POLITECNICO DI MILANO
Facoltà di Ingegneria dell'Informazione
Corso di Laurea Specialistica in Ingegneria Informatica

Progettazione e sviluppo di una piattaforma di power management basata su architettura multi-core per apparati di reti in fibra ottica

Relatore: **prof. Carlo Brandolese**
Correlatore: **Giorgio Parladori**

Tesi di laurea di
Michele MELCHIORI
matricola 735358

Anno Accademico 2011/2012

Experience is what you get
when you don't get what you want

Randolph Frederick (Randy) Pausch
(1960-10-23 — 2008-07-25)

Alla mia famiglia
ai miei amici
e a tutte le persone che, in un modo o nell'altro,
mi hanno supportato, sopportato e accompagnato
durante questo percorso.

Un grazie particolare a Nicola e Marco,
per il supporto morale che hanno saputo darmi.

Indice

1 Introduzione ed obiettivi **1**
- 1.1 Evoluzione delle reti di telecomunicazioni . . . 1
- 1.2 Obiettivi di lungo periodo . . . 3
- 1.3 Motivazione . . . 4
- 1.4 Obiettivo del lavoro di tesi . . . 6

2 Architettura **9**
- 2.1 Architettura d'insieme . . . 9
  - 2.1.1 Scelte implementative . . . 10
- 2.2 Tecnologie . . . 12
  - 2.2.1 Interne al nodo . . . 13
  - 2.2.2 Esterne al nodo . . . 15

3 Nodo di rete **19**
- 3.1 Introduzione . . . 19
- 3.2 Architetture multi core . . . 20
- 3.3 Comunicazione tra i due software . . . 22
- 3.4 Avvio asimmetrico del nodo . . . 23
  - 3.4.1 Bootloader . . . 24
  - 3.4.2 Device Tree . . . 24
- 3.5 Software di dump . . . 25
- 3.6 Modello del consumo di potenza . . . 27

4 Comunicazione **29**
- 4.1 Web Service . . . 29
- 4.2 Struttura di un Web Service . . . 30
- 4.3 File WSDL . . . 32
- 4.4 Implementazione in PHP . . . 34
- 4.5 Implementazione in JavaScript . . . 35
- 4.6 Interfaccia con l'amministratore . . . 36
  - 4.6.1 Requisiti funzionali . . . 36

4.6.2 Client JavaScript 38

5 Risultati ottenuti **39**

6 Conclusioni **43**
6.1 Sviluppi futuri 44

A Modello XML del nodo **47**
A.1 Modello XML 47
A.2 File DTD 48

B Avvio asimmetrico **51**
B.1 Device Tree 51
B.2 Configurazione dell'avvio asimmetrico 53
B.2.1 Configurazione e compilazione dei device tree 53
B.2.2 Configurazione e compilazione dei kernel 54

C Metodi implementati **55**

Elenco delle figure

1.1 Esempio di rete con carico asimmetrico 3
1.2 Struttura di esempio di un nodo di rete 5

2.1 Confronto tra soluzione centralizzata e distribuita 10
2.2 Schema della soluzione per un generico nodo di rete 12
2.3 Passaggio da architettura single core a dual core 13
2.4 Confronto tra DBMS con software server e DBMS embedded . 15
2.5 Architettura basata su Web Service con PHP e JavaScript . . 18

3.1 Diversi approcci alla programmazione multicore 20
3.2 Aggiunta dei power manager alla rete 23

4.1 Architettura generale di un Web Service 31
4.2 Esempio di doppia RPC 32
4.3 Confronto di specifiche tra WSDL 1.1 e 2.0 33
4.4 Struttura interna del provider 34
4.5 Struttura interna del requester 35

5.1 Direzioni nelle quali è stato trasferito il file di test 40
5.2 Architettura logica della rete di test 40
5.3 Schema di esecuzione di una query distribuita 41

C.1 Alberi delle strutture dati da fornire ai metodi implementati . 57
C.2 Albero della struttura dati restituita dai metodi implementati 57

Sommario

La crescita di importanza delle reti di telecomunicazione nella società odierna è un fatto ormai risaputo. Tali reti permettono di collegare qualsiasi dispositivo elettronico che ruota attorno alla persona, a partire da quelli più ovvi, come computer e telefoni cellulari, fino a raggiungere gli strumenti più remoti, come quelli dedicati alla gestione automatica dell'illuminazione e del riscaldamento all'interno delle case. Questo ha permesso di modificare le modalità con le quali si utilizza la rete: la navigazione web sta perdendo di importanza, a favore di servizi evoluti e di maggiore interattività come la telefonia o l'online gaming.

Anche all'interno di realtà più piccole della rete Internet, come ad esempio nelle aziende o nelle università, si sta riscontrando la stessa tendenza ad ampliare la rete di comunicazione interna, in modo che possa offrire agli utenti una piattaforma con la quale sia possibile accedere a risorse distribuite, come ad esempio uno spazio di archiviazione remoto tramite il quale sia possibile condividere progetti e documenti o fare in modo che file e archivi importanti siano registrati in sicurezza su supporti di backup.

Una proprietà dei dispositivi che sono alla base del funzionamento delle reti è il fatto che il loro consumo di potenza, anche se non costante, non varia di molto in funzione del carico di lavoro; questa osservazione, accoppiata a quella che permette di affermare che il traffico dati gestito dalla rete non è costante nel tempo né uniforme, porta a notare che la rete è un'area nella quale si trovano sprechi energetici proporzionali alla sua estensione. Questi sprechi, se non correttamente gestiti, in caso di reti discretamente estese possono essere considerati come non trascurabili.

Scopo del lavoro di tesi è progettare e costruire una base per una piattaforma che fornisca dei servizi di power saving proprio dei dispositivi di rete, con l'obiettivo di ridurre al minimo gli sprechi energetici del networking senza intaccare le prestazioni alle quali gli utenti della rete sono abituati.

Capitolo 1

Introduzione ed obiettivi

In questo capitolo si presenta l'obiettivo del lavoro di tesi. Inizialmente si descrive il contesto all'interno del quale si andrà ad operare, evidenziando le problematiche e le limitazioni della soluzione attuale e suggerendo un obiettivo di lungo periodo nei termini di quelle che dovrebbero essere le nuove funzionalità fornite per risolvere completamente tali problematiche. Si espone infine una breve analisi dei passi che devono essere compiuti per raggiungere tale obiettivo, e una descrizione delle principali attività sviluppate nel corso della tesi in questa direzione.

1.1 Evoluzione delle reti di telecomunicazioni

Le reti di telecomunicazione rivestono un'importanza sempre maggiore all'interno della società odierna. Le infrastrutture di rete, inizialmente concepite per supportare i servizi di telefonia, stanno sempre più evolvendo verso un'infrastruttura a supporto della connettività dati globale, in cui si assiste alla convergenza e all'integrazione di tecnologie e soluzioni di complessità sempre crescente, con scopi che vanno oltre la semplice comunicazione tra persone ma includono anche e soprattutto servizi più evoluti. Gli utenti si sono spostati dall'utilizzo del telefono fisso o del cellulare sempre più verso lo sfruttamento della gratuità del del VOIP (*Voice Over Internet Protocol*) e, data la maggiore larghezza di banda a disposizione, hanno avuto la possibilità di comunicare tramite video chiamate e video conferenze.

Anche la distribuzione di contenuti multimediali, come televisione e radio, sta evolvendo verso Internet: è diventata cosa normale trovare pubblicati tra le pagine del sito web di un'emittente alcuni contenuti già mostrati in passato tramite i canali standard (*podcast*), e spesso è anche a disposizione uno *streaming* in tempo reale di quanto si sta trasmettendo sulla rete dedicata.

Questo ha gettato la base per i concetti di IPTV e di *video on demand*, nei quali i contenuti sono fruibili dall'utente direttamente tramite il proprio televisore, il quale però li ottiene tramite la rete Internet.

Nel campo della domotica esistono soluzioni che permettono ad una persona di controllare la sorveglianza, l'illuminazione, il riscaldamento ed altri servizi della propria abitazione da qualsiasi punto della Rete.

I punti di accesso sono in continua crescita, tanto che gli indirizzi del protocollo IPv4 non sono più sufficienti ad identificare ogni singolo dispositivo connesso, nonostante ve ne siano a disposizione più di quattro miliardi. Da qui la tecnica del subnetting e, successivamente, la proposta dell'introduzione della sesta versione del protocollo (IPv6) per far fronte a questa crescita che sembra inarrestabile.

Anche in un contesto aziendale la rete sta assumendo un'importanza sempre più vitale: da tempo i dipendenti devono collaborare e mantenere i contatti anche se appartengono ad aree diverse o se si trovano in regioni distanti tra loro. Le conferenze audio e video sono diffuse molto più che nel mercato customer, inoltre l'evoluzione dei vari servizi a livello di cloud rendono semplice la condivisione di file e la collaborazione, al prezzo però di una maggiore pressione sulla rete. Questa in tutti i settori diventa sempre più ampia e pervasiva, di conseguenza le sue capacità, in termini di larghezza di banda, devono almeno inseguire la sua estensione.

I componenti base di ogni rete sono gli apparati di networking, ovvero quell'insieme più o meno ampio di hub, bridge, switch e router che, connessi tra loro e ai client della rete, forniscono agli utilizzatori le funzionalità della rete stessa. La continua crescita ed evoluzione di quest'ultima richiede un incremento del numero e delle capacità di questi apparati, in modo da consentire ad ogni nuovo client di avere la possibilità di poter sfruttare i servizi messi a disposizione dalla rete o fornire i propri senza dover soffrire limitazioni di banda.

Ognuno di questi dispositivi deve essere collegato alla rete elettrica per poter funzionare, ed il suo consumo di potenza è proporzionale alle prestazioni che fornisce. Da questo deriva che l'incremento del numero di apparati, delle connessioni e dei punti di accesso alla rete porta un incremento proporzionale del consumo di potenza che diventa sempre meno trascurabile nel bilancio energetico di un'azienda.

Inoltre, come è facile immaginare, il carico di lavoro demandato alla rete non gode della proprietà di essere costante nel tempo: ad esempio di notte il traffico da gestire è quasi nullo, mentre durante il giorno si avranno dei picchi di massimo. Un'altra proprietà che non si può assegnare al carico sulla rete è l'uniformità nello spazio: ad esempio se si analizza la rete di Figura 1.1 si può notare che molti nodi stanno comunicando con il server evidenziato, di

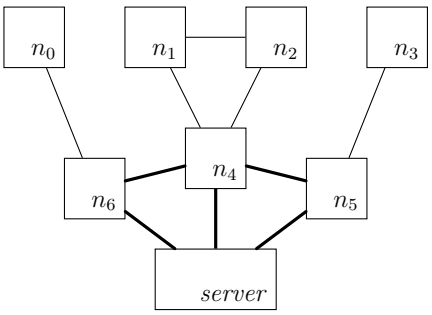


Figura 1.1: Esempio di rete nella quale alcune connessioni (in neretto) sono soggette ad un maggior traffico rispetto alle altre

conseguenza la maggior parte del traffico passa attraverso i link vicini a tale server.

Va da sé che la rete, soprattutto se situata in ambienti critici come ospedali o industrie che trattano materiali pericolosi, deve comunque essere in grado di fornire la larghezza di banda minima garantita ad ogni dispositivo, pena il sovraccarico che porta ad un *denial of service* di un terminale che potrebbe svolgere funzioni anche critiche.

L'obiettivo di lungo periodo che si vuole perseguire e che inizia con questo lavoro è quindi quello di sviluppare ed implementare politiche di power saving della rete con lo scopo di ridurre il consumo di potenza degli apparati di networking, sfruttando il fatto che, come appena descritto, l'utilizzo dei link non è costante né uniforme.

1.2 Obiettivi di lungo periodo

Da quanto è stato descritto nella sezione precedente è evidente la necessità di porre la riduzione del consumo di potenza degli apparati di networking tra gli obiettivi primari dello sviluppo di nuove reti di telecomunicazioni. Un modo per ottenere ciò è quello di modellare la rete come un insieme di apparati interconnessi, controllati da un elemento che gli permette di adattarsi dinamicamente alle variazioni del carico di lavoro, disattivando le componenti meno utilizzate per aggregare il traffico sul minor numero di link possibile, ma anche riattivando quanto è stato posto in precedenza in *stand-by* nel caso il traffico torni ad aumentare.

Queste operazioni sono già eseguibili manualmente da un operatore di rete: esso infatti è in grado di connettersi ad ogni nodo e, tramite una shell remota, di impartire comandi di *suspend* e di *wake-up*. Queste operazioni, essendo lente e da eseguire nodo per nodo, sono fattibili solo se si ritiene che

una porzione della rete resterà poco utilizzata per un periodo rilevante di tempo.

L'obiettivo che si vuole porre è la creazione di uno *sviluppo sostenibile* delle reti di telecomunicazione, tramite la creazione di una soluzione software che sia in grado di gestirle in maniera dinamica e quasi in tempo reale, in modo che sia possibile applicare una politica di risparmio energetico senza la necessità dell'intervento di un operatore umano. Per poter fare ciò il software deve essere in grado di analizzare il livello di utilizzo dei singoli nodi e delle interfacce che li compongono: con queste informazioni deve determinare una distribuzione del carico istantaneo sulla rete; se ha a disposizione anche dei dati storici può tentare una sorta di previsione del traffico futuro. Ad ogni modo deve prendere delle decisioni riguardanti quali interfacce attivare o disattivare, in modo che nessun link sia sovraccaricato e, d'altro canto, nessuno sia attivo ma con un utilizzo minimo.

1.3 Motivazione

L'obiettivo che si vuole raggiungere è facilmente motivabile dopo un'analisi dettagliata del consumo di potenza di un singolo apparato.

Struttura del nodo Un nodo di rete può essere modellizzato come mostrato in Figura 1.2. Esso è sostanzialmente una struttura gerarchica, che di conseguenza si presta molto bene ad essere descritta con linguaggi come XML o JSON; un esempio di file che descrive il nodo mostrato è presente in Appendice A.

Partendo dal livello più basso si trovano le interfacce di rete, i dispositivi che gestiscono fisicamente il traffico dei dati sui link. La *line card* è la scheda sulla quale sono montate le interfacce, il suo compito è quello di controllarle e smistare il traffico tra di esse. In un nodo di rete è possibile che siano montate più line card, quindi tra loro è sempre presente una matrice di switch (*matrix*) con il solo scopo di instradare il traffico tra una line card e un'altra. Essendo un componente abbastanza vitale del nodo, essa è sempre affiancata da almeno una matrix di backup che in caso di problemi può prendere il suo posto. A fianco di line card e matrix vi sono le altre parti comuni che compongono il nodo di rete, come l'alimentatore, la CPU ed il sistema di raffreddamento.

Ognuno di questi componenti ha un identificativo che ne evidenzia la posizione all'interno della rete. I nodi, essendo il componente più in alto nella gerarchia, hanno solamente un indice numerico progressivo; i dispositivi all'interno del nodo seguono una nuova numerazione con l'indice del nodo

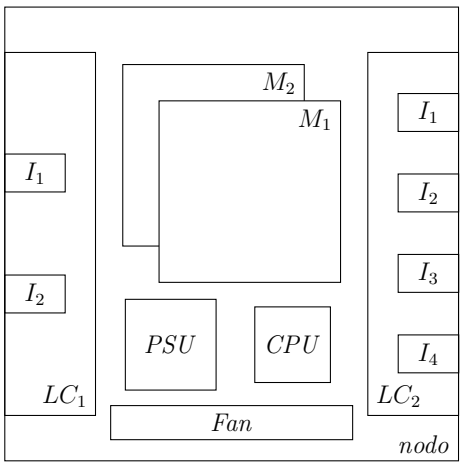


Figura 1.2: Struttura di esempio di un nodo di rete

come prefisso: ad esempio nel nodo n si potrebbe avere $n.1$ per la CPU, $n.2$ per l'alimentatore e $n.5$ per una line card. Ogni interfaccia di rete su questa line card sarà identificata da un altro numero progressivo i, nella forma $n.5.i$.

Consumo di potenza Il modello di consumo di potenza creato per questi dispositivi segue l'equazione:

$$P(t) = P_s + P_d \cdot \frac{BW(t)}{MBW}$$

dove $P(t)$ è il consumo di potenza e $BW(t)$ la larghezza di banda utilizzata, entrambi al tempo t. Gli altri elementi dell'equazione rappresentano i parametri statici del dispositivo:

P_s è il consumo di potenza statico, ovvero quanti Watt consuma il dispositivo mentre è acceso e completamente in *idle*;

P_d è il massimo consumo di potenza dinamico, ovvero quella potenza in più rispetto a P_s che serve al dispositivo per gestire il traffico quando $BW(t) = MBW$;

MBW è la capacità massima, in termini di larghezza di banda, del dispositivo: il flusso massimo di dati che è in grado di controllare.

Su questo modello è interessante fare almeno due osservazioni. La prima riguarda il consumo dinamico di potenza: nel funzionamento reale del dispositivo tale consumo è influenzato non solamente dalla larghezza di banda utilizzata, ma anche da altri fattori, non ultimo la temperatura di esercizio;

è stato deciso di adottare questa funzione lineare in quanto è già una buona approssimazione del comportamento reale del dispositivo, e permette di mantenere il modello in funzione delle sole variabili che è possibile controllare tramite il power manager, senza dover considerare componenti aleatorie.

La seconda considerazione riguarda invece la suddivisione presente nel modello tra consumo di potenza statico e dinamico, che mette in evidenza come il dispositivo assorbe energia anche se non sta eseguendo nessuna operazione, ma solamente per il fatto che esso è attivo. Per valutare il peso della componente statica del consumo di potenza si consideri un'interfaccia ottica in grado di gestire fino a 10Gb/s; i suoi parametri di funzionamento sono $P_s = 20W$ e $P_d = 10W$. Significa che il suo consumo di potenza, quando è attiva, varia tra 20 e 30 Watt. Si prenda ora una parte di rete nella quale è presente un nodo sul quale arriva un traffico di 8Gb/s, questo può essere indirizzato su due diverse interfacce che hanno gli stessi parametri appena indicati. Si possono isolare due diverse situazioni:

1. le due interfacce sono entrambe attive: in questo caso, data la simmetricità dei loro parametri, la somma dei loro consumi di potenza sarà data da $2 \cdot P_s + 8/MBW = 48W$;

2. il traffico è gestito completamente da una sola delle due interfacce, mentre l'altra è spenta: in questo caso la somma dei consumi di potenza si riduce a $P_s + 8/MBW = 28W$.

Da questo esempio emerge che lo stesso servizio fornito dal nodo alla rete, l'instradamento cioè di un traffico di 8Gb/s, può essere completato in entrambe le situazioni elencate, ma nella seconda il consumo di potenza è circa il 58% della prima: spegnere l'interfaccia che al momento può restare inutilizzata porta ad una riduzione di circa il 40% dei consumi. Questo dimostra l'utilità di una piattaforma software in grado di disattivare le interfacce o i dispositivi poco utilizzati per poterli riattivare all'occorrenza quando il volume di traffico sulla rete torna a crescere.

1.4 Obiettivo del lavoro di tesi

Una soluzione software in grado di compiere le operazioni descritte si sviluppa su diversi livelli, ognuno dei quali richiede un ampio sviluppo.

Al livello più basso vi è tutto quanto è richiesto per raccogliere le informazioni di consumo di potenza all'interno di ogni nodo della rete e fare in modo che queste informazioni siano accessibili al software di gestione. Questo prevede la progettazione di come raccogliere le informazioni, di come dar

loro persistenza fintanto che hanno un'utilità per le politiche di power management ed infine del protocollo di comunicazione che permette al manager di ottenere queste informazioni e di inviare comandi sulla rete.

Al secondo livello va creata un'interfaccia utente, con la duplice funzionalità di agire da piattaforma di debugging del livello base e di poter iniziare a far funzionare il sistema. Questa interfaccia infatti deve permettere ad un utente di ottenere determinare informazioni dalla rete, perché possa analizzarle e valutare l'invio di comandi di attivazione e disattivazione di apparati. Questo livello rappresenta una soluzione sub-ottima, in quanto la gestione della rete è demandata ad un operatore umano, ma almeno rappresenta una possibilità di analisi ed una velocizzazione nella gestione dei vari apparati, in quanto non necessita dell'accesso alla shell remota di ogni singolo nodo su cui si intende agire.

Più in alto si trova la necessità di creare il programma che automaticamente è in grado di gestire la rete. Questo deve essere in grado di eseguire in maniera autonoma quello che era compito dell'operatore al livello precedente, fermo restando la possibilità del controllo manuale imposto da un amministratore umano.

Lavoro di tesi Data la vastità della soluzione completa questo lavoro di tesi si concentra sullo sviluppo del suo livello più basso. Questa attività si suddivide in più passi, dei quali il primo deve riguardare una progettazione preventiva della struttura completa della soluzione: questa è esposta nel Capitolo 2. Durante questo passo devono essere anche compiute delle scelte di natura tecnologica, per determinare quali, tra le alternative hardware e software individuate, si adattano meglio ai requisiti posti. La prima analisi va compiuta sulla piattaforma hardware a supporto del *software embedded* che rappresenta il cuore del controllo e della gestione di ogni singolo nodo di telecomunicazione, per valutarne in particolare l'evoluzione verso il multi core con l'obiettivo di rispettare i requisiti di scalabilità, affidabilità e flessibilità futura posti. Analisi successive vanno compiute tra i DBMS MySQL e SQLite confrontati con dei file di log, per determinare la soluzione migliore in grado di dare persistenza ai dati storici di consumo di potenza di ogni dispositivo all'interno della rete. Sul piano delle comunicazioni le alternative da valutare sono lo standard SNMP contro un'implementazione RESTful; da quest'ultima scaturisce inoltre anche la soluzione dei Web Services. Altre tecnologie da valutare comprendono il boot asimmetrico del nodo di telecomunicazione tramite apposite configurazioni di U-Boot e dei relativi Device Tree (Sezione 3.4), e di un Hypervisor di comunicazione tra due kernel differenti (Sezione 3.3). Per quanto riguarda gli endpoint della comunicazione

sono da valutare dei software dedicati in alternativa a script PHP e/o client HTML con JavaScript.

Compiute le scelte implementative è necessario procedere con lo sviluppo di ogni componente del livello, sia per quanto riguarda il software in esecuzione su ogni nodo di rete (Capitolo 3) che le configurazioni necessarie per la corretta comunicazione tra i vari componenti (Capitolo 4). Il lavoro di tesi si concentra sulla creazione della piattaforma di base, relegando l'interfaccia utente ad un'implementazione minimale, descritta nella Sezione 4.6, che ha il semplice scopo di debug e analisi delle comunicazioni. Per l'analisi ed il testing di funzionamento dell'avvio asimmetrico di due Sistemi Operativi e la loro successiva capacità di comunicazione è stata sfruttata una piattaforma QorIQ™ P2020 fornita da Freescale. Questa è una piattaforma dual core basata sull'architettura PowerPC™, ed è stata scelta per la sua somiglianza con l'hardware che va a comporre il cuore di un nodo di telecomunicazione.

Lo sviluppo di un'interfaccia amministrativa più completa e dei livelli più elevati della soluzione è rimandato a futuri lavori di estensione.

Capitolo 2

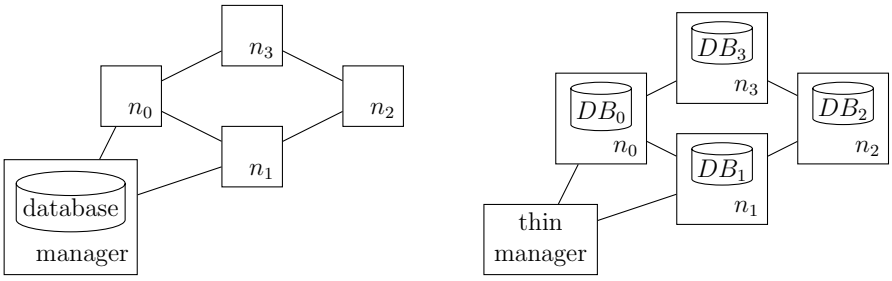
Architettura

In questo capitolo è analizzata la soluzione proposta. Nella prima parte ci si è soffermati sulla visione generale della soluzione e sul suo funzionamento, mentre solo successivamente si sono evidenziati i confronti tra le tecnologie disponibili e le scelte progettuali che sono state fatte sulla base dei requisiti specifici della soluzione.

2.1 Architettura d'insieme

Dati gli obiettivi posti al capitolo precedente, la prima scelta da affrontare è tra una soluzione centralizzata o distribuita. Nel primo caso (Figura 2.1a) si avrebbe la necessità di dedicare un dispositivo in rete ad eseguire due compiti: fungere da server sul quale ogni altro nodo deve inviare periodicamente le letture dei valori misurati dai propri sensori e, in secondo luogo, analizzare i dati ricevuti per poter inviare comandi di accensione o spegnimento dei dispositivi meno utilizzati. In questa situazione si ha che *tutta* l'informazione generata dai nodi è concentrata in un unico punto: questo porta alla possibilità di implementare una politica molto più accurata e complessa, dato che il power manager sarebbe a conoscenza della struttura dell'intera rete, ma a discapito di uno sbilanciamento del traffico nei pressi del manager, in quanto questo periodicamente deve poter ricevere le letture dai nodi. Tale sbilanciamento cresce con le dimensioni della rete, in quanto l'aumentare del numero dei nodi è seguito dall'incremento del volume di traffico necessario a trasmettere al manager ogni lettura.

In secondo luogo la soluzione centralizzata richiede che il manager sia sempre online e che abbia spazio di archiviazione sufficiente a mantenere tutto lo storico dell'intera rete. Tale storico deve poi essere analizzato, e



(a) Centralizzata (b) Distribuita

Figura 2.1: Confronto tra soluzione centralizzata e distribuita

questo può richiedere capacità computazionali non indifferenti per reti di dimensioni importanti.

In una soluzione distribuita (Figura 2.1b) si è invece in grado di suddividere uniformemente su tutta la rete il traffico necessario al corretto funzionamento delle politiche di power saving inoltre, data la possibilità di sfruttare lo storage locale, si rilassa il vincolo per il quale ogni nodo deve trasmettere in rete ogni lettura.

Un'ultima considerazione è di natura concettuale: in un'architettura centralizzata si crea una separazione fisica tra la rete e le informazioni che la riguardano, separazione che può diventare problematica nel caso in cui l'amministratore non abbia accesso al dispositivo che agisce da manager o peggio ancora nel caso di failure del manager stesso. La soluzione alternativa permette invece di implementare una sorta di base di dati distribuita: le informazioni restano strettamente legate alla rete descritta e sono disponibili fintanto che la rete è raggiungibile.

Tutte queste considerazioni hanno permesso di optare per la soluzione distribuita, la quale però ha per contro il costo di dover dedicare una parte dello storage di ogni nodo all'archiviazione delle letture dei propri sensori e una parte della capacità di elaborazione alle politiche di power saving.

2.1.1 Scelte implementative

Per la costruzione della soluzione si è posto un occhio di riguardo all'aderenza agli standard esistenti, in modo da permettere e facilitare future modifiche o estensioni della soluzione stessa. Si è reso necessario eseguire uno studio su due diversi fronti:

1. da un lato vi è la necessità di mantenere un registro storico dei valori di consumo di potenza e quantità di banda utilizzata da ogni interfaccia di rete del nodo;

2. dall'altro tali informazioni devono essere rese disponibili all'esterno tramite un software server in grado di analizzare le richieste ricevute e ritornare i dati interessati.

Sensing e persistenza dei dati Il punto 1 dell'elenco precedente richiede l'analisi di tre problematiche distinte:

1.a. come eseguire una lettura non invasiva delle informazioni di utilizzo di banda e consumo di potenza;

1.b. come salvare in maniera persistente queste informazioni in modo che possano essere recuperate e fornite in risposta ad un'interrogazione del manager;

1.c. come evitare di far crescere in maniera incontrollata la dimensione dell'archivio di letture passate, considerando anche il fatto che mano a mano che il dato appartiene ad un passato remoto il suo valore informativo si riduce.

La necessità di trasparenza rispetto al funzionamento standard del nodo richiesta al punto 1.a è risolta da sensori hardware che sono già presenti sui vari dispositivi, e che compiono le misurazioni senza interferire con l'operatività normale del nodo. Quando i sensori non sono disponibili le misurazioni sono sostituite da un modello di consumo di potenza sulla base della larghezza di banda utilizzata ottenuto da misurazioni eseguite direttamente sull'hardware in laboratorio.

Per quanto riguarda il problema di persistenza e reperibilità dei dati (1.b), la scelta era tra un file di log testuale e una base di dati relazionale. Si è deciso per quest'ultima alternativa, per ottenere in blocco sia una gestione ottimizzata dei dati sia un'interfaccia di inserimento e ricerca attraverso un linguaggio standard (SQL) che di conseguenza è facilmente riutilizzabile in futuro.

Questa scelta ha un altro risvolto positivo, cioè il fatto che tramite un database si è in grado di risolvere facilmente il problema 1.c: è infatti sufficiente la creazione di un trigger che in maniera automatica dopo ogni inserimento analizza lo stato del database e raggruppa tra loro le tuple appartenenti ad un passato troppo remoto, eliminando i dati grezzi. Quando anche gli aggregati storici perdono troppo valore informativo si ha la possibilità di creare un secondo livello di aggregazione oppure di eliminarli perché ormai inutili.

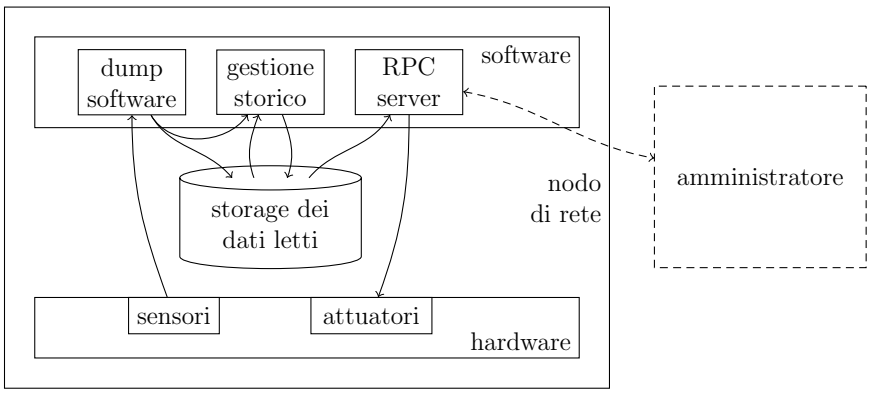


Figura 2.2: Schema della soluzione per un generico nodo di rete

Condivisione dei dati Il secondo fronte sul quale è stato progettato lo sviluppo riguarda la condivisione sulla rete dei dati ottenuti dai sensori, in modo che questi possano essere analizzati per potervi applicare la politica di ottimizzazione energetica.

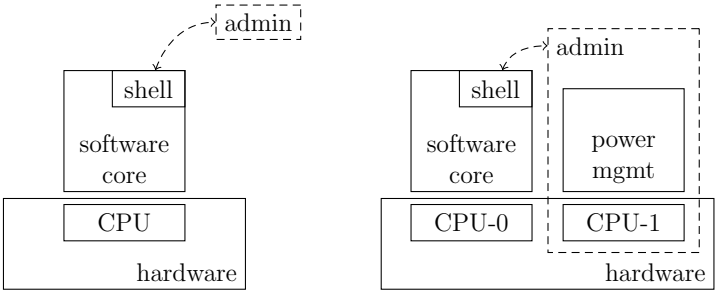
Date le scelte fatte in materia di dislocamento dei dati, per ogni nodo della rete si è deciso di creare un server che esponga un insieme di funzioni RPC (Remote Procedure Call) tramite le quali un client può eseguire due compiti: da un lato richiedere la disattivazione o l'attivazione di particolari componenti del nodo e dall'altra ottenere le informazioni di cui necessita per poter determinare quali comandi inviare.

Uno schema generale della struttura della soluzione è mostrato in Figura 2.2, nella quale si può notare una rappresentazione astratta degli elementi e delle interazioni che avvengono tra loro per il corretto funzionamento della piattaforma.

2.2 Tecnologie

Dopo aver compiuto delle scelte generali e di alto livello si è reso necessario entrare nel particolare di ogni tecnologia da sfruttare per poter completare il progetto della soluzione. Per ogni ambito che si è identificato si è cercato di valutare almeno un'alternativa, mantenendo l'attenzione su vantaggi e svantaggi di ognuna di esse.

Come situazione iniziale dal punto di vista tecnologico si ha il nodo che, tramite un'architettura single core, esegue il suo software di gestione ed espone verso la rete una shell remota che può essere utilizzata dall'amministratore per modificare i parametri di funzionamento.



(a) single core (b) dual core

Figura 2.3: Passaggio da architettura single core a dual core

2.2.1 Interne al nodo

Avvio asimmetrico Un primo requisito che è stato imposto all'applicazione è quello di mantenere un isolamento che sia maggiore possibile tra il software che esegue la gestione dei pacchetti di rete (il software *core*) e quello della soluzione, che esegue la parte del *management plane* della rete con l'obiettivo del *power saving*. Questo requisito è stato posto per fare in modo che le modifiche da apportare al software core del nodo siano ridotte al minimo, nella politica di "non introdurre potenziali problemi in ciò che è già funzionante".

Per poter soddisfare questo requisito si è deciso di sfruttare una particolare capacità del bootloader utilizzato: la possibilità di eseguire un avvio asimmetrico del nodo. Questo significa che, dato un adeguato partizionamento delle risorse hardware, è possibile istruire tale bootloader per caricare in memoria ed avviare l'esecuzione di due kernel differenti, totalmente isolati l'uno dall'altro. Come mostrato in Figura 2.3, questa idea prevede un cambio architetturale a livello hardware, il passaggio da single core a dual core, così da avere un processore dedicato ad eseguire il software originale del nodo e l'altro a gestire il *power management plane*. Dal punto di vista del software già scritto non cambia nulla, infatti esso continua a funzionare su un solo core con le risorse che aveva a disposizione in precedenza, e il suo controllo continua ad essere tramite shell remota. Questo software non è a conoscenza dell'esistenza del secondo core sul quale è in esecuzione il software della soluzione, per questo non si ha la possibilità di mettere in comunicazione direttamente i due software.

Si è deciso di sfruttare la possibilità di aggiungere ai kernel un'interfaccia di rete virtuale con il suo driver, per permettere al power manager di inviare comandi verso la shell del software core senza essere costretto ad impiegare un'interfaccia fisica per stabilire la connessione.

Base di dati La scelta di quale base di dati utilizzare per dare persistenza alle letture dai sensori è stata guidata innanzi tutto dalla tipologia di informazioni che devono esservi rappresentate. Esse contengono dati storici che devono essere recuperati solo periodicamente tramite particolari filtri, come i limiti di timestamp e di interfacce interessate; inoltre le informazioni registrate non subiscono alterazioni, essendo relative ad eventi passati.

Si è eseguito un confronto tra due soluzioni software: una classica, basata su un processo che agisce da database server ed espone un'interfaccia, solitamente TCP/IP, alla quale il software deve connettersi per poter interrogare la base di dati, ed un'altra che prevede di integrare l'accesso al database direttamente nelle applicazioni che devono accedere ai dati, come se questo fosse un semplice file al quale i due software devono accedere.

Come rappresentanti delle due categorie di soluzioni sono stati scelti rispettivamente MySQL e SQLite. Il primo è il nome di riferimento per i DBMS open source, con diverse applicazioni in ambito web e presente in molti stack preconfigurati per il desktop[1], mentre il secondo è spesso utilizzato in soluzioni embedded, come telefoni cellulari, o in software che richiede uno storage organizzato dei dati su disco, come molti browser Internet[2].

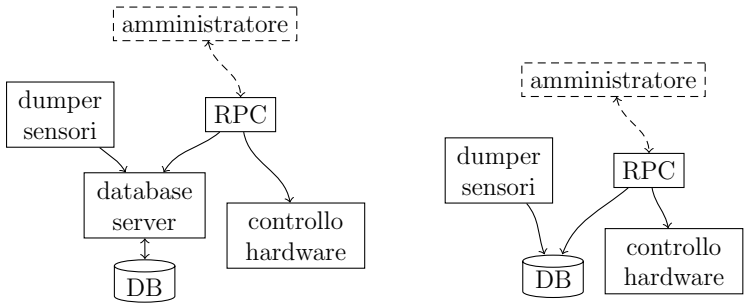
I vantaggi della prima soluzione derivano dalla presenza del software di accesso ai dati, al quale chi utilizza il database deve connettersi per richiedere l'esecuzione delle interrogazioni. Questo software intermedio tra dati e consumatori è in grado di garantire un alto livello di *fault tolerance* e di concorrenza degli accessi.

La seconda soluzione, invece, integra il software di accesso ai dati in una libreria compilata con l'applicazione. Questa chiaramente non garantisce gli stessi livelli di protezione dagli errori di programmazione forniti dalla soluzione precedente, però ha tra i suoi vantaggi il fatto di non richiedere la configurazione e la taratura del server per poter funzionare sul nodo di rete: è infatti sufficiente la possibilità di accedere in scrittura ad un'unità di storage locale. Questa soluzione inoltre permette di mantenere ridotto il consumo di memoria in quanto non rimane nessuna applicazione in esecuzione oltre ai client. Consente infine di ottimizzare lo storage in quanto si ha la possibilità di mantenere l'intero database in un unico file su disco, e la libreria non ha dipendenze esterne da soddisfare o file di configurazione da rispettare.

Le differenze tra le due soluzioni sono mostrate anche in Figura 2.4. Si è valutato che l'accesso fisico al database del nodo è compiuto solo da due software: quello di dump che periodicamente aggiunge nuove tuple e quello che implementa i metodi RPC. Quest'ultimo fornisce l'accesso al database al

[1] fonte: `http://en.wikipedia.org/wiki/MySQL`

[2] fonte: `http://en.wikipedia.org/wiki/SQLite#Adoption`



(a) MySQL (b) SQLite

Figura 2.4: Confronto tra DBMS con software server e DBMS embedded

client della rete (il sistema di power management) ed espone all'esterno altre funzionalità, si è quindi valutato che si ha poco vantaggio a configurare un server MySQL o un'alternativa equivalente per due soli client, uno dei quali funge già da aggregatore di accessi al database.

La scelta è quindi ricaduta su SQLite, grazie anche alla sua capacità di una minimale gestione della concorrenza, che permette ad entrambe le applicazioni di accedere al database in maniera contemporanea senza dover chiudere e riaprire il file per ogni scrittura di nuovi dati.

2.2.2 Esterne al nodo

Una volta che sono state decise le tecnologie da utilizzare all'interno del nodo si è passato a definire quella che deve essere l'interfaccia con la rete esterna e con l'amministratore. Per la forte dipendenza di queste due parti della soluzione si è reso necessario valutarle contemporaneamente; dall'analisi sono emerse tre differenti alternative:

- l'impiego dello standard SNMP (Simple Network Management Protocol) e la diffusione delle informazioni presenti all'interno del database tramite questo protocollo;
- lo sfruttamento di un web server accoppiato ad un processore PHP che sia in grado di costruire delle pagine web dinamiche contenenti le informazioni sullo stato della macchina;
- la costruzione di un Web Service che permetta di gestire le richieste di visualizzazione ricevute dal client e di inviare in rete degli oggetti che rappresentano lo stato della macchina.

Segue una breve analisi delle diverse soluzioni identificate.

SNMP È un protocollo che si posiziona al livello applicativo dello stack OSI e permette ai dispositivi connessi in rete di esporre informazioni, nella forma di variabili, che descrivono la loro configurazione ed il loro stato, e ai gestori di leggere ed eventualmente modificare da remoto tali variabili.

Il suo utilizzo si basa sulla definizione e creazione di tre componenti software:

- agent: è eseguito sul nodo di rete, il suo compito è quello di mettere a disposizione i dati letti dai sensori: agisce quindi come server RPC installato sul nodo;
- manager: il processo che si occupa di inviare richieste agli agent, elabora le informazioni ed invia comandi; in questo software deve anche essere inserita un'interfaccia grafica che permetta all'amministratore di dialogare con il sistema;
- Management Information Base (MIB): il documento che descrive gli oggetti che sono alla base della comunicazione tra manager ed agents.

Questa soluzione si adatta molto bene ad un approccio centralizzato, nel quale il database storico di tutta la rete si trova sul terminale dell'amministratore: in questa situazione infatti basta inviare sulla rete delle query SNMP periodiche e dare persistenza alle risposte ricevute, poi un'analisi del database locale permette di prendere le dovute decisioni di power management; in un approccio distribuito, invece, si rende necessario un utilizzo molto particolare delle variabili definite nella MIB in modo da poter trattare anche dati storici e non solo valori attuali.

RESTful Web Server Questa soluzione prevede l'installazione su ogni nodo della rete di un piccolo software con le capacità di operare come un server web, accompagnato da un processore capace di una generazione dinamica delle pagine HTML. Un amministratore di rete può collegarsi all'indirizzo che identifica un nodo ed ottenere in risposta, con una adeguata formattazione, le statistiche presenti all'interno del database.

Tramite lo sfruttamento di tecnologie come JavaScript e AJAX è possibile fare in modo che l'interfaccia dell'amministratore possa mostrare dati relativi ad intere sezioni di rete e non ai singoli nodi, mantenendo comunque limitato il carico sulla rete stessa.

La scelta di questa soluzione prevede la scrittura di alcune pagine HTML e relativi JavaScript per definire l'interfaccia utente, e di script PHP per generare queste pagine in base ai dati presenti nel database locale. In questo contesto il server RPC è rappresentato dal web server che interpreta i file

PHP e le richieste del client per poter generare correttamente le risposte necessarie.

Web Service I Web Service forniscono un approccio ad alto livello alla comunicazione di rete. Permettono infatti di astrarre da come vengono trasferiti i dati tra client e server, focalizzando l'attenzione sul significato di ogni singola comunicazione. Questa astrazione riduce al minimo la parte di software da produrre dedicata al parsing dei pacchetti ricevuti: le API utilizzate permettono di leggere e scrivere sulla rete degli oggetti complessi come array o strutture, e non solo tipi di dati atomici.

La soluzione presenta almeno un vantaggio rispetto alle precedenti: i dati che viaggiano sulla rete sono di tipo testuale ed i protocolli utilizzati (HTTP e XML) sono aperti e disponibili praticamente ovunque, il che permette una rapida implementazione di eventuali nuovi consumatori dei dati.

Per poter sfruttare questo approccio è necessario creare un file che fornisca una descrizione rigorosa dei metodi offerti dal Web Service: questo file è scritto in un apposito linguaggio, chiamato WSDL (*Web Service Description Language*), in modo che sia interpretabile dal client in maniera automatica.

Dopo aver completato questo passo si ha a disposizione un documento che specifica rigorosamente come deve avvenire la comunicazione tra il server (nodo di rete) ed il client (amministratore), dato questo documento i due software possono avvalersi di opportune librerie che gestiscono la comunicazione di rete tramite RPC in maniera autonoma.

Confronto tra gli approcci proposti Da quanto detto fino ad ora è possibile notare che, dal punto di vista implementativo, la soluzione basata su SNMP è molto simile a quella dei Web Services, in quanto entrambe sono differenti approcci ad una soluzione client-server, per la quale devono essere sviluppate due differenti applicazioni. Il server deve interfacciarsi con il database mentre il client deve fornire l'interfaccia utente.

Differente è la soluzione del web server RESTful: in questo caso infatti è più complicato differenziare lato server e lato client dell'applicazione in quanto quasi tutta l'elaborazione (dall'interrogazione della base di dati alla costruzione della pagina HTML) è eseguita dal server, mentre il browser si limita ad interpretare e mostrare all'amministratore la pagina che riceve e ad eseguire l'eventuale codice dinamico collegato ad essa.

Se si confrontano le tre soluzioni utilizzando la metrica dei costi di implementazione si ottiene che la soluzione RESTful è la scelta più veloce: essa infatti prevede la scrittura di alcune pagine web, mentre il server (il web server) e il client (il browser) sono già disponibili. Le altre due soluzioni

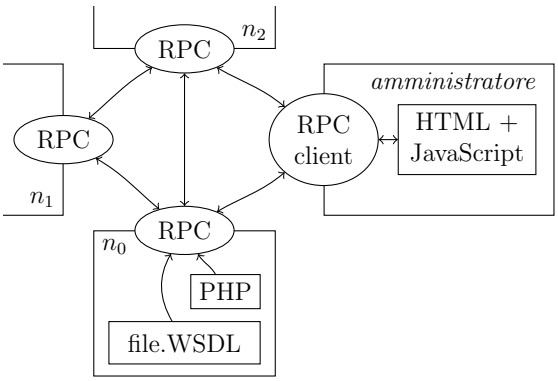


Figura 2.5: Architettura basata su Web Service con PHP e JavaScript

prevedono invece una maggiore formalizzazione del servizio e lo sviluppo di due software differenti da mettere in comunicazione.

Se però si intende utilizzare una metrica che valorizza la capacità del codice di adattarsi a future estensioni o manutenzioni si nota che la soluzione RESTful perde di interesse a favore delle altre due: se ad esempio si ipotizza di voler creare in futuro un software di analisi statistica del traffico che attraversa la rete su un periodo giornaliero o settimanale, ci si troverebbe a dover trattare quasi gli stessi dati della soluzione attuale, se questi dati però sono ricevuti dal software in formato HTML prima di poterli analizzare è necessario eseguire un parsing della pagina per separarli dalle informazioni di formattazione. Negli altri due casi invece si avrebbe a disposizione una descrizione formale dei dati e di come sono rappresentati nelle comunicazioni di rete e ci si potrebbe affidare ad una libreria in grado di convertirli direttamente in oggetti analizzabili dal software.

In Pautasso *e altri* (2008) è stato eseguito un confronto tra RESTful e Big Web Service che può essere riassunto in questo modo: la scelta RESTful si adatta molto bene a situazioni nelle quali il software è scritto *ad hoc* e per scenari semplici, mentre i Web Service sono da preferire in situazioni nelle quali è importante mantenere un'astrazione rispetto al protocollo di comunicazione sottostante e rispetto ai due capi della comunicazione.

Per questo progetto si è optato per una soluzione "ibrida" che consente di unire la formalizzazione del protocollo, permessa dal file WSDL, alla possibilità fornita dall'opzione RESTful di poter sfruttare il browser come software client: esistono delle classi PHP e JavaScript che permettono di leggere i file WSDL e fornire al resto dello script i metodi RPC in esso descritti. Queste, come mostrato in Figura 2.5 hanno permesso di implementare un'architettura basata su Web Service sfruttando HTML5, JavaScript e PHP.

Capitolo 3

Nodo di rete

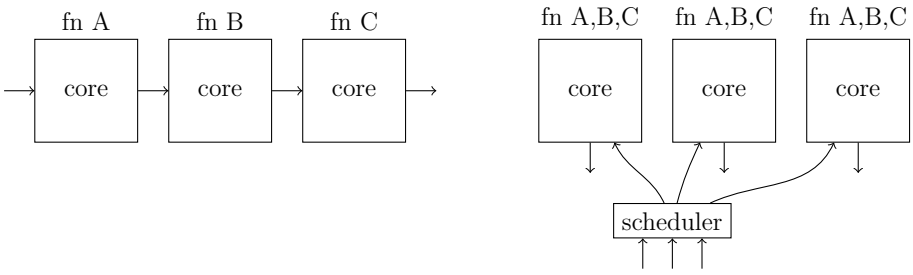
In questo capitolo è presentato il lato server della piattaforma, ovvero la parte di software che è eseguita da ogni nodo di rete ed a cui l'amministratore può collegarsi per controllare la piattaforma di power management. La trattazione attraversa innanzi tutto una parte di introduzione al multi core e di motivazioni che hanno portato alla sua scelta, successivamente sono analizzati i vari aspetti dell'implementazione del lato server: dalla comunicazione del power manager col nodo alla configurazione dell'avvio asimmetrico di due kernel, toccando infine il software che esegue il dump delle letture dei sensori all'interno del database locale ed il modello del consumo di potenza dei vari componenti del nodo, modello che deve essere utilizzato nel caso in cui alcuni sensori non siano disponibili.

3.1 Introduzione

In questo documento si definisce *nodo* della rete di telecomunicazione ogni dispositivo che è in grado di collegare tra loro diversi segmenti della rete, ovvero l'unità che, assieme ai collegamenti fisici, può essere definita come sua componente base.

Le sue funzioni sono quelle di ricevere il pacchetto dall'interfaccia collegata al segmento sorgente, compiere delle elaborazioni per verificarne la correttezza e l'integrità, ed infine inoltrarlo verso il destinatario. Nel caso il pacchetto debba attraversare più nodi per essere trasferito tra sorgente e destinazione il singolo nodo è in grado, tramite opportuni protocolli ed euristiche, di inoltrarlo solamente sul segmento di rete che lo avvicina alla destinazione.

Per compiere queste operazioni il nodo può sfruttare dell'elettronica dedicata ed un core che esegue le operazioni più ad alto livello. Proprio a causa



(a) Modello *pipeline* (b) Modello *run to completion*

Figura 3.1: Diversi approcci alla programmazione multicore

del fatto che per le elaborazioni di base non è necessario impiegare la CPU, è difficile trovare dei nodi che non siano basati su un'architettura single core, se non in casi in cui la rete ha un'importanza critica e deve sostenere continuamente volumi importanti di traffico. In Dronamraju Subramanyam (2012) sono mostrati due differenti tipi di metodi per sfruttare il multi core sugli apparati di networking: il *pipelining* dei core ed il modello *run to completion*. Nel primo (Figura 3.1a) si suddividono le operazioni in moduli indipendenti che devono essere eseguiti in maniera sequenziale (pipeline) per completare l'elaborazione sul pacchetto, in modo che l'output di un modulo rappresenti l'input del successivo. Nel secondo caso (Figura 3.1b) è invece presente uno *scheduler* che ha il compito di assegnare i pacchetti in ingresso ad uno qualsiasi dei core liberi: questi eseguono tutte le elaborazioni che sono richieste in maniera indipendente uno dall'altro.

In entrambi i casi riportati l'approccio del multi core ha l'obiettivo di migliorare le prestazioni generali del nodo di rete, possibilmente aumentando il suo throughput. La soluzione proposta non ha invece scopi prestazionali, ma intende usare il secondo core di un'architettura dual per eseguire il software di power management, che ha scopi completamente differenti rispetto a quelli del nodo.

3.2 Architetture multi core

Introduzione Le architetture multi core sono la naturale evoluzione della continua ricerca di un incremento di prestazioni dei sistemi di elaborazione. Per anni i produttori di hardware, sulla scia delle legge di Moore[1], hanno

[1]Il numero di componenti elettronici nei circuiti integrati raddoppierà ogni due anni per almeno altri dieci anni dal 1965 — Gordon E. Moore, co-fondatore Intel, 1965

spinto sulla miniaturizzazione dei transistor e sul parallelismo a livello di istruzione, per fornire agli utenti dispositivi in grado di trattare le moli di dati sempre maggiori che sono stati in grado di creare. Questo trend esponenziale è durato ben oltre il 1975: solo nel 2010 si è riconosciuto che il tasso di crescita è rallentato.

Una causa per questo rallentamento è sicuramente da attribuire a quella che viene definita come la seconda legge di Moore, o legge di Rock[2] ed il conseguente cambio di obiettivi da parte dei produttori: Intel e AMD nel 2005 lanciarono sul mercato i primi processori dual core, anticipati di un paio d'anni solo da IBM con i suoi PowerPC. In quegli anni si è raggiunta la barriera dei 3GHz di frequenza di lavoro, le ottimizzazioni che sono state applicate in precedenza erano difficilmente migliorabili, ed i consumi di potenza con i conseguenti costi di *cooling* stavano diventando decisamente importanti. Il Pentium 4 di Intel è considerato da alcuni la migliore CPU single core mai prodotta sotto il profilo architetturale, con pipeline e tecniche di prediction molto spinte ed accurate.

L'unica opportunità di migliorare ulteriormente era offerta da una maggiore parallelizzazione del carico di lavoro, suddividendolo prima su due, poi su più core.

Le soluzione multi core maggiormente diffuse sul mercato vedono la presenza di due o più unità di elaborazione perfettamente identiche: su queste architetture è possibile implementare il modello *run to completion* citato nell'introduzione. I core sono infatti in grado di eseguire le stesse operazioni ed un task può essere allocato indifferentemente su uno qualsiasi dei core disponibili.

In Merritt (2008) si propone un approccio differente, ovvero la specializzazione dei core: questo approccio segue quanto è già implementato in molti telefoni cellulari: tante unità altamente ottimizzate per l'esecuzione di un particolare compito e controllate da un gestore centrale. Questo approccio permette di ottimizzare le prestazioni, in quanto ogni modulo attivo sfrutta tutte le sue componenti per eseguire il proprio compito, altrimenti se non ha operazioni da eseguire è possibile spegnerlo completamente.

La proposta di questo lavoro di tesi non ricerca nel multi core un incremento di prestazioni, ma un isolamento tra compiti differenti. Prevede di caricare in memoria due kernel indipendenti ed assegnare un core a ciascuno: questo è chiamato avvio asimmetrico o *Asymmetric MultiProcessing* (AMP) e si contrappone all'utilizzo simmetrico delle risorse, nel quale tutto

[2]Anche i costi di ricerca e sviluppo e di produzione di circuiti integrati hanno una crescita esponenziale con ogni nuova generazione di chip — Arthur Rock, *venture capitalist* e finanziatore di molte aziende della Silicon Valley

l'hardware è controllato dallo stesso Sistema Operativo, il quale assegna le risorse ai processi che ne fanno richiesta. In una soluzione asimmetrica ad ognuno dei due kernel sono assegnate in fase di caricamento una partizione delle risorse fisiche del sistema e durante la sua esecuzione esso può accedere solamente a quell'insieme di risorse.

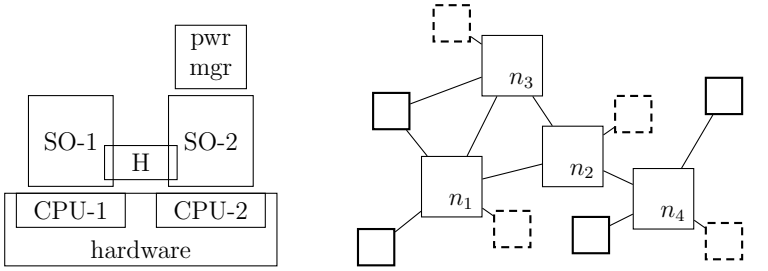
Motivazioni dell'approccio scelto Le motivazioni che hanno portato alla scelta di questo tipo di approccio sono in prima battuta di carattere industriale: la rete è già operativa e tutti i suoi nodi sono di tipo single core. Questo significa che il software che è eseguito da ognuno di essi è ottimizzato per funzionare e sfruttare al massimo questo tipo di architettura. Aggiungere nuovo software all'immagine già funzionante richiede un'accurata analisi delle performance e approfonditi casi di test per garantire la convivenza del nuovo software con i requisiti di quello attuale, mentre un avvio asimmetrico fornisce la completa trasparenza rispetto al software del nodo, garantendo una migliore stabilità di tutto il sistema proprio per il fatto che le modifiche da apportare all'immagine presente sono minime.

Una seconda motivazione prende in considerazione quelli che possono essere gli sviluppi futuri che può avere la piattaforma di power management. In un possibile futuro il software della soluzione potrebbe diventare arbitrariamente complesso e compiere delle predizioni di carico sulla rete analizzando lo storico a sua disposizione; queste predizioni potrebbero richiedere capacità computazionali importanti, ed in momenti particolari potrebbero far venir meno le funzionalità di base del nodo, portando ad un *denial of service* di una parte della rete. In uno scenario alternativo invece il nodo di rete si potrebbe arricchire di una CPU dedicata e dimensionata in maniera tale da soddisfare esattamente i requisiti della politica di power management implementata, sulla scia dell'approccio citato in Merritt (2008) dei core specializzati. In questo caso l'avvio asimmetrico diventerebbe necessario per poter sfruttare appieno questa particolare CPU.

3.3 Comunicazione tra i due software

Per consentire al power manager di svolgere il suo compito è necessario permettergli di comunicare con il software del nodo. I due sistemi operativi che si intende avviare non sono configurabili per assistere questo tipo di comunicazione attraverso un'area di memoria condivisa, in quanto la *InterProcess Communication* (IPC) che sono in grado di gestire riguarda processi che sono in esecuzione all'interno dello stesso sistema operativo.

È già stato detto che il software del nodo espone verso la rete una shell



(a) Ruolo dell'hypervisor nella comunicazione

(b) Schema logico della rete dopo l'aggiunta dei power manager

Figura 3.2: Aggiunta dei power manager alla rete

alla quale un amministratore può accedere da remoto per modificare la configurazione del nodo. Questa è la via che è stata scelta per consentire al power manager di inviare i suoi comandi: stabilire una connessione verso quella shell e sfruttarla per indicare al nodo di attivare o spegnere alcune delle proprie interfacce.

Tramite un hypervisor dedicato è possibile evitare il passaggio attraverso un'interfaccia di rete esterna per stabilire la comunicazione, in quanto questo permette di creare un'interfaccia virtuale per ogni kernel e fare in modo che queste possano comunicare tra loro. La Figura 3.2 mostra come cambia la rete con l'aggiunta dei power manager: nella 3.2a si evidenzia come l'hypervisor (H) si pone tra i due Sistemi Operativi per garantire la comunicazione senza l'impiego di interfacce esterne, mentre la 3.2b mostra come cambia a livello logico la rete dopo l'aggiunta dei power manager.

Dal punto di vista della rete infatti la situazione che si crea è simile a quella che si avrebbe aggiungendo un nodo terminale per ogni nodo di comunicazione. Nell'esempio mostrato in Figura 3.2b i nodi etichettati con n_i sono quelli di comunicazione, mentre con un tratto più marcato sono mostrati i nodi terminali. Quelli tratteggiati sono infine i power manager, uno per ogni nodo di comunicazione, che si pongono come ulteriori terminali ed utilizzatori della rete, in quanto generano dell'ulteriore traffico che deve essere gestito.

3.4 Avvio asimmetrico del nodo

La strada scelta per l'implementazione della soluzione è quella di mantenere attivi due kernel differenti per disaccoppiare il software di gestione del nodo con l'ottimizzazione energetica. Per consentire questo tipo di approc-

cio, ammessa la presenza di un hardware con almeno due core, vi sono due prerequisiti che devono essere soddisfatti:

- un bootloader che supporti questa funzionalità, che permetta cioè di caricare in memoria i due kernel e di impartire comandi di avvio ad una CPU per volta;
- una coppia di kernel correttamente configurati prima della compilazione in modo che non tentino entrambi di inizializzare tutto l'hardware presente sulla macchina.

3.4.1 Bootloader

Il primo requisito è soddisfatto da U-Boot[3], il bootloader già utilizzato per i nodi di rete e largamente sfruttato in ambito embedded per la sua ampia configurabilità. In questo bootloader sono già presenti tutte le funzionalità necessarie ad adattarsi agli impieghi più disparati: al suo interno è possibile trovare client DHCP e TFTP e supporto ai protocolli BOOTP e TFTPBOOT per caricare un kernel presente su un dispositivo remoto in rete, gestione della memoria flash e delle partizioni linux per poter caricare un kernel locale e supporto al trasferimento di file attraverso la linea seriale per situazioni di debug o di testing prima della messa in opera dell'hardware. Ha anche il supporto per compiere una verifica CRC dei dati caricati in memoria.

Il suo comportamento di default è quello di presentare una console sulla porta seriale attraverso la quale l'utente può interagire con il bootloader. Tramite appositi comandi e variabili d'ambiente è possibile caricare in memoria ad indirizzi specifici le immagini del kernel e del filesystem iniziale e avviare il Sistema Operativo. U-Boot ha un supporto anche per script minimali e la possibilità di eseguire delle istruzioni di default, senza l'intervento dell'utente sulla porta seriale: questi permettono di caricare un kernel in maniera autonoma, e possono essere utilizzati quando il sistema esce dal testing ed entra in fase produttiva.

3.4.2 Device Tree

Per fornire al kernel l'elenco dei dispositivi hardware presenti sul nodo sono stati creati i *Flattened Device Tree* o FDT, dei file binari che descrivono l'hardware sul quale il kernel deve operare. Quest'ultimo può utilizzare tali file per trovare e registrare le varie periferiche del sistema, dato che al loro interno sono presenti anche gli indirizzi ai quali le periferiche sono collegate.

[3] `http://www.denx.de/wiki/U-Boot/WebHome`

Tramite i device tree è possibile descrivere molti aspetti di design architetturale, come ad esempio numero e tipo di CPU, gli indirizzi dedicati alla memoria RAM, gli interrupt hardware e le connessioni con le periferiche.

L'approccio classico alla problematica della ricerca dell'hardware a disposizione prevede che all'interno del kernel sia compilato del codice di inizializzazione che esegue del probing su indirizzi noti per verificare se un determinato hardware è presente e risponde correttamente all'indirizzo testato.

I device tree, come alternativa, hanno il vantaggio di essere una struttura dati standard e formale, ed in grado di descrivere praticamente tutto l'hardware che dovrebbe essere rilevato tramite probing. Essi possono sostituire questa procedura, ed hanno almeno due risvolti positivi: da un lato i produttori di hardware possono focalizzare le proprie risorse sulla stesura di driver di Sistema Operativo per i loro prodotti, evitando di dover includere al loro interno anche la procedura di ricerca ed identificazione della periferica; dall'altro i kernel compilati possono essere utilizzati anche su hardware creato ed aggiunto a posteriori in quanto tramite i device tree è possibile sostituire il codice di probing. Questo permette quindi di ridurre il quantitativo di codice *hardware dependand* ed aumentare la genericità del kernel.

Se necessario è possibile fornire al kernel in fase di avvio un device tree che descrive solamente un sottoinsieme delle periferiche effettivamente presenti, in questo casi esso si limiterà ad inizializzare e sfruttare solamente quelle che vi sono elencate, ignorando l'esistenza delle altre. Questa flessibilità è sfruttabile per completare l'avvio asimmetrico dei due kernel: è infatti sufficiente partizionare l'hardware e creare un device tree per ognuno dei due sottoinsiemi della partizione, infine tramite appositi comandi impartiti al bootloader è possibile caricare in sequenza i kernel sui due core passandogli il corretto device tree.

In Appendice B sono presenti degli ulteriori dettagli riguardanti l'avvio asimmetrico di due kernel sullo stesso hardware e le operazioni preventive che è importante eseguire.

3.5 Software di dump

Componente di base della soluzione è rappresentata dal dumper, ovvero quel software, in esecuzione su ogni nodo di comunicazione della rete, che periodicamente registra nel database locale le informazioni relative al consumo di potenza di ogni componente del nodo. Senza la presenza di questo software le politiche implementabili sarebbero molto limitate in quanto non avrebbero accesso ai dati storici della rete.

Il software di dump, eseguito sul core del power manager, è perfettamente

trasparente rispetto al funzionamento normale del nodo. Questo è possibile grazie innanzi tutto alla scelta di dove allocare tale processo, che consente di lasciare liberi memoria e tempo di processore dedicati al software del nodo, richiedendoli invece al power manager, il quale non ha gli stessi requisiti di prestazioni del primo. In secondo luogo i valori di consumo di potenza e larghezza di banda utilizzati sono letti da appositi sensori hardware, che permettono di ottenere tali valori senza dover interrompere la periferica o il software che la sta utilizzando. L'elettronica dei sensori si preoccupa di mantenere aggiornate delle aree di memoria con gli stessi valori rilevati dalla periferica, in modo che siano leggibili dal dumper come fossero semplici variabili condivise.

Gli unici requisiti per il corretto funzionamento di questo software sono quindi la possibilità di accedere alla zona di memoria dove sono mappati i valori dei sensori e di scrivere dei dati in un database situato su un supporto di archiviazione, possibilmente non volatile, locale al nodo. Soddisfatti questi requisiti il software ha il solo compito di scandire, ad intervalli di tempo regolari e preconfigurati, tutte le aree di memoria che contengono i valori aggiornati dai sensori e, una volta ottenute tali misurazioni, di inserirle all'interno del database.

Al momento dell'avvio il dumper ha la necessità di conoscere le periferiche presenti in hardware ed i sensori dai quali è possibile ricavare le informazioni che deve trattare. Per fare ciò il software analizza il contenuto di un file XML che gli è fornito: tale file è simile a quello mostrato in Appendice A, e deve rispettare il DTD allegato. In esso sono presenti tutte le informazioni di cui necessita il dumper:

- la struttura hardware del nodo per quanto riguarda le componenti interessate dalle politiche di power saving, con espressa anche la loro gerarchia;
- per ognuna delle componenti descritte sono inoltre specificati:
  - l'identificativo di rete, ovvero quella stringa, di cui si è trattato nell'introduzione, che permette di identificare univocamente il singolo dispositivo all'interno dell'intera rete;
  - i parametri di funzionamento, in particolare la larghezza di banda massima ed i suoi consumi di potenza statico e dinamico, in modo che in mancanza del supporto dei sensori hardware sia possibile ricavare ugualmente dei valori da inserire nel database;
  - la lista di tutti i sensori disponibili, con specificato il tipo di grandezza misurata, potenza o larghezza di banda, e l'indirizzo di memoria al quale è possibile ottenere copia del suo valore.

Date queste informazioni il software è in grado di creare delle query di inserimento nel database contenenti i valori letti dai sensori o generati dal modello ed il timestamp della lettura[4]. Dato che la rete può essere distribuita anche attraverso fusi orari differenti si è deciso di fare in modo che tutti i record in tutte le basi di dati all'interno della rete abbiano il timestamp espresso rispetto al tempo universale coordinato (UTC). Questa decisione è stata presa anche alla luce del fatto che non sono stati posti vincoli all'accesso dell'eventuale amministratore, di conseguenza esso può analizzare e modificare lo stato della rete anche da fusi orari molto distanti da quelli nei quali la rete è ubicata.

La natura distribuita della base di dati pone anche una problematica di sincronizzazione dei nodi di comunicazione. Innanzi tutto è necessario assumere che l'orologio di sistema dei power manager dei vari nodi sia sincronizzato almeno all'interno della rete, in modo che tutte le registrazioni compiute in un determinato istante siano etichettate con lo stesso timestamp. In secondo luogo si è ritenuto importante che le registrazioni nei vari database fossero il più possibile sincronizzate in modo che, se l'amministratore richiede lo stato di una porzione della rete in un determinato istante, tali informazioni siano tutte reperibili ed etichettate con lo stesso timestamp, senza che questo abbia delle variazioni di qualche secondo tra un nodo ed un altro. Questo significa che il software di dump, una volta terminata l'analisi del file XML, procede con la lettura dei sensori ed il conseguente primo inserimento nel database al secondo zero del minuto successivo a quello attuale.

3.6 Modello del consumo di potenza

Nel caso in cui alcuni sensori non fossero disponibili il software di dump è predisposto per calcolare il valore mancante sulla base delle informazioni che sono disponibili, sfruttando un modello compilato all'interno del codice; parte di questo modello è già stato presentato nella Sezione 1.3 a pagina 4. Tale modello è presentato completamente in questa sezione.

Le componenti statiche di un nodo di comunicazione, come PSU e FAN, hanno chiaramente due soli livelli di consumo di potenza e non hanno una larghezza di banda utilizzata. Per le altre componenti si è adottata l'identificazione tramite pedici, ovvero ad ogni variabile è stato aggiungo a pedice la lettera iniziale del tipo di componente associato: I per le interfacce, L per le line card e M per le matrix. Quindi ad esempio il consumo di potenza di una

[4]Il timestamp è espresso come stringa testuale aderente allo standard ISO 8601:2004, secondo la raccomandazione W3C, ovvero nel formato `YYYY-MM-DDTHH:MM:SSZ`

line card sarà etichettato come P_L, mentre la larghezza di banda utilizzata all'istante t di una matrix sarà $BW_M(t)$.

Iniziando dal calcolo della larghezza di banda utilizzata si assuma che sia disponibile quella di ogni interfaccia di rete, allora per ogni line card essa sarà pari alla somma di quella di ogni sua interfaccia.

$$BW_L(t) = \sum_{i \in L} BW_i(t)$$

Dato che le matrix gestiscono il traffico tra una line card e l'altra si ha che tutto il traffico che attraversa il nodo deve essere gestito dalla line card attiva: quindi la sua larghezza di banda utilizzata è pari alla somma di tutte quelle di tutte le line card del nodo.

$$BW_M(t) = \sum_{l \in N} BW_l(t)$$

Date le informazioni sulla larghezza di banda utilizzata è possibile calcolare il consumo di potenza delle interfacce di rete e delle matrix con lo stesso modello mostrato in precedenza a pagina 4.

$$P_I(t) = P_{IS} + P_{ID} \cdot \frac{BW_I(t)}{MBW_I}$$

$$P_M(t) = P_{MS} + P_{MD} \cdot \frac{BW_M(t)}{MBW_M}$$

Per quanto riguarda le line card è invece necessario evidenziare che le interfacce sono dei loro componenti, di conseguenza il loro consumo di potenza contribuisce direttamente al consumo della line card a cui appartengono.

$$P_L(t) = \sum_{i \in L} P_i(t) + P_{LS} + P_{LD} \cdot \frac{BW_L(t)}{MBW_L}$$

Il consumo di potenza dell'intero nodo infine sarà dato dalla somma di quello di ogni componente statico presente (C), delle matrix e delle line card. Le singole interfacce non sono considerate in questo computo in quanto il loro consumo di potenza è già inserito nel calcolo di $P_L(t)$.

$$P_N(t) = \sum_{c \in N} P_c(t) + \sum_{l \in N} P_l(t) + \sum_{m \in N} P_m(t)$$

Come è facile immaginare i consumi di potenza dei componenti statici non dipendono dalla larghezza di banda attualmente utilizzata, il loro valore è in funzione della variabile temporale t solamente per il fatto che tali componenti potrebbero essere spenti, ed in questo caso il loro consumo sarebbe nullo.

Capitolo 4

Comunicazione

Requisito di base affinché tutta la piattaforma di power management sia in grado di funzionare risiede nella possibilità che devono avere i vari manager di comunicare tra loro e con l'eventuale amministratore. Questo perché l'informazione presente nel database di ogni nodo ha un valore molto limitato se non può essere associata a quella dei nodi circostanti, ed è inoltre importante che l'amministratore possa avere accesso a queste informazioni di performance per poter gestire adeguatamente la rete.

Per garantire questa comunicazione, sia tra nodi che con l'amministratore, sono stati scelti i Web Service. In questo capitolo è presentata una descrizione di tale piattaforma, con i suoi vantaggi e le sue limitazioni, ed un approfondimento sul suo componente principale, il formato WSDL. Successivamente sono descritti i due endpoint con la loro struttura interna infine, nell'ultima sezione, è presentato il client realizzato.

4.1 Web Service

Secondo la definizione del World Wide Web Consortium (W3C) un Web Service è un sistema software progettato per fornire un'interoperabilità standard tra differenti applicazioni, in grado di funzionare su una varietà di piattaforme e framework[1]. Questa descrizione si adatta bene ad evidenziare le potenzialità della piattaforma: innanzi tutto è uno standard W3C con un gruppo di lavoro dedicato che si preoccupa di aggiornare e migliorare le sue specifiche per inseguire e guidare l'evoluzione del web; in secondo luogo è una soluzione che non dipende da una particolare piattaforma software o di sviluppo per poter funzionare, è quindi accessibile ad una grande varietà di ambienti implementativi. Il suo punto di forza rispetto alle alternative sta nel fatto

[1] `http://www.w3.org/TR/ws-arch/#whatis`

che il suo layer di trasporto è composto di messaggi in formato XML che, essendo testuale e non binario, è facile da generare ed interpretare in qualsiasi ambiente software.

Le sue potenzialità nascono dalla grandissima diffusione della rete Internet, tramite la quale il Web Service trova la sua naturale implementazione, ma nel contempo non è vincolato ad utilizzarla: anche una rete dedicata o due software all'interno dello stesso Sistema Operativo sono in grado di gestire ed utilizzare un Web Service tramite un socket locale. Esso infatti, come già accennato, utilizza il formato XML per i messaggi e la porta HTTP per le connessioni: questa porta è lasciata aperta da quasi tutti i firewall, infatti è quella dedicata alla navigazione web. Questo significa che anche in ambito enterprise non è necessaria la riconfigurazione di complicate politiche di sicurezza della rete, ma al contrario la comunicazione è già pronta all'uso.

Un'altra peculiarità dei Web Service risiede nel disaccoppiamento che sono in grado di fornire tra i due software ai capi della comunicazione: come nello stile delle *Remote Procedure Call* il client invoca un metodo locale della libreria che gestisce la comunicazione fornendogli alcuni parametri, tale metodo si occupa di inviarli al server, sul quale è compiuta l'elaborazione, e di ritornare i risultati ricevuti al chiamante, rendendogli del tutto trasparenti le problematiche della comunicazione. All'altro capo del canale, sul server, si trovano le implementazioni dei metodi che sono messi a disposizione del client, tali metodi sono invocati dalla libreria di comunicazione nel momento in cui un messaggio è ricevuto dalla rete.

4.2 Struttura di un Web Service

Un'architettura basata su Web Service è solitamente suddivisa in tre macro componenti:

- un insieme di client del servizio, detti *Service Requester*, che necessitano e richiedono delle prestazioni fornite dai server;
- un insieme non vuoto di server, detti *Service Provider*, su cui sono implementati i metodi che possono essere richiesti dai client;
- almeno un *Service Broker*, che ha funzione di indirizzamento dei provider: su richiesta può fornire una descrizione di tutti i servizi messi a disposizione dalla rete e delle modalità con le quali possono essere invocati.

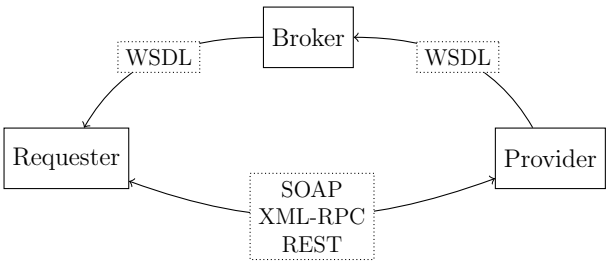


Figura 4.1: Architettura generale di un Web Service

L'architettura standard di un Web Service è mostrata in Figura 4.1, nella quale i nodi rappresentati con tratto continuo sono i tre attori della comunicazione, mentre sugli archi sono indicati i vari protocolli utilizzati. È possibile notare che, mentre le comunicazioni con il broker utilizzano obbligatoriamente il formato WSDL, appositamente progettato per questo scopo, quelle tra requester e provider non hanno un protocollo specifico da rispettare ma esistono più alternative possibili. All'interno del file WSDL fornito dal broker è specificato, tra le altre informazioni, quale tra i protocolli alternativi va utilizzato da un requester per potersi mettere in comunicazione con un particolare service provider.

Da evidenziare è il fatto che tutte le comunicazioni di questa architettura, dai WSDL trattati dal broker alla comunicazione tra requester e provider, sono in formato testuale. Questo ha il grosso vantaggio di essere facilmente e quasi sempre interpretabile in maniera univoca, a meno di differenze di encoding dei caratteri, da qualsiasi software, indipendentemente dalla piattaforma sulla quale stanno funzionando i due endpoint: questo garantisce la massima interoperabilità tra software di natura anche eterogenea.

Non è inoltre necessario che le tre componenti elencate siano fisicamente separate tra loro: ad esempio in situazioni nelle quali l'indirizzo del provider è noto a priori non c'è la necessità di creare un broker esterno, semplicemente ogni provider agisce da broker di sé stesso. Inoltre anche la separazione tra provider e requester spesso non è ben definita: se si considera l'esempio di Figura 4.2 il software sul nodo A, dopo aver ricevuto la descrizione del servizio fornito dal software di B, invia a quest'ultimo una richiesta di esecuzione di un metodo `f()`. Per poter completare questa esecuzione però B necessita di una chiamata al metodo `g()` che è fornito dal software sul nodo C. In una situazione come quella descritto è facile identificare A come requester e C come provider, mentre B si pone sia come provider di `f()` che come requester di `g()`.

Questo esempio mostra come sia possibile integrare tra loro servizi web distribuiti in tempi e modalità differenti: per estendere una funzionalità fornita

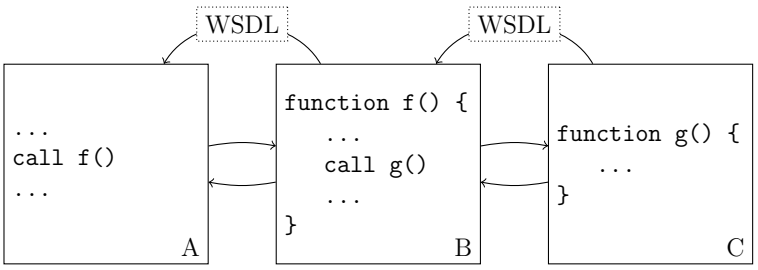


Figura 4.2: Esempio di doppia RPC

da un software è sufficiente infatti contattare il broker corretto e scegliere, tra le alternative proposte, la descrizione del servizio che meglio si adatta alle necessità di estensione richieste e, con questa, configurare la libreria di interfaccia con la rete per iniziare immediatamente a sfruttare il nuovo servizio.

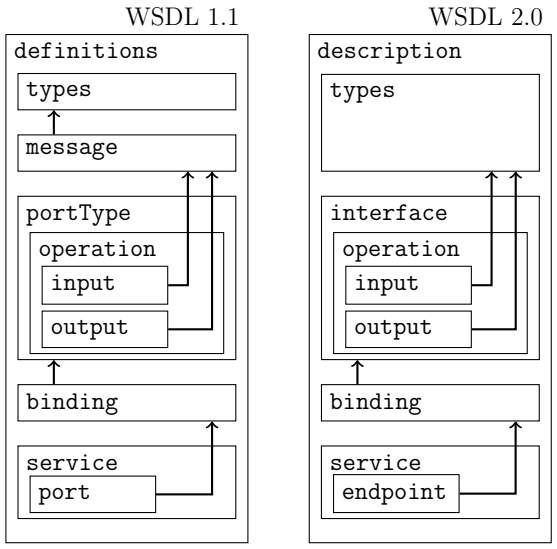
Per questa soluzione software ci si è appoggiati ai Web Service, sfruttando lo standard HTTP come mezzo di trasporto e SOAP (*Simple Object Access Protocol*) come contenitore dei messaggi scambiati. Quest'ultimo definisce il formato che deve avere il pacchetto all'interno del quale sono posti i messaggi, ovvero specifiche strutture basate su XML.

4.3 File WSDL

Il Web Service Description Language (WSDL) è il linguaggio nel quale è scritto il file cuore della struttura dei Web Service. Tale file, infatti, è distribuito dal broker ai requester e contiene una descrizione di tutta la struttura e delle componenti del servizio: a partire dai tipi di dati impiegati fino agli indirizzi all'interno della rete ai quali è possibile raggiungere determinate procedure. Questo file è suddiviso in cinque sezioni, ognuna dedicata a descrivere un particolare aspetto del Web Service, che devono essere specificate in questa sequenza:

1. I tipi di dati (`types`) che possono essere trasmessi sulla rete incluse, se necessarie, strutture di arbitraria complessità e dimensione. Per la loro definizione è utilizzato XML-Schema, i DTD non sono applicabili in questo ambito a causa della loro impossibilità di definire tipi di dati e per il fatto che la loro sintassi non è in formato XML.

2. I messaggi (`message`) che requester e provider possono scambiarsi ed i relativi dati che devono accompagnarli. Tali dati possono essere atomi-



WSDL 1.1	WSDL 2.0
`types`	`types`
`message`	*n. d.*
`portType`	`interface`
`operation`	`operation`
`binding`	`binding`
`service`	`service`
`port`	`endpoint`

Figura 4.3: Confronto tra le specifiche di formati di WSDL 1.1 e 2.0

ci, come stringhe o valori numerici, oppure istanze di strutture definite all'interno della sezione types.

3. I gruppi (`portType`) di metodi implementati sui vari provider, ognuno dei quali raccoglie sotto lo stesso nome un insieme di operazioni (`operation`) che sono in qualche modo correlate tra loro. Per ogni metodo sono inoltre specificati i messaggi in ingresso e in uscita utilizzati per completare l'elaborazione.

4. Il tipo di collegamento (`binding`) tra i portType definiti precedentemente ed il protocollo da utilizzare per la comunicazione. È possibile avere più binding per un singolo portType, ad esempio nel caso in cui su due server differenti è implementato lo stesso insieme di metodi ma sono utilizzati protocolli differenti per la comunicazione. In questa lavoro si è deciso di sfruttare il protocollo SOAP per tutti i metodi, di conseguenza i binding sono stati utilizzati solamente per rendere esplicita questa scelta.

5. I servizi (`service`) che raccolgono un set di metodi messi a disposizione dalla rete. Al loro interno sono elencati i punti di accesso (`port`) per i vari binding che sono stati definiti, i quali consistono solamente in un URL al quale è possibile raggiungere il servizio.

Con la versione 2.0 il formato WSDL ha subito alcune revisioni sintattiche, alcune delle quali sono mostrate nella Figura 4.3. La più evidente di

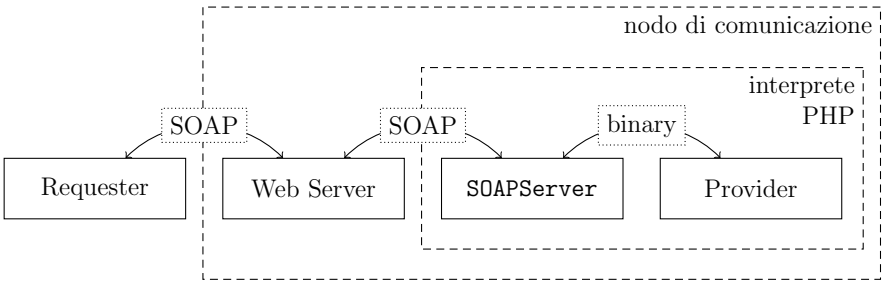


Figura 4.4: Schema di comunicazione tra requester e metodi provider implementati in una classe PHP

queste modifiche è la scomparsa degli elementi `message`: con la versione 1.1 del protocollo ogni `operation` ha uno ed un solo elemento di `input` che fa riferimento ad un particolare `message`, il quale elenca i parametri da fornire alla `operation`. In molti casi però questo porta ad avere un mapping 1:1 tra le `operation` ed i `message`, di conseguenza nella versione 2.0 del protocollo si è deciso di eliminare questa ridondanza e permettere la creazione di più parametri di `input` e `output` direttamente all'interno dei nodi `operation`, ognuno dei quali fa riferimento ad un particolare dato atomico o definito in `types`.

Questa ed altre modifiche, apportate per aumentare la flessibilità e la semantica del linguaggio con lo scopo di renderlo più facilmente utilizzabile agli sviluppatori, hanno permesso al formato WSDL di divenire, nel giugno 2007, uno standard raccomandato dal W3C[2]. Purtroppo il supporto di WSDL 2.0 da parte di PHP è ancora limitato, di conseguenza per questo lavoro è stato creato un file aderente alla versione 1.1 del formato, in modo che potesse essere correttamente interpretato dalle classi impiegate.

4.4 Implementazione in PHP

I metodi del nodo di comunicazione, inteso come provider del Web Service, sono stati implementati in PHP, in modo da poter sfruttare un'architettura come quella mostrata in Figura 4.4. In essa i messaggi SOAP inviati dal requester verso il provider sono ricevuti da un piccolo web server, come ad esempio lighttpd[3], che si occupa di inoltrarle all'interprete PHP ed alla sua classe `SOAPServer`, creata appositamente per gestire il lato provider di questo tipo di comunicazione. Tale classe si occupa di invocare i metodi del Web

[2] `http://www.w3.org/TR/wsdl20/`
[3] `http://www.lighttpd.net`

Service indicati nel messaggio SOAP ricevuto, tali metodi sono implementati all'interno di un'altra classe PHP creata ad hoc.

In questa figura è evidenziato il tipo di formato che hanno i vari messaggi che si scambiano gli attori: a partire dal requester fino alla classe `SoapServer` tale formato rimane SOAP, poi questa si occupa di interpretare il messaggio e convertire i parametri in valori binari, in modo che possano essere utilizzati dalla classe provider del Web Service. In direzione contraria i valori ritornati da tale classe sono passati a `SOAPServer`, la quale si occupa di convertirli in formato testuale ed impacchettarli all'interno di un frame SOAP che sarà successivamente inviato al requester tramite il web server.

Questo mette in evidenza come la classe che contiene i metodi del Web Service è completamente agnostica delle modalità con le quali tali dati viaggiano sulla rete: essa si occupa solamente di gestire delle chiamate ricevute dalla classe `SOAPServer` che le è stata posta come interfaccia con il resto dell'architettura.

4.5 Implementazione in JavaScript

Ad esempio dell'interoperabilità messa a disposizione dai Web Service è possibile prendere la piattaforma di management fornita all'amministratore della rete: questa si basa su HTML e JavaScript ma, grazie al protocollo SOAP, è in grado di richiedere l'esecuzione di procedure remote scritte in PHP e trattare dati ritornati come se l'esecuzione fosse locale.

La classe `soapclient` impiegata ha dovuto subire alcuni adattamenti in quanto è stata creata con lo scopo di funzionare su specifici WSDL generati automaticamente da server quali asp.net e altri. È stato fatto un primo lavoro di generalizzazione della classe per avvicinarla maggiormente al funzionamento standard come interfaccia SOAP, ma parte del lavoro non è ancora completa.

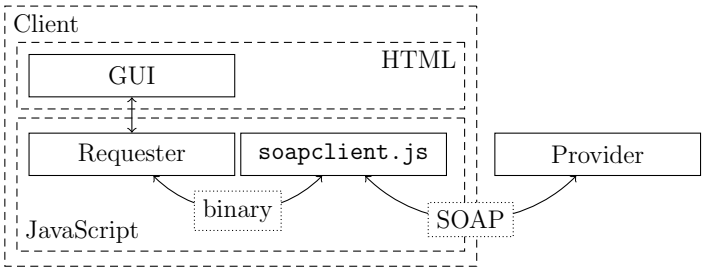


Figura 4.5: Schema di comunicazione tra provider del servizio e logica del requester implementata in JavaScript

La struttura del lato client della comunicazione è mostrata in Figura 4.5. Il requester è simile al provider già descritto (Figura 4.4), con la classe `soapclient` che si pone come interfaccia per il protocollo SOAP ed il codice del requester che controlla l'interfaccia utente utilizzata dall'amministratore. In entrambe le immagini è stato volutamente ridotto ad un solo blocco l'endpoint con il quale il software comunica, per sottolineare il fatto che, se è rispettato lo standard, non è necessario essere a conoscenza della natura dell'interlocutore per il corretto funzionamento della piattaforma.

4.6 Interfaccia con l'amministratore

Deciso di utilizzare gli strumenti appena descritti per le comunicazioni di rete, le scelte riguardanti il client da fornire all'amministratore sono state orientate verso il software più aderente agli standard del W3C e presente in tutti i sistemi operativi: il browser Internet. Questa scelta è stata presa anche sulla base delle osservazioni che vedono la sempre maggiore presenza di applicazioni fruibili tramite questo tipo di software ed anche la nascita di interi Sistemi Operativi consumer di tipo browser based: queste osservazioni permettono di predire che nel prossimo futuro questo strumento non verrà a mancare all'interno di un generico Sistema Operativo, ma al contrario sarà sempre maggiormente soggetto ad ottimizzazioni e miglioramenti. Questa scelta apre inoltre la porta a possibilità future di estensione della piattaforma in maniera tale che sia utilizzabile anche in mobilità, su dispositivi con ridotta risoluzione come smartphone, tablet e netbook.

4.6.1 Requisiti funzionali

Per consentire all'amministratore di analizzare le informazioni ottenute dai sensori della rete è necessario che questo sia a conoscenza della sua topologia. Per questo lavoro si è assunto che l'amministratore abbia già a disposizione questa informazione, e che sia in grado di accedere direttamente al power manager di un nodo qualsiasi, conoscendone l'indirizzo di rete. Dato questo prerequisito è possibile semplificare il compito dell'interfaccia utente ad un semplice reporting di quanto richiesto dall'amministratore, senza porre il requisito di istruire quest'ultimo a riguardo della topologia reale della rete.

Fatta questa premessa è possibile definire un insieme di requisiti funzionali che un'interfaccia completa deve soddisfare. Innanzi tutto deve essere permesso all'amministratore di ottenere l'andamento del consumo di potenza e/o della larghezza di banda utilizzata in un periodo fissato, ad esempio durante l'ultima mezz'ora. Queste informazioni possono essere visualizza-

te in formato testuale, oppure mediante un grafico; quest'ultima soluzione permette di avere un migliore colpo d'occhio su eventuali picchi massimi o minimi di utilizzo che potrebbero suggerire la necessità di una riconfigurazione dell'architettura della rete intorno a quel nodo.

Un altro requisito che è stato posto è quello della cosiddetta *sliding window* sui dati mostrati nei grafici, ovvero fare in modo che questi ultimi possano aggiornarsi periodicamente ed in maniera automatica, mostrando i nuovi dati registrati nel database dai vari dumper presenti sulla rete e nascondendo i dati troppo vecchi. Questo permette di dare all'amministratore l'idea della dinamicità della rete, e gli consente di seguire la sua evoluzione senza dover inviare manualmente nuove query ad ogni periodo di dump. Questa sliding window non deve però essere mandatoria, ma un'opzione che l'amministratore può attivare o rimuovere: se infatti la sua intenzione è quella di concentrarsi su un particolare intervallo di tempo il grafico non deve spostarsi da tale intervallo solamente per il fatto che sono disponibili nuovi dati.

All'interno dello stesso grafico deve essere inoltre possibile richiedere la visualizzazione di più di una grandezza: ad esempio potrebbe essere necessario per l'amministratore valutare l'andamento della larghezza di banda utilizzata di due interfacce dello stesso nodo, per ricercare una correlazione tra le due. Questo potrebbe portare ad osservare che il traffico che le attraversa è in qualche modo correlato e che un'interfaccia è sufficiente a sostenere il carico di entrambe senza introdurre eccessivi ritardi nelle comunicazioni in atto. Attenzione deve essere posta a non confrontare grandezze di natura diversa: non ha infatti senso mettere a confronto una larghezza di banda con un consumo di potenza in quanto tali dati, anche se direttamente proporzionali, non hanno valori paragonabili.

Se sulla rete sta funzionando un power manager automatico l'amministratore deve inoltre essere in grado di prendere visione del suo comportamento, possibilmente ottenendo un logfile delle operazioni eseguite in un determinato periodo di tempo. Questo log potrebbe dover essere confrontato con quello di altri nodi nel caso sia avvenuta una comunicazione tra i power manager, oppure con le informazioni ottenute dai sensori durante il medesimo periodo di tempo.

Motivo di creazione di questa interfaccia è anche quello di permettere all'amministratore di avere una parte attiva sulla piattaforma di power management. In mancanza del manager automatico l'amministratore deve essere in grado di modificare i parametri di funzionamento di qualsiasi nodo della rete, mentre in presenza del software di gestione deve avere la facoltà di modificare il suo comportamento cambiando alcune variabili come ad esempio la sua aggressività nel power saving, oppure forzando alcuni elementi della rete in modo che rimangano costantemente attivi o spenti.

4.6.2 Client JavaScript

I requisiti sopra elencati devono essere rispettati da un'interfaccia di amministrazione completa e perfettamente funzionante della piattaforma di power management. Quanto implementato fino ad ora, come già descritto nell'introduzione, è un'interfaccia che invece ha il solo scopo di debug e visualizzazione dello stato e della storia della rete, senza la possibilità di compiere azioni attive.

Le funzionalità che sono state implementate sono di interrogazione della base di dati distribuita: l'amministratore può inserire l'identificativo di un particolare dispositivo e scegliere per esso la grandezza da richiedere al database con il tipo di aggregato da applicare a tale grandezza. Attualmente i valori salvati all'interno del database sono la larghezza di banda utilizzata ed il consumo di potenza, rispettivamente espressi in Gb/s ed in Watt. Gli aggregati che sono calcolati automaticamente dal motore di database sono il valore medio ed i due limiti minimo e massimo. Una volta che l'amministratore ha compilato una richiesta contenente i dispositivi e le informazioni da visualizzare per ognuno di essi, tramite un apposito pulsante può lanciare l'operazione di query. Questa sarà inviata per intero al nodo di rete dal quale è stata scaricata l'interfaccia, successivamente questo dovrà suddividerla in modo da poterla inoltrare ai nodi che contengono le informazioni richieste e, una volta ottenute le risposte, dovrà essere in grado di aggregare i dati e fornire un'unica risposta all'interfaccia di amministrazione. Per la visualizzazione delle informazioni ricevute l'interfaccia crea un grafico appoggiandosi alla libreria d3js[4]. Questa permette di creare i cosiddetti *Data-Driven Documents*, ovvero delle pagine HTML completamente dinamiche e modificabili sulla base dei dati da mostrare all'utente: le sue funzionalità non sono limitate alla creazione di grafici, ma ad esempio consentono la costruzione, direttamente nel DOM della pagina HTML, di tabelle contenenti i dati da mostrare. Per funzionare sfrutta tecnologie standardizzate dal W3C, come HTML per la struttura dei documenti, SVG per le componenti grafiche e CSS per definire lo stile di ogni elemento.

[4] `http://d3js.org`

Capitolo 5

Risultati ottenuti

Durante lo sviluppo della soluzione non è stato possibile eseguire test sull'hardware target in quanto non sono stati resi disponibili nodi di rete sui quali eseguire il software. Gli unici test possibili hanno riguardato l'avvio asimmetrico di una macchina dual core basata su architettura PowerPC™. Si è scelto questo tipo di architettura in quanto è quella che più si avvicina al target di funzionamento pianificato. Un primo periodo di testing è stato fatto per valutare il corretto funzionamento dell'avvio asimmetrico di due Sistemi Operativi, dopo aver ottenuto la compilazione di due kernel e completato la stesura di due device tree ad hoc per la macchina di test impiegata. Da questa attività sono state estratte le informazioni riportate in Appendice B riguardanti gli accorgimenti da tenere in considerazione durante la fase di progettazione dell'avvio e dell'assegnamento statico delle risorse ai due kernel.

Per l'ispezione di questa modalità di avvio si è seguito un approccio incrementale: inizialmente si è studiato l'avvio di un kernel al quale è assegnata solamente una parte ridotta della memoria effettivamente disponibile; successivamente si è passati all'avvio di un altro al quale è assegnata l'altra parte della memoria, lasciando inutilizzata la prima metà di questa. Si è quindi arrivati ad unire i due risultati ed a creare una lista di comandi da impartire ad U-Boot per configurare l'avvio dei due kernel, partizionando la memoria di sistema in due aree disgiunte.

Non essendo a disposizione l'hypervisor citato nella Sezione 3.3 ci si è limitati a valutare la comunicazione attraverso due interfacce fisiche collegate ad uno switch di rete. Su entrambi i Sistemi Operativi è stato installato il web server lighttpd, poi per ognuno si è proceduto con il download di un piccolo file di esempio verso l'host di compilazione, con lo scopo di valutare la comunicazione con l'interfaccia di amministrazione della rete. Un secondo download è stato fatto da un Sistema Operativo verso l'altro e vice versa,

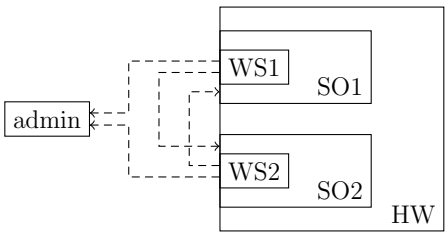


Figura 5.1: Direzioni nelle quali è stato trasferito il file di test

per valutare la comunicazione in caso di query distribuita. Uno schema dei trasferimenti provati è mostrato anche in Figura 5.1.

Completato questo passo ci si è concentrati sulla creazione di un ambiente di test per valutare la comunicazione tra PHP su nodi diversi e tra PHP e JavaScript usando i Web Service e SOAP. Non avendo a disposizione l'hardware target si è inserito tale piattaforma software in una rete simulata, composta di tre macchine virtuali QEMU collegate all'host tramite delle interfacce connesse su un bridge software appositamente creato, come mostrato in Figura 5.2.

Queste macchine hanno eseguito tre istanze di kernel, ognuna equipaggiata con un web server, l'interprete PHP, gli script host, il software di dump ed il database locale, ovvero la configurazione software che dovrebbe avere il power manager installato su un generico nodo di rete. Non avendo a disposizione la doppia cpu PowerPC^{TM} su queste macchine virtuali non è stato provato l'avvio asimmetrico, ma solamente l'esecuzione del kernel incaricato di eseguire il power manager. Questo ha impedito di dimostrare la non intrusività del power manager rispetto al normale funzionamento del nodo di comunicazione, ma ha comunque fornito la possibilità di valutare la capacità di comunicazione tramite SOAP dei nodi e della piattaforma di amministrazione.

Per eseguire questa categoria di test si è deciso di sfruttare il modello

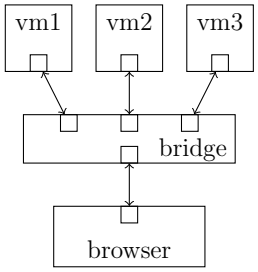


Figura 5.2: Architettura logica della rete di test

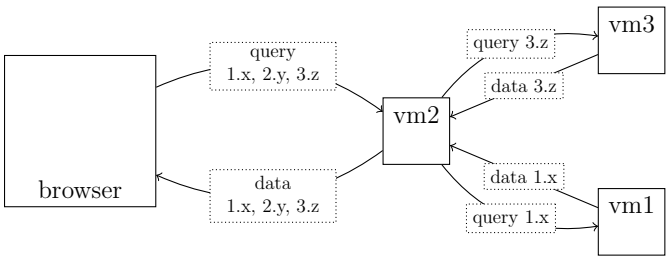


Figura 5.3: Schema di esecuzione di una query distribuita

di consumo di potenza implementato all'interno del codice del software di dump. Come è stato mostrato nell'introduzione è possibile ricavare alcune informazioni sullo stato del nodo anche quando non sono disponibili i relativi sensori: è sufficiente applicare le equazioni del modello ai dati disponibili. Si è quindi deciso di far funzionare il software di dump facendogli sfruttare il modello su dei dati di utilizzo della larghezza di banda disponibile generati in maniera pseudo-casuale. Tale modello è stato esteso rispetto alla sua versione standard in quanto aggiunge uno spostamento della media e della varianza dei valori casuali generati sulla base dell'ora di sistema alla quale è eseguita la misurazione. Questa modifica è stata effettuata per dare più coerenza ai dati casuali generati e una maggiore prevedibilità alle informazioni di consumo di potenza registrate all'interno del database.

Data questa piattaforma sono state compiute due categorie di test, una locale e una distribuita. Nella prima si è utilizzato il browser per scaricare la piattaforma di amministrazione da una particolare macchina virtuale, e tramite questa si è eseguita una query sul database locale al nodo. Tramite wireshark[1] si sono studiate le comunicazioni con tale macchina virtuale e si è riscontrata l'effettiva aderenza dei pacchetti allo standard SOAP e la loro corretta formattazione rispetto al WSDL definito.

Per quanto riguarda il test distribuito sono state compiute le stesse operazioni, ma questa volta dall'interfaccia dell'amministratore si sono anche richieste delle informazioni presenti sulle altre macchine virtuali. Questo tipo di interrogazione ha obbligato gli script PHP presenti sul nodo da cui è stata scaricata l'interfaccia di partizionare la richiesta ricevuta come mostrato in Figura 5.3. Si è potuto verificare che le query inoltrate verso gli altri nodi contenevano solamente informazioni delle quali il sorgente non era a conoscenza, il quale successivamente ha aggregato le risposte ricevute in un unico messaggio ritornato all'interfaccia di amministrazione.

[1]Software di packet sniffing che permette di studiare le comunicazioni di rete mostrando in una lista ordinata tutte le comunicazioni che attraversano una particolare interfaccia, evidenziando i vari protocolli utilizzati — `http://www.wireshark.org`

Capitolo 6

Conclusioni

In questo lavoro di tesi si sono poste le basi per un futuro sviluppo di una architettura di power management distribuito a livello di rete. La situazione di partenza vedeva una generica rete di comunicazione in una situazione completamente non gestita e dal comportamento non analizzabile: un eventuale amministratore avrebbe avuto un complicato compito di valutazione delle performance nodo per nodo e, conoscendo le abitudini degli utenti della rete o stimandole da profili di utilizzo generici, avrebbe dovuto applicare delle politiche di power saving connettendosi manualmente ad ogni nodo per attivare o disattivare i componenti che richiedevano modifiche.

L'obiettivo che ci si è posti prevede che la rete di comunicazione, in particolare i singoli nodi, sia in grado di gestirsi in maniera autonoma, applicando alcune politiche di power saving specificate mediante un linguaggio formale direttamente interpretabile dal software do power management del nodo. Questo software sarà in grado di inviare autonomamente opportuni comandi di attivazione e spegnimento di ogni possibile elemento della rete suscettibile di ottimizzazione sotto il profilo del consumo di potenza. Dopo un'analisi di alcune alternative è stato deciso di implementare un software distribuito che, appoggiandosi su ogni nodo di comunicazione, fosse in grado di analizzare lo stato attuale e storico del nodo ospite e di quelli adiacenti: grazie a tali informazioni il power manager deve essere in grado di compiere scelte adeguate relativamente a quali componenti della rete attivare o disattivare secondo quanto descritto dalla politica che sta attuando. Il power manager distribuito, durante la fase di raccolta dei dati da analizzare, non deve influenzare in alcun modo il funzionamento normale della rete, tranne che per il piccolo quantitativo di traffico necessario alle comunicazioni tra i vari nodi; è stato quindi deciso di disaccoppiarne il funzionamento dal software del nodo, dedicandogli di conseguenza un core di elaborazione e una parte della memoria di sistema disponibile.

Per garantire la comunicazione tra i vari moduli del power manager distribuito sulla rete si è deciso di sfruttare lo standard dei Web Service, con l'obiettivo di ottenere una comunicazione facilmente implementabile permettendo di astrarre da tutte le problematiche legate alla gestione del canale, all'encoding e al parsing dei messaggi scambiati. Questa soluzione rende inoltre molto semplici eventuali estensioni future della comunicazione tramite nuovi metodi e nuovi messaggi.

Per permettere ad un amministratore di studiare ed eventualmente modificare il comportamento del power manager, si è resa necessaria la realizzazione di un'apposita piattaforma in grado di comunicare con questo utilizzando il suo stesso linguaggio, quello dei Web Service. È stato deciso di implementare tale piattaforma in HTML e JavaScript, in modo da non vincolarla ad una particolare piattaforma software, ma facendo invece in modo che fosse aderente agli standard W3C, l'ente che si occupa della standardizzazione di tutto quello che ruota attorno alla rete Internet.

Questo lavoro di tesi ha riguardato principalmente il backend della piattaforma, in particolare lo studio dell'avvio asimmetrico di due Sistemi Operativi, uno dei quali destinato ad eseguire il solo power manager. Per ogni power manager è stato creato un software in grado di leggere periodicamente le informazioni di prestazione dai vari sensori presenti sul nodo, e di calcolare i valori per i quali i sensori non sono disponibili, sfruttando un apposito modello. A tutte queste informazioni è stata data persistenza all'interno di un database locale al nodo e sono state esposte verso la rete, tramite i Web Service, le funzionalità per interrogare tale base di dati.

Per la comunicazione di rete è stato progettato e creato il file WSDL che descrive i metodi esposti, e si è fatto in modo che le classi di interfaccia tra il software e la rete potessero usare tale file per auto-configurarsi. Dal lato della piattaforma di amministrazione ci si è concentrati sulla creazione di uno strumento di debug della rete, non inteso come utilizzabile in ambito produttivo a meno di non apportare le dovute estensioni per completare la sua struttura. Questa interfaccia è infatti in grado solo di eseguire alcune query al database distribuito e di visualizzare i dati ottenuti dalla rete.

Quanto è stato fatto deve essere posto come base per futuri sviluppi ed estensioni della piattaforma. Alcuni di questi possibili obiettivi futuri sono elencati nella sezione seguente.

6.1 Sviluppi futuri

Per il proseguimento dello sviluppo di questa piattaforma sono stati identificati alcuni passi che sono determinanti per il suo corretto funzionamento.

Il primo passo da completare riguarda la possibilità di comunicazione tra nodo di rete e relativo power manager utilizzando un socket virtuale in vece di un'interfaccia fisica. Ciò si rivela essenziale per ridurre al minimo l'impiego di risorse hardware da parte della piattaforma di power management e, di conseguenza, evitare sprechi di potenza innanzitutto da parte della piattaforma stessa. Permette inoltre di massimizzare l'utilizzo di hardware di networking a supporto della rete, senza dedicare delle componenti al solo funzionamento del power manager.

A pari priorità rispetto al passo sopra citato si trova la necessità di rendere la piattaforma in grado di avere un ruolo attivo nella gestione energetica della rete, potendo inviare al software del nodo specifici comandi di attivazione e spegnimento di particolari dispositivi. Questo passo non è dipendente da quello citato in precedenza, in quanto in via temporanea è possibile sfruttare un'interfaccia fisica per la comunicazione tra power manager e nodo; è ad ogni modo un passo molto importante in quanto elemento determinante per l'attuazione di una qualsiasi politica di power management, sia essa gestita da un amministratore umano o da un software appositamente creato.

Completato questo passo diventa importante un primo lavoro di estensione dell'interfaccia amministratore per garantire a quest'ultimo la possibilità di inviare comandi ai vari nodi e non solamente di effettuare delle query. Ciò consente innanzitutto di valutare la correttezza della comunicazione tra power manager e nodo di rete e, in secondo luogo, l'interfaccia può essere utilizzata da un amministratore per modificare manualmente la struttura della rete senza dover impiegare la console di amministrazione dei nodi oggetto della riconfigurazione.

Un ulteriore sviluppo necessario al completamento del power manager consiste nell'implementazione di un policy manager, ovvero di quel software, cuore della piattaforma automatica e in esecuzione su ogni nodo di rete, con il compito di analizzare lo stato del nodo e, tramite apposite comunicazioni con gli altri policy manager, di valutare l'attivazione o lo spegnimento di alcuni suoi componenti. Per una maggiore versatilità di questo software potrebbe essere necessario mantenere in un file di configurazione esterno alcuni suoi parametri di funzionamento, in modo che un amministratore umano possa agire su tali parametri per modificare il comportamento del software. La soluzione ottima sul piano della versatilità potrebbe essere quella in cui va sviluppato preventivamente un linguaggio di descrizione di generiche politiche e, tramite questo linguaggio, sia possibile comunicare al software sopra citato le azioni da compiere in particolari situazioni che potrebbero verificarsi. Un'ipotesi interessante da questo punto di vista potrebbe essere quella di valutare l'applicabilità di regole scritte in Fuzzy Logic, definendo delle variabili letterali per descrivere lo stato del nodo e della rete nel suo intorno e,

tramite queste regole, prendere le decisioni di power saving.

Altri possibili sviluppi della piattaforma riguardano principalmente il client, e spaziano dalla correzione della libreria `soapclient` per garantirle una maggiore aderenza allo standard WSDL, all'aggiornamento delle librerie di interfaccia utilizzate, in modo che possano accettare ed utilizzare lo standard WSDL 2.0 invece della sua versione 1.1, dato che quest'ultima non è mai stata ratificata come standard W3C. Un'altra estensione potrebbe essere infine quella che rilassa il vincolo indicato all'inizio della Sezione 4.6.1, secondo il quale l'amministratore deve essere a conoscenza della struttura della rete: potrebbe essere interessante fare in modo che la piattaforma di amministrazione possa costruire una mappa incrementale della rete, mostrando inizialmente solo il nodo dal quale è stata scaricata l'interfaccia e, tramite richieste successive, possa fornire all'amministratore una mappa più o meno dettagliata della rete a seconda delle necessità.

Appendice A

Modello XML del nodo

In questa appendice, ad esempio della modellazione che è stata eseguita, è mostrato un file XML che descrive la struttura di un generico nodo di rete. Per la validazione di questo tipo di file è stata fatta la scelta dei Document Type Definition (DTD) in quanto non sono necessarie regole stringenti su tipi di dati ma è invece importante la struttura del documento.

A.1 Modello XML

A seguire è riportato il modello XML di un nodo di rete con due matrix (1.5 e 1.6), due line card (1.7 e 1.8) con rispettivamente due e quattro interfacce di rete (1.7.1, 1.7.2, 1.8.1 - 1.8.4). Ogni dispositivo è identificato da un omonimo nodo nell'albero XML, tranne per le parti comuni, per le quali vi è solo un consumo di potenza statico: queste sono rappresentate dai nodi `commonPart` in cima all'albero.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE node SYSTEM "node.dtd">
<node id="1">
   <commonPart id="1" description="Fan"    sPower="30" />
   <commonPart id="2" description="Supply" sPower="50" />
   <commonPart id="3" description="EC"     sPower="20" />
   <commonPart id="4" description="SC"     sPower="18" />

   <matrix id="5"
           description="Matrix 1"
           maxRate="320" sPower="100" dPower="50" />
   <matrix id="6"
           description="Matrix 2"
```

```
            maxRate="320" sPower="100" dPower="50" />

   <lineCard id="7"
             description="2x10Gb/s"
             maxRate="20" sPower="30" dPower="20">
      <interface id="1"
                 description="SFP 1x10Gb"
                 maxRate="10" sPower="20" dPower="10" />
      <interface id="2"
                 description="SFP 1x10Gb"
                 maxRate="10" sPower="20" dPower="10" />
   </lineCard>

   <lineCard id="8"
             description="10x1Gb/s"
             maxRate="10" sPower="30" dPower="20">
      <interface id="1"
                 description="SFP 1x1Gb"
                 maxRate="1" sPower="5" dPower="5" />
      <interface id="2"
                 description="SFP 1x1Gb"
                 maxRate="1" sPower="5" dPower="5" />
      <interface id="3"
                 description="SFP 1x1Gb"
                 maxRate="1" sPower="5" dPower="5" />
      <interface id="4"
                 description="SFP 1x1Gb"
                 maxRate="1" sPower="5" dPower="5" />
   </lineCard>
</node>
```

A.2 File DTD

A seguire il file `node.dtd` che permette di validare l'XML di descrizione del nodo. Nel file XML mostrato alla sezione precedente non sono elencati sensori, ma dal DTD è possibile notare che è consentito aggiungerli all'interno di un apposito nodo che, se presente, deve comparire come primo figlio di ogni nodo dell'albero XML.

```
<!ELEMENT node (commonPart*, matrix+, lineCard+)>
```

```
<!ATTLIST node id CDATA #REQUIRED>

<!ELEMENT commonPart (sensors?)>
<!ATTLIST commonPart id          CDATA #REQUIRED>
<!ATTLIST commonPart description CDATA #IMPLIED>
<!ATTLIST commonPart sPower      CDATA #REQUIRED>

<!ELEMENT matrix (sensors?)>
<!ATTLIST matrix id           CDATA #REQUIRED>
<!ATTLIST matrix description  CDATA #IMPLIED>
<!ATTLIST matrix maxRate      CDATA #REQUIRED>
<!ATTLIST matrix sPower       CDATA #REQUIRED>
<!ATTLIST matrix dPower       CDATA #REQUIRED>

<!ELEMENT lineCard (sensors?, interface+)>
<!ATTLIST lineCard id          CDATA #REQUIRED>
<!ATTLIST lineCard description CDATA #IMPLIED>
<!ATTLIST lineCard maxRate     CDATA #REQUIRED>
<!ATTLIST lineCard sPower      CDATA #REQUIRED>
<!ATTLIST lineCard dPower      CDATA #REQUIRED>

<!ELEMENT interface (sensors?)>
<!ATTLIST interface id           CDATA #REQUIRED>
<!ATTLIST interface description  CDATA #IMPLIED>
<!ATTLIST interface maxRate      CDATA #REQUIRED>
<!ATTLIST interface sPower       CDATA #REQUIRED>
<!ATTLIST interface dPower       CDATA #REQUIRED>

<!ELEMENT sensors (state|rate|power)+>

<!-- power state sensor -->
<!ELEMENT state EMPTY>
<!ATTLIST state address CDATA #REQUIRED>
<!-- current bandwidth usage sensor -->
<!ELEMENT rate  EMPTY>
<!ATTLIST rate  address CDATA #REQUIRED>
<!-- power consumption sensor -->
<!ELEMENT power EMPTY>
<!ATTLIST power address CDATA #REQUIRED>
```

Appendice B

Configurazione di avvio asimmetrico

In questa appendice sono presenti maggiori informazioni riguardanti la configurazione di un avvio asimmetrico. Nella prima sezione si tratta una breve introduzione ai device tree come strumento di descrizione dell'hardware, successivamente si porrà l'attenzione sulla configurazione richiesta per il caricamento di due kernel sullo stesso nodo di rete.

B.1 Device Tree

Come è stato detto i flattened device tree che devono essere forniti al kernel al momento dell'avvio sono dei file binari che descrivono l'hardware a sua disposizione. Tali file, proprio a motivo della loro natura, sono detti *Device Tree Blob* (DTB) e sono generati da un apposito compilatore a partire da un file sorgente testuale definito *Device Tree Source* (DTS). Il motivo di questa compilazione preventiva da DTS a DTB risiede nella maggiore compattezza del binario rispetto al sorgente, e dal fatto che il primo è direttamente utilizzabile dal kernel, eliminando la necessità del parsing di stringhe di testo.

Inoltre per essere più facilmente trattabili i file DTS possono avere una struttura gerarchica che rispecchia la stessa struttura che è possibile trovare in hardware. Questo ad esempio porta con sé un vantaggio nella descrizione degli indirizzi delle risorse: le periferiche collegate ad un determinato bus hanno una proprietà `reg` che permette di definire gli indirizzi relativi a tale bus e non quelli globali di sistema. Nel nodo che descrive il bus vi sarà poi il campo `ranges` che permette al compilatore di convertire gli indirizzi locali in indirizzi di sistema.

Ad esempio si consideri la seguente parte di file DTS, preso da Grant Likely (2008):

```
opb {
   compatible = "simple-bus";
   #address-cells = <1>;
   #size-cells = <1>;
   ranges = <0x0 0xe0000000 0x20000000>;
   serial@0 {
      compatible = "ns16550";
      reg = <0x0 0x10>;
      interrupt-parent = <&UIC0>;
      interrupts = <1 4>;
   };
   serial@10000 {
      compatible = "ns16550";
      reg = <0x10000 0x10>;
      interrupt-parent = <&UIC0>;
      interrupts = <2 4>;
   };
   flash@1ff00000 {
      compatible = "amd,s29gl256n", "cfi-flash";
      reg = <0x1ff00000 0x100000>;
   };
};
```

È possibile notare una sintassi gerarchica simile al formato JSON. Ogni nodo ha una proprietà `compatible` per suggerire al kernel quale driver caricare per gestire tale periferica, e la proprietà `reg` per definire gli indirizzi ai quali può essere acceduta. In particolare il primo valore in quel campo definisce l'indirizzo base, mentre il secondo definisce la dimensione della regione di indirizzi dedicata a tale periferica.

La porta seriale etichettata come `serial@10000`, ad esempio, è raggiungibile all'indirizzo `0x10000` del bus `opb`, ed il campo `ranges` di quest'ultimo indica che l'indirizzo `0xe0000000` del nodo genitore (non mostrato in questo esempio) è riservato a tale bus ed è considerato come indirizzo base (`0x0`) per i nodi figli. Di conseguenza la porta seriale citata in precedenza ha indirizzo `0xe0010000` all'interno del nodo genitore di `opb`.

Per ulteriori dettagli sui device tree si rimanda a Grant Likely (2008) e alla pagina a loro dedicata su `elinux.org`[1].

[1] `http://elinux.org/Device_Trees`

B.2 Configurazione dell'avvio asimmetrico

Per ottenere una corretta configurazione di un avvio asimmetrico sulla stessa macchina vi sono alcune considerazioni di cui tenere conto. Molta attenzione deve essere posta nella creazione dei device tree, in quanto servono ad indicare al kernel quali periferiche gli è consentito utilizzare, serve inoltre una configurazione particolare al kernel che non può sfruttare l'indirizzo `0x0` della memoria di sistema.

B.2.1 Configurazione e compilazione dei device tree

Per primo è necessario considerare il fatto che, salvo alcune particolari eccezioni come la cache L2, una determinata periferica non può essere condivisa dai due kernel: in questa situazione infatti si avrebbe un conflitto in quanto entrambi cercherebbero in fase di avvio di inviare delle stringhe di inizializzazione che modificherebbero la configurazione già compiuta dall'altro. Ricordando che ognuno dei due kernel è progettato per essere l'unico utilizzatore di ognuna delle periferiche è necessario che non vi siano zone grige, in quanto in questo caso il comportamento della singola periferica o dell'intero sistema se questa rappresenta una componente importante dello stesso, rimane indefinito.

Una delle eccezioni a quanto appena scritto riguarda la memoria di sistema: essa infatti all'interno dei device tree è descritta tramite uno o più appositi nodi, etichettati con `memory`, che al loro interno contengono solamente la proprietà `reg` che definisce l'indirizzo base e la dimensione della memoria disponibile. Tramite una corretta configurazione è quindi possibile dedicare parte della memoria ad un Sistema Operativo e parte all'altro. All'intero del file DTS è possibile lasciare pari a zero i campi relativi alla memoria di sistema, in quanto il bootloader è in grado di fornire queste informazioni al kernel tramite delle apposite variabili d'ambiente.

Per consentire la comunicazione tra i due Sistemi Operativi evitando l'impiego di interfacce di rete esterne è necessario dedicare una parte della memoria di sistema in modo che sia utilizzabile esclusivamente per questa comunicazione. Quest'area di memoria non deve essere accessibile ai due kernel, in quanto è importante che non vi si eseguano operazioni di paginazione o swap su disco. In questo caso infatti si avrebbe che in quest'area potrebbero essere allocati processi da parte di entrambi i Sistemi Operativi, con ovvie conseguenze di inconsistenza della memoria stessa.

Una volta creati i due file DTS da convertire in binari è importante ricordare che il compilatore dei device tree ha un parametro per specificare qual è la CPU sulla quale il kernel deve essere avviato. Questa informazione

deve essere specificata tramite riga di comando, in quanto all'interno del file DTS è presente solamente una lista delle CPU disponibili, ma non è possibile definire quale tra queste è quella alla quale è demandato di eseguire le primissime istruzioni di avvio del kernel dopo il suo caricamento in memoria.

B.2.2 Configurazione e compilazione dei kernel

È inoltre importante ricordare che di default il kernel è compilato con l'assunzione di essere allocato all'inizio della memoria di sistema, a partire dall'indirizzo `0x0`. Questo è possibile per il kernel del Sistema Operativo al quale è assegnata la porzione bassa della memoria, quella cioè che inizia effettivamente all'indirizzo `0x0`, mentre per l'altro è necessario modificare tale costante in modo che rispecchi l'effettiva ubicazione della memoria che gli è dedicata.

Questa operazione è necessaria in quanto il device tree non può contenere un campo di traduzione di indirizzi all'interno del nodo `memory` ma ha solamente un campo `reg` che definisce indirizzo base e una dimensione della memoria disponibile. Questo tipo di modifica dei parametri di configurazione del kernel è necessaria per ogni hardware che non ha la memoria di sistema raggiungibile all'indirizzo `0x0`.

La posizione che deve occupare in memoria il kernel pone un vincolo sul partizionamento della RAM: questo infatti è progettato con l'assunzione di essere allocato nella parte bassa della memoria disponibile. Anche se questa può essere impostata tramite il bootloader è necessario definire il partizionamento il prima possibile, in quanto l'indirizzo di inizio della memoria riservata al secondo Sistema Operativo coincide con quello di avvio del kernel, e questo rappresenta uno dei parametri di compilazione dello stesso.

Appendice C

Metodi implementati

Su ogni nodo di comunicazione sono stati implementati alcuni metodi per verificare il corretto funzionamento della piattaforma, l'interfaccia con tali metodi è descritti in dettaglio all'interno del file WSDL.

In questa appendice si intende fornire una loro breve descrizione.

`getXMLHistory()`
Permette di richiedere la lista delle letture eseguite in un determinato periodo dai dispositivi indicati. Gli argomenti sono elencati di seguito e mostrati in maniera grafica in Figura C.1a.

1. `devices`: Array di stringhe, contenenti l'elenco degli identificativi di rete dei dispositivi richiesti
2. `fields`: Array di stringhe, con l'elenco delle informazioni richieste per ogni dispositivo
3. `startTS`: Stringa rappresentante il timestamp dell'inizio dell'intervallo di tempo oggetto della query
4. `endTS`: Stringa rappresentante il timestamp del termine dell'intervallo di tempo oggetto della query

`getJSONHistory()`
Il suo comportamento è identico a quello del metodo precedente, la differenza riguarda la formattazione del messaggio di risposta: mentre per `getXMLHistory` questo è in formato XML, `getJSONHistory` ritorna una stringa JSON. La struttura logica e le informazioni contenute nei messaggi è identica, il vantaggio di JSON rispetto ad XML sta nel fatto che il primo utilizza una notazione decisamente meno verbosa e di conseguenza più compatta.

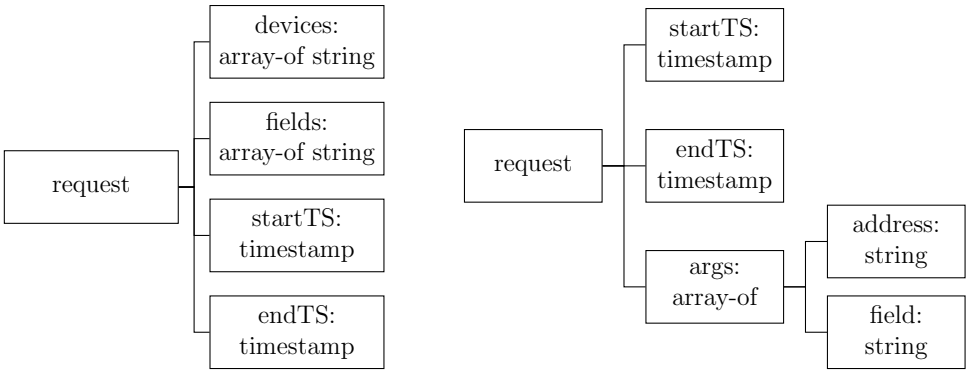
`getXMLHistoryV2()`
Il suo comportamento è simile a quello dell'omonimo `getXMLHistory`, rappresenta infatti semplicemente una variante di tale metodo. Mentre con il precedente era necessario richiedere gli stessi campi per ogni dispositivo, con questo metodo è possibile diversificare tale richiesta, in quanto il campo `fields` è affiancato ad ogni istanza di `address`. L'elenco dei parametri da fornire a questo metodo è mostrato nell'albero di Figura C.1b

`getJSONHystoryV2()`
È la controparte `V2` di `getJSONHistory`, anche questo con lo stesso comportamento ma con la modifica ai parametri già descritta al punto precedente per `getXMLHistoryV2`.

Per tutti i metodi elencati le due stringhe che rappresentano i timestamp limite richiesti `startTS` e `endTS` sono opzionali, e se non specificati non è imposto un filtro ai record recuperati dai database. Questo significa che ad esempio se nessuno dei due valori è specificato vengono ritornati tutti i record che rispettano gli altri vincoli.

Il formato delle stringhe che rappresentano dei timestamp, siano esse tra i parametri di ingresso dei metodi o tra i valori restituiti, devono essere aderenti allo standard ISO 8601:2004, adottato anche dal W3C. Questo impone il formato `YYYY-MM-DDTHH:MM:SSZ`, nel quale i primi dieci caratteri rappresentano una data, la lettera `T` è utilizzato come carattere separatore ed il resto della stringa rappresenta un orario. Tale orario è terminato dalla lettera `Z` che specifica tale timestamp come relativo al Tempo Universale Coordinato (UTC).

La struttura dei messaggi ritornati dai metodi è quella mostrata in Figura C.2, in base al metodo invocato essa può essere espressa in XML o in JSON, ma per entrambe il contenitore rimane lo stesso, ovvero un messaggio SOAP espresso in XML. Nel caso di JSON, quindi, non è possibile eseguire una validazione usando XML Schema, in quanto tale messaggio è identificato come un contenitore SOAP con al suo interno una generica stringa di caratteri.



(a) Metodi `V1` (b) Metodi `V2`

Figura C.1: Alberi delle strutture dati da fornire ai metodi implementati

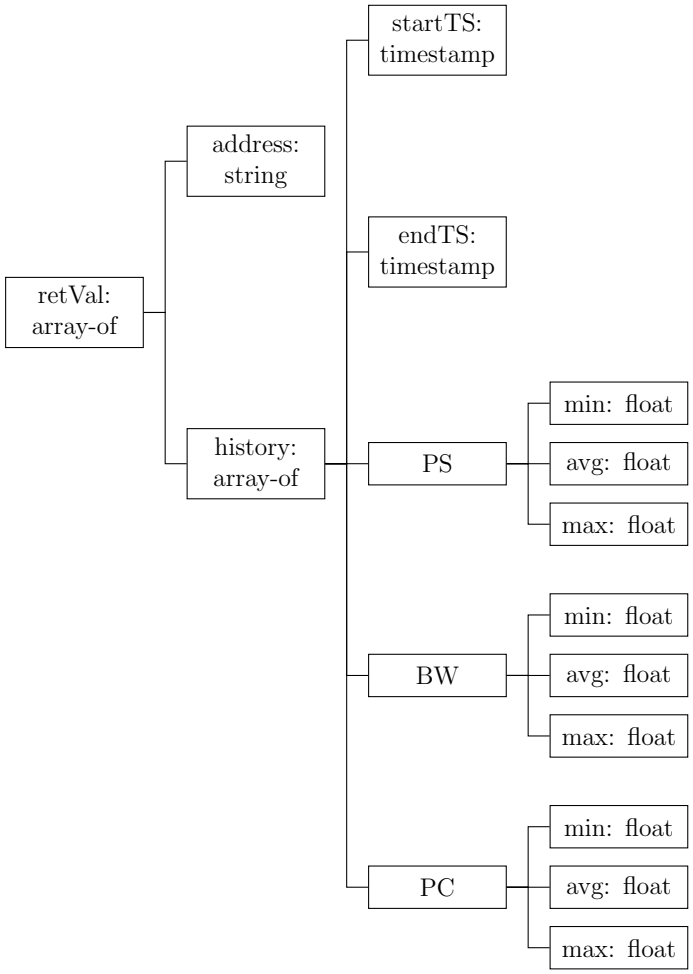


Figura C.2: Albero della struttura dati restituita dai metodi implementati

Bibliografia

Dronamraju Subramanyam, John Rekesh S. A. (2012). Multicore networking in linux user space with no performance overhead. Embedded Online, `http://www.embedded.com`.

Grant Likely J. B. (2008). A symphony of flavours: Using the device tree to describe embedded hardware. Proceeding of Linux Symposium, `http://ols.fedoraproject.org/OLS/Reprints-2008/likely2-reprint.pdf`.

Merritt R. (2008). Cpu designers debate multi-core future. EE Times Online, `http://www.eetimes.com`.

Pautasso C.; Zimmermann O.; Leymann F. (2008). Restful web services vs. big' web services: making the right architectural decision. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pp. 805–814, New York, NY, USA. ACM.