. Abbiamo passato tantissimi momenti indimenticabili insieme.

Un ringraziamento particolare va a Ema e Riky, membri importanti del team, con cui ho condiviso questi mesi finali e senza i quali questo lavoro non sarebbe stato possibile.

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

Grazie a tutti, Luca.