ALMA MATER STUDIORUM • UNIVERSITÀ DI BOLOGNA

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea in Informatica

Supporto alla mobilità e alla autoconfigurazione per il framework Virtual Distributed Ethernet

Tesi di Laurea in Sistemi Operativi

Tesi di Laurea di:
Luca Bigliardi

Relatore:
Renzo Davoli

III Sessione

Anno Accademico 2005-2006

ALMA MATER STUDIORUM • UNIVERSITÀ DI BOLOGNA

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea in Informatica

Supporto alla mobilità e alla autoconfigurazione per il framework Virtual Distributed Ethernet

Tesi di Laurea in Sistemi Operativi

Parole Chiave:
mobilità, virtualizzazione, reti, virtual networking, sistemi operativi

Tesi di Laurea di:
Luca Bigliardi

Relatore:
Renzo Davoli

III Sessione

Anno Accademico 2005-2006

A chi mi deve sopportare.

Indice

1 INTRODUZIONE **3**

2 IDEA E SCENARIO **6**

2.1 Motivazioni E Vantaggi 6

2.2 VDE 7

2.2.1 Swiss Army Knife 8

2.2.2 Struttura 9

2.3 Virtual Square 12

2.3.1 Il Progetto 13

2.3.2 La Situazione Odierna 13

2.3.3 Il Futuro 14

2.4 vde_switch 15

2.4.1 Il Sistema Di Console Management 17

2.4.2 Connessione Di Un Cavo 20

2.4.3 Gestione Dei Pacchetti 20

2.5 Spanning Tree 21

2.5.1 Descrizione 21

2.5.2 Protocollo 23

Elezione Della Radice 24

Gli Stati Di Una Porta 25
2.5.3 Rapid Spanning Tree Protocol 26
Stati E Ruoli 26
Formato E Gestione Dei BPDU 27
Porte Edge 28
Topology Change 28
2.5.4 Implementazione In vde_switch 29

3 IMPLEMENTAZIONE 31
3.1 Struttura Modulare 31
3.2 vde_autolink 33
3.2.1 Logica di utilizzo 33
3.2.2 Struttura interna 37
3.3 unixtermcmd 39
3.4 Il Sistema 40
3.4.1 guessnet 41
3.4.2 ifplugd - udev 41
3.4.3 Composizione dei software 42

4 SVILUPPI FUTURI 43
4.1 Asincronia 43
4.2 libvdemgmt 44
4.3 Cavi VDE 44
4.4 Action 44
4.5 Esempi 45
4.6 Bi-mobile 45

5 CONCLUSIONI 46

Capitolo 1

INTRODUZIONE

La mobilità e la virtualizzazione sono tematiche molto sentite nello scenario informatico odierno. Per dimostrarlo basta considerare la costante crescita del numero di progetti ideati per risolvere problemi riconducibili a questi due aspetti.

Negli ultimi anni gli enormi passi in avanti fatti dai laboratori di ricerca delle ditte costruttrici di hardware hanno messo a disposizione del grande pubblico sistemi per calcolare ed immagazzinare i dati sempre più compatti e performanti, ma soprattutto modelli di reti senza fili che permettono di estendere la copertura di una connessione per lo scambio di dati a praticamente tutto il pianeta. Se da un lato esistono standard e protocolli ideati appositamente per risolvere in modo efficiente il problema della mobilità di un dispositivo all'interno di varie reti[1], dall'altro lato è prassi comune creare reti non trasparenti e con limitazioni nelle funzionalità, in perfetta controtendenza con la ricerca di soluzioni mobili. In una situazione simile l'unica possibilità per ottenere trasparenza sembra utilizzare reti virtuali che si appoggino in modo dinamico alle reti reali.

Ad oggi la virtualizzazione è ritenuto il tema più scottante nel campo dei Sistemi Operativi. L'idea in sè non è nuova[2], ma recentemente

[1] Si pensi ad esempio a MIPv6.
[2] Si consideri il sistema IBM VM/370.

è stata ripresa ed utilizzata su larga scala quando ci si è accorti che la potenza di un singolo calcolatore comune è sufficiente a simulare all'interno di un sistema (parti di) altri sistemi altrimenti inutilizzabili[3]. Come testimonianza dell'attualità della virtualizzazione basta considerare la recente aggiunta di istruzioni macchina specifiche per virtualizzare nei processori Intel e AMD e l'integrazione nei maggiori sistemi operativi di sistemi per la virtualizzazione.

Nel *capitolo 2 - IDEA E SCENARIO* viene proposto un problema utilizzato come riferimento durante lo sviluppo della tesi. Vengono quindi presentati il progetto Virtual Square, che si propone di rivoluzionare il modo di pensare un sistema operativo grazie alla virtualizzazione, ed il framework Virtual Distributed Ethernet, una suite di applicazioni dedicate alla virtualizzazione di rete altamente flessibili sviluppate all'interno di Virtual Square. Le restanti sezioni sono dedicate all'analisi strutturale di vde_switch, la componente principale del framework VDE, al fine di avere un'idea chiara sulla logica di funzionamento e sulle peculiarità dei sottosistemi che lo compongono.

Nel *capitolo 3 - IMPLEMENTAZIONE* viene illustrato lo schema di ragionamento utilizzato per proporre una soluzione al problema presentato nel capitolo 2. In seguito vengono presentate due applicazioni aggiuntive proposte per il framework VDE che combinandosi con altri software preesitenti permettono di raggiungere l'obbiettivo stabilito.

Nel *capitolo 4 - SVILUPPI FUTURI* sono stati elencate delle idee nate durante lo svolgimento del progetto che si propongono di migliorare alcuni aspetti della suite di applicazioni Virtual Distributed Ethernet, di rendere più flessibile il comportamento delle applicazioni sviluppate ed infine di espandere le possibilità stesse di impiego di un sistema mobile.

Nel *capitolo 5 - CONCLUSIONI* si conclude il lavoro, riassumendo

[3]Come esempi si possono considerare le macchine virtuali complete che permettono di far girare più sistemi operativi contemporaneamente (Qemu, Xen, OpenVZ), gli emulatori di architettura (PearPC) e gli emulatori di chiamate a sistema (Wine).

le fasi del progetto e le linee guida seguite che hanno permesso di risolvere con successo il problema iniziale.

Capitolo 2

IDEA E SCENARIO

2.1 Motivazioni E Vantaggi

Il sistema oggetto di questa tesi viene proposto per risolvere una problematica precisa e ben definita: *riuscire a mantenere, senza bisogno di intervento da parte di un operatore, un collegamento virtuale tra due nodi di rete anche a fronte di cambiamenti nell'ambiente in cui si trova uno dei due*.

Oggi i calcolatori portatili sono estremamente diffusi, è quindi facile pensare un possibile scenario che necessiti di un supporto simile.

Immaginiamo ad esempio Foo, uno studente di informatica che sta sviluppando un software sul proprio laptop che deve effettuare degli scambi di dati tramite protocollo IPv6 con altri servizi attivi sui server della sua facoltà. Foo sta facendo debugging sul suo progetto a casa, ma deve recarsi presso il proprio dipartimento per una lezione; mette quindi in sospensione il sistema operativo del notebook, si reca presso il suo dipartimento e, dopo aver constatato che la lezione è stata sospesa per mancanza del docente, ripristina il lavoro sospeso semplicemente riaprendo il portatile senza riconfigurare nulla.

In alternativa possiamo considerare il signor Bar che ha aperto una sessione sul software gestionale della propria ditta dal suo ufficio nel-

la sede centrale e che si deve spostare in un distaccamento per una riunione. Egli può collegare il proprio cellulare al computer portatile ottenendo così una migrazione automatica della connessione dalla rete LAN dell'azienda alla rete GPRS che gli permette di continuare a lavorare sul software gestionale senza interruzioni mentre l'autista lo porta a destinazione. Una volta arrivato il signor Bar può staccare il laptop dalla connessione GPRS e, sempre senza interruzioni, concludere le operazioni che stava eseguendo attraverso la connessione della seconda sede.

Data la natura della problematica in discussione è immediato ricavare i vantaggi che porta una sua soluzione: una volta configurato il sistema secondo le proprie esigenze l'adattamento all'ambiente avviene in modo automatico e quindi *l'utente evita di dover eseguire operazioni ripetitive e meccaniche*, evitando perdite di tempo. L'automatismo ottenuto abbassa inoltre la soglia di difficoltà nell'utilizzo dei sistemi per il networking virtuale allargando il bacino di utenza: nel secondo esempio il signor Bar, che non conosce minimamente i dettagli delle impostazioni di rete, può aver affidato la configurazione al tecnico della propria ditta.

2.2 VDE

L'idea qui discussa si basa su Virtual Distributed Ethernet[1], un progetto di ampio respiro studiato per permettere qualsiasi tipo di virtualizzazione nelle topologie di rete. Come suggerisce l'acronimo stesso vde è *virtuale* poichè interamente software, è *distribuito* poichè parti della stessa rete possono risiedere su calcolatori fisicamente separati ed infine è una *ethernet* poichè emula uno stack di rete a livello ethernet e si compone di switch, cavi e plug.

[1] http://vde.sourceforge.net/

2.2.1 Swiss Army Knife

L'autore, Renzo Davoli, definisce VDE come *"a swiss army knife for emulated networks"* in riferimento alla sua struttura. I motivi di questa affermazione sono da ricercare nelle peculiarità del software:

- VDE offre piena compatibilità ethernet.
- VDE è distribuito, ossia non ha una gestione centralizzata, nè esiste un componente che abbia una visione globale della rete emulata. Questo garantisce scalabilità.
- VDE è generale, un'architettura di rete virtuale collegabile a OS reali, emulatori, macchine virtuali, altri strumenti di connettività. Il lavorare a livello 2 permette inoltre di costruire un qualsiasi stack di rete al di sopra di esso.
- VDE è scarsamente invasivo, in quanto può essere eseguito (tranne che per alcune applicazioni) senza alcun privilegio di superutente. Questo è un vantaggio dal punto di vista della sicurezza.

A dimostrazione della versatilità di VDE vediamo come esso possa essere utilizzato per ricoprire varie sfumature del "virtual networking". Per ogni aspetto viene citato il nome di un software specifico le cui funzionalità sono quindi da considerarsi un sottoinsieme di quello che offre VDE:

VPN (esempio: *OpenVPN*[2]) Una "virtual private network" è una rete di comunicazione privata spesso utilizzata per comunicare in modo sicuro attraverso una rete pubblica (ad esempio Internet). Ciò che caratterizza una VPN è l'essere punto-punto e il collegare due host fisici.

[2] http://openvpn.net/

Overlay-Network (esempio: *Akamai*[3], *peer-to-peer software*) Una "overlay network" è una rete di computer costruita al di sopra di un'altra rete. I nodi di una overlay network possono essere connessi da link virtuali o logici. Ogni collegamento si può estendere attraverso più link della rete sottostante. Un esempio di overlay network è la stessa Internet, se usata in modo Dial-Up tramite una rete telefonica.

VM-network (esempio: *uml-switch*, *VMWare's vmnet*[4]) Una "virtual machine network" è una rete simulata usata per far comunicare le macchine virtuali presenti su un unico host fisico.

2.2.2 Struttura

VDE è una collezione di applicazioni sviluppate per sistemi POSIX, interamente rilasciate sotto licenza GPL. Verranno ora mostrate le singole applicazioni, iniziando da quelle che implementano i componenti Ethernet, gli switch e i cavi.

vde_switch È l'applicazione fondamentale di VDE, alla quale le altre fanno rifermento. Basato su uml_switch originariamente sviluppato da Jeff Dike per User Mode Linux[5], si tratta di una implementazione fedele di uno switch Ethernet, ossia un punto di collegamento per più cavi che instrada i pacchetti ricevuti al cavo il cui MAC address corrisponde a quello di destinazione del pacchetto. È anche possibile fare funzionare vde_switch come un repeater (hub). Lo switch può essere controllato avviandolo come processo in foreground e premendo Invio per accedere al prompt dei comandi, oppure (se lanciato come demone) con *unixterm*, altra applicazione di VDE utilizzata per dialogare con i socket di

[3] http://www.akamai.com/
[4] http://www.vmware.com/
[5] http://user-mode-linux.sourceforge.net/

controllo delle varie componenti. (Vedi la sezione 2.4 per una descrizione dettagliata del funzionamento di vde_switch.)

vde_plug2tap È un cavo particolare che si collega da una parte ad un VDE switch e dall'altra ad un'interfaccia TUN/TAP. Tutti i dati che arrivano al plug dallo switch vengono inviati all'interfaccia e viceversa. Con vde_plug2tap è possibile creare dei collegamenti tra le reti virtuali di VDE e le reti reali utilizzando il sistema operativo reale.

vde_plug* e *dpipe Sono le applicazioni che permettono di creare cavi virtuali tra switch VDE. *vde_plug* si collega ad un vde_switch e invia i pacchetti che riceve dallo switch sullo standard output e i pacchetti che riceve dallo standard input allo switch stesso. *dpipe* implementa una double pipe, ossia una pipe UNIX a due vie; se nella pipe il primo processo dirige il suo standard output allo standard input del secondo, nella double pipe anche il secondo processo manda il suo standard output allo standard input del primo. La sintassi di base di dpipe è *dpipe comando1 = comando2* (comando1 e comando2 comprendono eventuali opzioni) ma può essere estesa per collegare anche più comandi. vde_plug e dpipe, da soli, permettono di collegare due switch sulla stessa macchina; per collegare due switch remoti serve un ulteriore sistema di connessione remota bidirezionale che permetta, se possibile, di lanciare comandi sul sistema remoto. Due applicazioni che già permettono questo sono netcat e SSH. La prima non cripta i dati ed è quindi insicura, mentre la seconda, che utilizza una connessione TCP, può portare ad un degrado delle prestazioni.

vde_cryptcab È un sistema distribuito per l'amministrazione di cavi VDE. Con vde_cryptcab è possibile stabilire una comunicazione attraverso un canale criptato blowfish tra due vde_switch che risiedono su due macchine fisicamente separate. Se usato in modalità client (opzione -c) genera una chiave blowfish random e la

trasferisce sul server remoto utilizzando “scp”. Dopo una fase di verifica delle credenziali a 4 passi il server ed il client iniziano ad inviare e ricevere datagrammi vde incapsulati in cryptogrammi inviati all'altro host con UDP. Se usato in modalità server, ad esempio con il comando vde_cryptcab -s /tmp/vde2.ctl -p 2100 , vde_cryptcab inizializza un server multi-peer in ascolto sulla porta UDP 2100 che collega ogni nuovo client correttamente autentificato ad una diversa porta dello switch vde. A livello pratico per ogni nuova connessione viene lanciata un'istanza di vde_plug. Se i due switch vde sono collegati alle TUN/TAP allora tra le due macchine viene creato a tutti gli effetti un collegamento di livello 2 criptato.

wirefilter È uno strumento di corredo a VDE, utile per emulare reti reali e le problematiche che presentano. Si usa all'interno di un VDE cable, con la sintassi *dpipe cmd_vde_plug1 = wirefilter [opzioni] = cmd_vde_plug2*. Le sue opzioni, impostabili anche a runtime tramite *unixterm*, permettono di aggiungere anche asimmetricamente nei due sensi di comunicazione:

- Perdita di pacchetti (percentuale);
- Ritardi;
- Duplicazione di pacchetti (percentuale);
- Limiti di banda del canale e di velocità di spedizione;
- Limiti alla coda dei pacchetti e all'MTU;
- Rumore (numero di bit danneggiati per megabyte)
- Alterazione della politica FIFO del canale.

È dunque possibile simulare praticamente qualsiasi caratteristica di una connessione di rete, persino senza alcuna connessione reale; una possibilità molto interessante sia per studiare il comportamento di un'applicazione in fase di sviluppo, sia per fornire servizi con limitazioni predefinite.

slirpvde È un'alternativa a TUN/TAP per collegare uno switch alla rete reale. Il codice è preso in gran parte dal supporto di rete di QEMU, a sua volta estensione di Slirp. In particolare, slirpvde si collega ad un vde_switch, riceve i pacchetti Ethernet, li disassembla a livello 4, ne incapsula i payload in nuovi pacchetti tenendo traccia degli IP e delle porte di origine e, aprendo porte locali di conseguenza, li instrada come se fossero pacchetti propri. Alla ricezione di un pacchetto dalla rete reale, slirpvde ricostruisce, in base alle porte, la destinazione nella rete emulata e genera il pacchetto relativo. slirpvde può inoltre far apparire allo switch ad esso connesso un server DHCP, che assegna indirizzi in base ad una subnet definibile dall'utente ed imposta il default gateway al nodo 2 e un server DNS al nodo 3, il quale non fa altro che instradare le query al DNS predefinito della macchina reale. Il vantaggio di slirpvde rispetto a TUN/TAP è la possibilità di instradare il traffico VDE verso la rete reale senza privilegi amministrativi; lo svantaggio è che, proprio per il fatto di essere user mode, slirpvde non è stato progettato per generare pacchetti che richiedono appunto i privilegi di superutente[6]. Se anche questo supporto ci fosse, per poterlo usare, slirpvde perderebbe ogni vantaggio rispetto a TUN/TAP, dovendo anch'esso essere eseguito da root.

2.3 Virtual Square

La suite Virtual Distributed Ethernet è parte di *Virtual Square*[7], un progetto di Renzo Davoli che mira ad ottenere un sistema di virtualizzazione modulare, in cui i vari moduli siano utilizzabili anche singolarmente e che ricopra tutti gli ambiti ad oggi riconosciuti come virtuali, cercando di generalizzare il concetto stesso di sistema virtuale.

[6] Ad esempio alcuni pacchetti ICMP.
[7] http://www.virtualsquare.org/

2.3.1 Il Progetto

Virtual Square è un gioco di parole dal doppio significato, intraducibile in italiano in modo diretto. “Square” può essere tradotto con il termine matematico “quadrato” per indicare il virtuale elevato alla seconda potenza, ma può essere anche tradotto con “piazza” intesa come luogo di incontro.

Con il termine *mondo virtuale* indichiamo comunemente l’ambiente creato da computer e da reti che permette a persone e programmi di comunicare ed interagire. Un mondo virtuale elevato al quadrato è un mondo creato da computer e reti virtuali che appaiono agli utenti come se fossero computer e reti classici, standard.

Oggi è chiaro che un mondo virtuale è anche un luogo di incontro ove comunicare e scambiare idee, opinioni ed informazioni. Virtual Square può quindi anche essere considerato un sistema per definire questi luoghi all’interno di reti virtuali-virtuali.

2.3.2 La Situazione Odierna

Dopo più di due anni di ricerca sono state prodotte e rese disponibili pubblicamente sotto licenza free oltre 100.000 righe di codice C. Svariati sistemi e prototipi sono stati creati, tra questi VDE e UMVIEW sono i componenti maggiormente degni di nota.

UMVIEW è il risultato di una riconsiderazione dell’idea di virtual machine. Si è passati dal pensare una macchina virtuale come sistema monolitico all’idea di una VM creata a partire da una struttura portante, uno scheletro e da vari moduli che implementano specifiche virtualizzazioni. UMVIEW è la struttura portante e ci sono moduli per virtualizzare file system, reti, device, ecc ecc. Reti e file system reali possono coesistere e cooperare con reti e file system virtuali. Ad esempio un processo può accedere ad un unico file system che in realtà è l’unione di una parte di fs reale ed una parte di fs virtuale. UMVIEW

è basato sull'idea di *partial virtual machine* ed è un'implementazione di ViewOS[8]. In ViewOS ogni processo può avere la sua prospettiva, un punto di vista singolare sull'ambiente di esecuzione. In altre parole il file system (cosa significa un path), la rete (che indirizzi IP, dispositivi, rotte ed interfacce sono disponibili), i device, le comunicazioni interprocesso... possono essere definite in base ai processi.

2.3.3 Il Futuro

Virtual Square aggiunge un nuovo grado di libertà nel campo dei sistemi operativi e lascia spazio ad infinite nuove applicazioni, ma conduce allo stesso tempo ad una classe interessante di problemi da analizzare e risolvere.

UMVIEW è un'implementazione user-mode di ViewOS. È possibile creare un kernel che integri sia un supporto alle viste di un processo che un efficente sistema di macchine virtuali (sia parziali che monolitiche)?

Questo approccio può essere un compromesso accettabile tra micro e macro kernel? I due paradigmi possono coesistere all'interno di uno stesso kernel che può quindi essere configurato a seconda che si necessiti maggiormente di performance, sicurezza o disponibilità di driver e servizi.

L'approccio di Virtual Square unifica diversi servizi virtuali come macchine virtuali monolitiche ed altre funzionalità che ad oggi necessitano di tool e soluzioni specifiche (ed a volte complesse). Questo è il caso della system call "chroot", del sistema per montare in loopback le immagini di un filesystem, del tool "fakeroot" e di molti altri. Quali altri servizi possono essere forniti dal framework di Virtual Square? Quali tra questi possono basarsi su servizi esistenti e quali necessitano di espansioni del framework?

Qual è il ruolo dei sistemi operativi multiutente in Virtual Square? Un

[8] http://www.nongnu.org/view-os/web/

utente reale può aver bisogno di definire differenti “ruoli” o “personalità” per disporre di diversi permessi di accesso alle risorse offerte dal sistema in base al livello di sicurezza necessario al momento. Viceversa, è possibile definire delle utenze “community” che si estendono sia ai cluster virtuali che ai servizi condivisi?

Queste sono le sfide che si propone il progetto Virtual Square per i prossimi anni.

2.4 vde_switch

In questa sezione viene descritto in dettaglio vde_switch presente in VDE versione 2.1.3, soffermandosi in particolar modo su alcune caratteristiche chiave che permettono il corretto funzionamento dell'applicazione oggetto di questo studio. Si consiglia di leggere quanto segue con un occhio sul testo e l'altro sul codice.

vde_switch2 presenta una struttura modulare. Vi sono tre moduli principali:

consmgmt che gestisce i socket di management*;*

datasock che gestisce i socket dati*;*

tuntap che gestisce l'eventuale interfaccia “tap” del sistema operativo.

Lo switch utilizza questi moduli esclusivamente tramite le funzioni esportate al loro avvio nella procedura “start_modules()” (vde_switch.c riga 539). Le funzioni che ogni modulo deve esportare sono definite nella “struct swmodule” (switch.h riga 31).

In aggiunta ai moduli possiamo individuare altri sottosistemi:

qtimer che gestisce lo scheduling di alcune operazioni[9] tramite un handler del segnale SIGALRM;

[9]Per capire quali operazioni vengono “schedulate” cercare la chiamata a “qtimer_add()”.

port che gestisce le porte, le virtual-lan e i dati in ingresso passatigli da datasock;

fstp che gestisce le strutture dati relative al fast spanning tree dello switch.

Le strutture dati di maggiore importanza sono:

portv (port.c riga 93) è un vettore statico contenente i puntatori a tutte le porte. Una porta viene rappresentata da una *struct port* (port.c riga 63). Ogni porta ha al proprio interno una lista di *struct endpoint* (port.c riga 53). Ciascun endpoint rappresenta un cavo, questo significa che una porta può avere più cavi collegati ad essa in modo concorrente;

vlant (port.c riga 88) vde_switch, come molti switch reali, supporta varie virtual-lan. Questa struttura viene utilizzata per registrare l'appartenenza di una porta ad una vlan e per controllare che non ci siano scambi di dati tra due vlan;

fds/fdpp (vde_switch.c riga 99) sono due vettori statici accoppiati. Il primo contiene tutti i file descriptor passati alla "poll()" (vedi descrizione del loop principale piu' sotto), il secondo nella posizione i-esima contiene informazioni aggiuntive riguardanti il file descriptor presente nella posizione i-esima del primo. I dati in queste due strutture vengono inseriti utilizzando la funzione "add_fd()" (vde_switch.c riga 145) e possono essere o socket "listening" per consmgmt e datasock o console di management aperte o socket di controllo/dati delle porte o "tap".

hash-table (hash.c riga 41) tabella che tiene traccia dei *MAC address* corrispondenti ad ogni porta, gli indirizzi vengono aggiornati controllando la sorgente dei pacchetti in ingresso e vengono usati per capire a che porta inoltrare il traffico;

packetq (packetq.c riga 41) coda dei pacchetti in uscita;

clh (consmgmt.c riga 51) è un puntatore alla lista dei comandi che vde_switch accetta tramite le console di management. Ogni elemento della lista è una *struct comlist* (consmgmt.h riga 9) che contiene la stringa corrispondente al comando, un elenco formale degli eventuali parametri, la descrizione del comando, il puntatore alla funzione invocata dal comando ed un elenco dei tipi di parametri formali che la funzione si aspetta;

fsttab (fstp.c riga 74) è un vettore che contiene un puntatore ad una *struct vlst* (fstp.c riga 46) per ogni vlan configurata nello switch. Ogni vlst contiene informazioni relative al protocollo di fast spanning tree per una virtual-lan.

Una volta avviato ed inizializzato lo switch esegue un loop infinito "main_loop()" (vde_switch.c riga 208). A questo punto l'esecuzione di una funzione del sistema può avvenire o come conseguenza di una catena di chiamate partite dal ciclo principale o come conseguenza di un'operazione "schedulata".

Il loop principale puo' essere cosi' schematizzato:

- Effettua una "poll()" su tutti i file descriptor registrati dai tre moduli;
- Se "poll()" torna dei file descriptor allora rintraccia il modulo che li ha registrati ed invoca l'handler appropriato;
- Se la coda di pacchetti in uscita non è vuota allora esegui una "packetq_try()" che tenta l'invio dei pacchetti.

2.4.1 Il Sistema Di Console Management

Dalla versione 2.0 vde_switch offre la possibilità di essere controllato e riconfigurato a runtime con comandi inviati ad appositi socket di management. "consmgmt" è il modulo preposto all'amministrazione di tali socket.

All'avvio dello switch i moduli ed i sottosistemi esportano i comandi che vogliono mettere a disposizione dell'utente semplicemente aggiungendoli alla lista "clh" tramite la macro "ADD_CL()" (consmgmt.h riga 30) che richiama la funzione "addcl()" (consmgmt.c riga 53).

Il modulo "consmgmt", oltre ad esportare i propri comandi, esegue le seguenti operazioni (consmgmt.c riga 330):

- Se non viene richiesto allo switch di comportarsi come demone (opzione -d) aggiunge il file descriptor della console da cui è stato lanciato il comando vde_switch alla lista dei file descriptor controllati da "poll()";
- Se viene richiesto allo switch di avere un socket per il management (opzione -M) apre il socket e aggiunge il relativo file descriptor alla lista degli fd controllati da "poll()".

In origine il sistema di console management venne pensato per interagire con un utente in modo sincrono, ma durante lo sviluppo di questo progetto e di altre applicazioni a corredo della suite VDE è emersa la necessità di far dialogare direttamente questi software con lo switch tramite la console. Questo ha portato alla progettazione di un sistema asincrono per la gestione dei messaggi e di una libreria lato client che semplifichi lo scambio di informazioni attraverso il socket management. Tali estensioni verranno discusse in modo approfondito nel capitolo dedicato agli *Sviluppi Futuri*.

Analizziamo cosa succede quando si connette l'applicazione "unixterm" al socket di management di vde_switch, si invia un comando ed infine si chiude il terminale[10]:

- Nel "main_loop()" la "poll()" torna il file descriptor relativo al socket in ascolto per nuove connessioni di console management;

[10]Un comando ricevuto direttamente dal terminale in cui è stato lanciato vde_switch può essere visto come sottocaso di questo: non sono presenti la fase di apertura e di chiusura del socket.

- Viene chiamata la “handle_input()” (consmgmt.c riga 178) che apre un file descriptor per la nuova connessione tramite la “accept()”, lo aggiunge alla lista dei file descriptor controllati dalla “poll()” del ciclo principale e scrive un prompt nella nuova connessione. La gestione torna al “main_loop()”;

- L’utente invia un comando, quindi nel “main_loop()” dello switch la “poll()” torna il nuovo file descriptor;

- Viene chiamata nuovamente la “handle_input()” del modulo consmgmt che legge il comando dal socket e lo passa alla funzione “handle_cmd()” (consmgmt.c riga 112) che ricerca il comando inserito dall’utente nella lista dei comandi con una “strncmp()”;

- Se la ricerca termina con successo “handle_cmd()” invoca la funzione relativa al comando passandole eventuali parametri (ad esempio, se la funzione prevede la stampa di un output, viene passato il file descriptor della console da cui è stato inviato il comando);

- Il valore di ritorno della funzione appena eseguita viene raccolto da “handle_cmd()”, convertito in un messaggio utilizzando “strerror()”, quindi stampato nella console da cui è stato inviato il comando ed usato a sua volta come valore di ritorno;

- Il controllo torna a “handle_input()” che scrive un nuovo prompt. La gestione torna al “main_loop()”;

- Se l’utente chiude unixterm nel “main_loop()” dello switch la “poll()” torna il file descriptor relativo alla connessione appena chiusa;

- Viene chiamata nuovamente la “handle_input()” del modulo consmgmt che, leggendo 0 byte dal socket, si accorge che la connessione è stata chiusa e quindi rimuove il file descriptor dalla lista controllata da “poll()” e lo chiude.

2.4.2 Connessione Di Un Cavo

Analizziamo cosa succede quando un nuovo cavo richiede una connessione allo switch:

- Nel “main_loop()” la “poll()” torna il file descriptor relativo al socket in ascolto per nuove connessioni dati;
- Viene chiamata la “handle_input()” (datasock.c riga 232) che apre un file descriptor per la nuova connessione tramite la “accept()” e lo aggiunge nella lista dei file descriptor controllati dalla “poll()” principale. La gestione torna al “main_loop()”;
- Il cavo effettua la richiesta per una porta, quindi nel “main_loop()” dello switch la “poll()” torna il nuovo file descriptor;
- Viene chiamata nuovamente la “handle_input()” del modulo datasock che controlla il tipo di richiesta inviata dal cavo e, se tutto va bene, invoca la “new_port_v1_v3()” (datasock.c riga 184);
- “new_port_v1_v3()” chiama “setup_ep()” (port.c riga 148) che si occupa di allocare un nuovo endpoint ed eventualmente una nuova porta e di inserirle nelle strutture dati appropriate;
- Quando “setup_ep()” torna “new_port_v1_v3()” crea un socket unix per lo scambio dei dati e ne comunica il path al cavo;
- La gestione torna al “main_loop()”.

2.4.3 Gestione Dei Pacchetti

Analizziamo cosa succede quando arriva un frame ethernet in una porta:

- Nel “main_loop()” la “poll()” torna il file descriptor relativo al socket dati della porta;

- Viene invocata “handle_input()” (datasock.c riga 232) che legge i dati dal socket e li passa ad “handle_in_packet()” (port.c riga 398);
- Come suggerisce il nome in “handle_in_packet()” si trovano tutte le politiche per la gestione dei frame in ingresso:
  - vengono estratti i pacchetti BPDU e consegnati al sottosistema che cura lo spanning tree;
  - viene aggiornata la tabella hash utilizzando l’indirizzo sorgente (non broadcast) di ogni pacchetto;
  - viene stabilita la porta (o le porte, nel caso di pacchetti non unicast) di destinazione e il pacchetto viene immesso nella coda di uscita packetq con la macro “SEND_PACKET_PORT()” (port.c riga 278) che a sua volta richiama “packetq_add()”;
- La gestione torna al “main_loop()”.

2.5 Spanning Tree

La soluzione che andremo a presentare utilizza determinate proprietà del sottosistema di vde_switch che gestisce il protocollo per la creazione dello spanning tree. Per capirne il funzionamento è necessario conoscere sia la teoria alla base di questo algoritmo distribuito che l’implementazione all’interno di vde_switch.

2.5.1 Descrizione

Lo spanning tree *è un algoritmo utilizzato per realizzare reti complesse con percorsi ridondanti.*

Una LAN complessa può essere costituita da diversi segmenti di rete, connessi tra loro tramite switch, con il vincolo che la topologia di una

LAN non contenga cicli, ovvero che tra ogni coppia di calcolatori esista un solo percorso.

Se così non fosse alcuni pacchetti verrebbero replicati all'infinito sulla rete con risultati disastrosi. Lo switch, infatti, conosce gli indirizzi MAC degli host connessi su ogni segmento, ma se riceve un pacchetto con destinazione sconosciuta o un pacchetto broadcast, lo invia su tutti segmenti. Se esiste un ciclo nella rete il pacchetto raggiungerà nuovamente il segmento da cui è partito venendo nuovamente replicato. Questo porterebbe alla proliferazione di infinite copie dello stesso pacchetto sulla rete e quindi alla saturazione della rete stessa.

Una rete complessa priva di percorsi ridondanti è però estremamente fragile, perchè il guasto di un solo switch o collegamento la partiziona in due reti che non comunicano tra di loro.

In una LAN complessa è necessario che ci siano collegamenti ridondanti, ma che alcuni di questi siano mantenuti disabilitati fino a quando non si rendono necessari per sopperire a guasti di altri collegamenti o switch.

L'algoritmo di spanning tree è un algoritmo distribuito che opera su tutti gli switch, facendo in modo che in ogni istante la rete sia connessa ma priva di cicli, ovvero che il grafo dei collegamenti disponibili abbia una conformazione ad albero.

Ciò si ottiene mediante la creazione di una gerarchia di switch. Uno switch viene individuato come "root" ovvero radice dell'albero coprente e una parte dei collegamenti disponibili viene messa in standby, portando in stato "BLOCKING" alcune delle porte degli switch.

Nel caso in cui un nodo diventi irraggiungibile, oppure cambi il costo di connessione, lo switch cerca di arrivare al nodo attivando i percorsi alternativi che sono disabilitati, ripristinando in questo modo la connettività completa della rete, se possibile.

Nella teoria dei grafi, questo problema è noto come Spanning-Tree (albero di copertura).

Questo processo avviene periodicamente per cui se si scollega uno switch o si interrompe un collegamento lo spanning tree viene ricostruito e la rete continua a funzionare.

L'algoritmo tende automaticamente a mantenere in funzione i collegamenti di capacità superiore, ma talvolta la scelta di collegamenti da mantenere attivi è inadeguata alle caratteristiche della rete o del traffico che la attraversa. Configurando opportuni parametri sugli switch è possibile influenzare sia la scelta del root bridge che la scelta dei collegamenti da mantenere in servizio.

Tale algoritmo è stato inventato da Radia Perlman e standardizzato in IEEE 802.1D.

L'algoritmo di Spanning Tree permette di estendere reti locali mantenendo un buon grado di ridondanza, ma *presenta alcuni limiti*:

- I Tempi di Convergenza, ovvero il tempo necessario al protocollo per reagire al guasto di un elemento della rete o al suo ripristino, tendono a crescere con il numero di switch coinvolti nel processo;
- Il protocollo di spanning tree genera a sua volta traffico sulla rete, che può contribuire alla saturazione delle connessioni.
- La capacità dei collegamenti lasciati in stand-by non può essere sfruttata.

Per ovviare ai sopracitati limiti sono stati sviluppate estensioni al protocollo di Spanning Tree originario.

2.5.2 Protocollo

Entriamo ora nei dettagli dello Spanning Tree Protocol.

Elezione Della Radice

Tutti gli switch di una LAN estesa che supportano il protocollo Spanning-Tree raccolgono le informazioni relative agli altri switch presenti nella rete tramite uno scambio di messaggi. Questi messaggi sono chiamati *BPDU, bridge protocol data unit*. Lo scambio di messaggi porta a questi risultati:

- L'elezione di uno switch "root" nella topologia di rete dello spanning-tree;
- L'elezione di uno switch "designated" per ogni segmento di rete;
- La rimozione di tutti i cicli presenti nel grafo della rete impostando tutte le porte ridondanti in uno stato di "backup".

Nel protocollo Spanning-Tree il root switch è il centro logico della topologia di rete. Tutti i percorsi che non sono necessari per raggiungere la radice da qualsiasi punto della rete vengono configurati come "backup".

Uno switch invia dei frame BPDU utilizzando l'indirizzo MAC della porta come sorgente e come destinazione l'indirizzo multicast riservato al protocollo, ovvero 01:80:C2:00:00:00. Il payload di questi frame include la priorità dello switch, la priorità e il costo della porta da cui è stato inviato. Il protocollo Spanning-Tree utilizza tutte queste informazioni per eleggere il root switch della rete e per stabilire la *root port* e la *porta designated* per ogni segmento di rete. I frame BPDU non vengono mai inoltrati da uno switch ma le informazioni in essi contenute vengono confrontate con lo stato interno di ogni switch e, se la topologia di rete è cambiata, possono generare l'invio di nuovi pacchetti BPDU. L'arrivo di informazioni tramite BPDU genera quindi:

- La notifica dello switch radice;
- Il calcolo per ogni switch della minima distanza tra la radice e lo switch stesso;

- L'elezione di uno switch designated. Questo è lo switch più vicino alla radice, tutto il traffico diretto a root verrà inoltrato ad esso;
- Per ogni switch viene selezionata la porta che permette di raggiungere la radice con il percorso migliore;
- Vengono identificate tutte le porte che partecipano allo Spanning-Tree.

Gli Stati Di Una Porta

Ogni porta all'interno dello Spanning-Tree può essere alternativamente in uno stato *blocking, listening, learning, forwarding o disabled*. Se una rete non ha subito cambiamenti recenti nella topologia allora tutte le porte attive si troveranno negli stati stabili, ovvero forwarding e blocking. I passaggi di stato possono essere:

- Da initialization a blocking;
- Da blocking a listening o da blocking a disabled;
- Da listening a learning o da listening a disabled;
- Da learning a forwarding o da learning a disabled;
- Da forwarding a disabled.

L'algoritmo di Spanning-Tree determina se una porta può assumere lo stato forwarding nel seguente modo:

- La porta viene messa nello stato listening ed aspetta eventuali informazioni che suggeriscano di spostarla nello stato blocking;
- Trascorso un determinato intervallo di tempo la porta passa quindi nello stato learning;

- Nello stato learning la porta continua a non far passare il traffico ma apprende dai pacchetti arrivati le informazioni locali;
- Trascorso un determinato intervallo di tempo la porta passa nello stato forwarding dove continua ad apprendere le informazioni locali ma permette anche l'inoltro del traffico.

2.5.3 Rapid Spanning Tree Protocol

Il protocollo Rapid Spanning Tree, standardizzato nel 1998 in IEEE 802.1W, è stato proposto come evoluzione dello Spanning-Tree classico e, come suggerisce il nome, in caso di cambiamenti nella topologia della rete permette di convergere ad un nuovo albero molto più velocemente.

Studiando il Rapid Spanning Tree Protocol bisogna tenere in considerazione che *dati due BPDU si può sempre stabilire quale contenga le informazioni più utili* confrontando i valori contenuti degli stessi BPDU e, occasionalmente, confrontando i parametri relativi alle porte da cui sono stati ricevuti.

Stati E Ruoli

Nello STP la differenziazione delle porte solamente in base allo stato poteva portare a situazioni di ambiguità, per questo motivo in RSTP gli stati sono stati ridotti a tre: *discarding, learning, forwarding* ed è stato introdotto il concetto di *ruolo* di una porta come variabile associata alla porta stessa aumentandone il numero: *root, designated, backup e alternate*.

root La porta di uno switch che riceve i frame BPDU contenenti le informazioni più utili viene impostata come root. Questa è la porta più vicina allo switch radice in termini di costo. L'algoritmo di Spanning-Tree elegge un unico switch radice nella rete. I BPDU inviati dalla radice sono i più utili rispetto a quelli degli altri

switch. Ovviamente lo switch radice è l'unico nodo della rete a non avere una porta root.

designated Una porta viene marcata designated se invia i frame BPDU più utili al segmento di rete alla quale è connessa. Gli switch STP collegano vari segmenti di rete, in ogni segmento può esserci al massimo un percorso che porta alla radice, altrimenti possono crearsi dei cicli. Tutti gli switch collegati ad un dato segmento restano sempre in ricezione di pacchetti BPDU e rispondono allo switch che invia il BPDU più utile il quale imposterà come designated la porta sulla quale riceve risposta.

backup/alternate Sia le porte backup che quelle alternate corrispondono al ruolo blocked di STP. Una porta viene definita blocked se riceve BPDU più utili rispetto a quelli che può inviare. Una porta *alternate* riceve BPDU più utili da un altro switch collegato allo stesso segmento di rete. Una porta *backup* riceve BPDU più utili da un'altra porta dello stesso switch a cui appartiene collegata allo stesso segmento di rete. Questa distinzione permette ad una porta di backup di passare allo stato designated più rapidamente nel caso di problemi al precedente uplink.

Formato E Gestione Dei BPDU

In STP venivano usati solo 2 flag in un BPDU: topology change (TC) e topology change acknowledgement (TCA). RSTP aggiunge informazioni che permettono sia di codificare il ruolo e lo stato della porta da cui parte il BPDU che di gestire il meccanismo di proposta/accettazione.

In STP uno switch non radice generava un BPDU solo se ne riceveva uno dalla porta root. RSTP prevede che ogni switch invii sulle porte un BPDU contenente le proprie informazioni ogni "hello-time" secondi. Questo permette di usare i BPDU come sistema di keep-alive: se lo switch non riceve pacchetti "hello" per più di tre intervalli di tem-

po consecutivi da un suo vicino allora può considerare obsolete le sue informazioni circa lo stato dello spanning-tree.

Porte Edge

La transizione rapida allo stato di inoltro è senza dubbio la caratteristica più importante di RSTP. Lo STP aspettava che la rete convergesse ad uno stato stabile prima di attivare l'inoltro del traffico su una porta. RSTP, per ottenere nel minor tempo possibile una convergenza sullo stato della singola porta, utilizza un ulteriore identificativo: le porte di tipo edge.

Le porte possono essere configurate come "edge" nel caso in cui siano collegate a LAN foglia e quindi vengono immediatamente configurate per l'inoltro del traffico. Ovviamente l'algoritmo continua a monitorare anche queste porte nel caso in cui un pacchetto BPDU arrivi da queste porte.

Topology Change

In STP la notifica di un cambiamento nella topologia della rete veniva inviata allo switch radice che si occupava di propagarla su tutto l'albero.

RSTP modifica notevolmente il sistema. Una notifica di topology change viene inviata solo se una porta non-edge entra nello stato forwarding. Quando uno switch rileva un cambiamento nella topologia ripulisce le tabelle dei MAC address per tutte le sue porte e per un intervallo di tempo pari a due "hello-time" invia dei BPDU di notifica attraverso le porte attive (root e designated). Se ad uno switch arriva una notifica di topology change ripulisce le tabelle dei MAC address relative a tutte le porte tranne quella da cui è arrivata la notifica e per un intervallo di tempo pari a due "hello-time" trasmette le proprie informazioni in un BPDU con il flag TC impostato.

In questo modo una notifica percorre più rapidamente l'albero di copertura della rete, raggiungendo molto prima soprattutto i nodi più vicini allo switch in cui è stato rilevato un cambiamento.

2.5.4 Implementazione In vde_switch

Il sistema che gestisce l'algoritmo spanning tree in vde_switch segue in generale le specifiche definite in IEEE 802.1W.

La struttura dati "vlst" (fstp.c riga 46) contiene i dati relativi alla situazione dell'albero dei collegamenti per una vlan: il ruolo delle porte, gli eventuali costi, le variabili di timing e una semplice struttura a due bitarray che consente di tener traccia dell'arrivo di pacchetti BPDU in una porta. Lo stato di una porta viene gestito in un array di strutture "vlant[]" (port.c riga 84).

Vediamo cosa succede all'avvio di un'istanza di vde_switch in cui è stato abilitato lo spanning-tree (opzione -F):

- Dal "main()" viene invocata la funzione "fst_init()" (fstp.c riga 690) che inizializza il sottosistema di spanning-tree;
- Durante l'inizializzazione del sottosistema il controllo passa a "fstinitpkt()" (fstp.c riga 539) che forgia un modello di BPDU hello e quindi pianifica l'esecuzione della funzione "fst_hello()" (fstp.c riga 319) ogni intervallo di tempo helloperiod. Il controllo torna a "main()".

A questo punto ogni *helloperiod* secondi il sottosistema *qtimer* invoca "fst_hello()" che:

- Per ogni vlan invoca la "fst_hello_vlan()" (fstp.c riga 250) la quale si occupa di inviare un BPDU di hello ad ogni porta presente nella vlan;

- Ogni *helloperiod* * 3 secondi invoca per ogni vlan invoca la funzione “fst_updatebackup()” (fstp.c riga 307) che per ogni porta di backup[11] controlla se questa ha ricevuto pacchetti BPDU per un intervallo di tempo e, in caso negativo, la rimuove dal gruppo di porte di backup.

Come già accennato nella sezione 2.4.3, il codice di “handle_in_packet()”, se arriva un pacchetto BPDU, chiama “fst_in_bpdu()” (fstp.c riga 423) che processa il BPDU in ingresso. Dopo aver controllato la correttezza del frame ed aver registrato l’arrivo di un BPDU per quella porta, la funzione confronta i nuovi dati arrivati per capire se contengono informazioni utili a cambiare lo stato delle connessioni.

Se dalle comparazioni si deduce una modifica riguardante la radice viene invocata la funzione “fastprotocol()” (fstp.c riga 400) che blocca tutte le porte designed non attive e invia un “ack” in risposta alla notifica processata.

Se dalle comparazioni si deduce un qualsiasi tipo di modifica (anche riguardante la radice) viene invocata la funzione “topology_change()” (fstp.c riga 375) che ha il compito di informare i vicini del cambiamento nella topologia.

Sia “fastprotocol()” che “topology_change()” usano “fst_sendbpdu()” (fstp.c riga 333) per forgiare ed inviare i frame.

[11] Non vi è differenza tra alternate e backup

Capitolo 3

IMPLEMENTAZIONE

3.1 Struttura Modulare

All'inizio di questo progetto, durante la fase di analisi preliminare che mi ha portato alle considerazioni elencate più sotto, ho cercato di studiare una soluzione che si attenesse per quanto possibile ad alcuni principi di buon senso che mi pare quindi giusto menzionare:

K.I.S.S. Ovvero *keep it sweet and simple*. Si tratta di un principio cardine che ha guidato lo sviluppo delle più importanti applicazioni unix. In modo meno stringato si potrebbe dire: "Cerca di creare applicazioni piccole, semplici e generiche. In questo modo causerai meno errori (come conseguenza delle piccole dimensioni e della semplicità) e difficilmente dovrai ri-pensare una buona parte dell'architettura del progetto per comprendere un caso insolito che magari non avevi considerato in fase di progetto (come conseguenza della genericità)";

riutilizzo Agli aspiranti informatici viene spesso ricordato di scrivere codice modulare e riutilizzabile, ma se un programma risolve un problema allora un modulo di un programma risolve un sottoproblema. In generale durante l'analisi di un problema, è buona

norma cercare informazioni anche su argomenti ad esso correlati per capire se la soluzione può esserene influenzata o addirittura basarvisi. Questa considerazione è particolarmente utile e vera se si tenta di arrivare ad una rappresentazione informatica del problema. Fortunatamente la grande disponibilità di software open-source aiuta moltissimo a (per abusare di una citazione) "reggersi sulle spalle dei giganti".

In breve alcune considerazioni fatte prima di iniziare ad architettare la soluzione:

- VDE è un framework che può girare su più OS. La configurazione di un dispositivo di rete è intimamente correlata alla struttura del sistema operativo ed è molto complesso creare un software che se ne occupi agilmente su tutte le piattaforme. Il progetto deve quindi essere almeno diviso in due parti: una standard che si occupi dello switch VDE ed una dipendente dal sistema operativo che si occupi di configurare la rete.
- Non è possibile pensare a priori tutte le varie situazioni d'ambiente e di transizioni di rete in cui il sistema dovrà operare. Il progetto non deve quindi cercare di categorizzare le situazioni o assumere determinati schemi di comportamento, sia per il modulo relativo a VDE che, a maggior ragione, per la parte dipendente dal sistema operativo.
- Il software deve essere altamente "scriptabile" ovvero deve offrire ci la possibilità di essere comandato ed interrogato in qualsiasi momento utilizzando gli strumenti normalmente a disposizione di un amministratore di sistema.

La soluzione finale che propongo, scritta e sviluppata sul sistema operativo Linux, è divisa in tre parti: un software che collega gli switch con cavi virtuali, uno che rileva l'ambiente in cui si trova il sistema ed infine uno che rileva la disponibilità di rete per un'interfaccia.

3.2 vde_autolink

vde_autolink e unixtermcmd sono due programmi che ho scritto per gestire i cavi dello switch VDE.

vde_autolink, come suggerisce il nome, ha un unico scopo: *assicurarsi che una serie di cavi collegati allo switch siano sempre funzionanti*.

Il ruolo di vde_autolink è generico e può essere usato anche in altri contesti meno dinamici di quello che stiamo analizzando. Se ad esempio utilizziamo vde_switch in un sistema non mobile di overlay-network o di vpn, vde_autolink ci assicura una riconessione automatica delle reti virtuali in seguito a temporanei disservizi delle reti reali.

3.2.1 Logica di utilizzo

Prima di addentrarci nei dettagli implementativi di vde_autolink vediamo come usare questo tool.

Un link è un cavo virtuale collegato da un lato allo switch locale e dall'altro ad uno switch su un sistema remoto. Per utilizzare vde_autolink bisogna quindi indicare nella configurazione le peculiarità che deve avere il nostro link automatico in termini di cavo e di sistema remoto.

Un collegamento può essere stabilito utilizzando un qualsiasi cavo del framework VDE ed il sistema remoto può essere raggiugibile utilizzando nomi o indirizzi differenti[1], per cui la configurazione di uno stesso link prevede la possibilità di indicare più tipi di cavo e più host remoti.

Ogni link per essere considerato attivo deve avere un tipo di cavo collegato ad uno degli host remoti. Un link è unico: non è possibile avere più cavi connessi contemporaneamente ad uno o più host tra quelli elencati. Nel caso si volessero più cavi connessi in parallelo ad uno stesso sistema remoto semplicemente si configurano più link.

[1]Si pensi ad una batteria di server in balancing o anche ad un unico host raggiungibile in alcuni punti da una rete IPv4 e in altri da una rete IPv6.

Durante lo sviluppo di vde_autolink si è reso necessario aggiungere allo switch VDE alcune funzionalità (discusse in dettaglio nella prossima sezione). Il codice che le implementa è solamente un'implementazione propositiva, quindi al momento il sistema per lanciare vde_autolink e collegarlo a vde_switch non è molto pratico e lineare. Analizzando la bozza di funzioni assieme a Renzo Davoli e ad altre persone coinvolte nello sviluppo di VDE si è arrivati a teorizzare una possibile soluzione generale, descritta nei capitolo dedicato agli Sviluppi Futuri, che comincerà ad essere implementata non appena avrò finito di scrivere questa "utilissima" tesi di laurea.

Vediamo come eseguire un vde_switch appositamente modificato e le aggiunte fatte:

```
$ ./vde_switch -F -s /tmp/test -M /tmp/test.mgmt
vde: help alink
0000 DATA END WITH '.'
COMMAND PATH       SYNTAX          HELP
------------       --------------  ------------
alink              ============    AUTOLINK MENU
alink/init         ALSOCK          setup autolink ..
alink/rst                          reset autolink ..
alink/reserve                      return portno of ..
alink/dispose      N               dispose port N
alink/onhello      N               start ST monitor
alink/offhello     N               stop ST monitor
.
1000 Success
vde:
```

"help alink" mostra i comandi del sottosistema bozza aggiunto. I comandi vengono usati da vde_autolink tramite la console di management ed è quindi necessario avviare lo switch in modo che apra il socket di management.

Piccola parentesi sugli sviluppi futuri: una volta implementata la nuova soluzione di cui parlavo poco sopra questo sottosistema scomparirà: i comandi relativi alle porte saranno aggiunti al sottosistema port, i comandi relativi al monitoring dello spanning tree saranno integrati in un nuovo meccanismo di debugging dello switch ed infine i comandi per il setup del socket di management autolink semplicemente scompariranno perchè resi superflui dalla messaggistica asincrona integrata nel meccanismo di debugging.

Vediamo quindi come eseguire vde_autolink in modo che utilizzi lo switch appena attivato:

```
$ ./vde_autolink -M /tmp/test.alink -s /tmp/test \
    -S /tmp/test.mgmt
VDEal: help
0000 DATA END WITH '.'
help:         print a summary of mgmt commands
shutdown:     terminate
runscript:    load a config file [args: PATH]
showwires:    list inserted wires
addwire:      add a type of wire, with variables ...
delwire:      delete a type of wire [args: TYPE]
showlinks:    list inserted autolinks
runninglinks: print running links
addlink:      add an autolink [args: NAME REMOTEHOSTS]
dellink:      delete an autolink [args: NAME]
addtypelink:  add a type of wire to named link ...
deltypelink:  delete a type of wire from named link ...
linkonoff:    activate/deactivate autolink ...
.
1000 Success
VDEal:
```

L'opzione "-M" indica a vde_autolink di aprire un socket di management (utilizzato per il momento dall'hack di vde_switch), le opzioni "-s"

e “-S” indicano rispettivamente il socket per le connessioni dei cavi ed il socket di management dello switch.

È possibile ottenere un elenco delle opzioni di vde_autolink semplicemente lanciando il comando senza alcun parametro:

```
$ ./vde_autolink
  -h, --help                  Display this help
  -f, --rcfile                Configuration file
                              (overrides
                              /etc/vde_autolink.rc
                              and ~/.vde_autolinkrc)
  -d, --daemon                Daemonize vde_autolink
                              once run
  -p, --pidfile PIDFILE       Write pid of daemon
                              to PIDFILE
  -M, --mgmt SOCK         [*] Path of the management
                              UNIX socket
      --mgmtmode MODE         Management UNIX socket
                              access mode (octal)
  -s, --sock              [*] Attach to this
                              vde_switch socket
  -S, --switchmgmt        [*] Attach to this
                              vde_switch management socket
[*] == Required option!
$
```

Per usare vde_autolink bisogna configurarlo direttamente dalla console o tramite un rc-file. Nella configurazione bisogna:

- indicare uno o più tipi di cavo (*addwire*);
- inserire uno o più link (*addlink*);

- associare ai collegamenti i tipi di cavo che si vogliono utilizzare (*addtypelink*);
- infine attivare un link (*linkonoff*).

A scopo di esempio ecco il file di configurazione che uso per collegare lo switch locale al mio portatile con il sistema di rete virtuale universitario (tappo e tappo6 sono due entry di /etc/hosts impostate per comodità):

```
addwire SSH dpipe ssh lbigliar@$remotehost \
            vde_plug = vde_plug -p $myport $mysock
addlink VDEUNI tappo6 tappo
addtypelink VDEUNI SSH
linkonoff VDEUNI 1
```

3.2.2 Struttura interna

Spiegherò ora in che modo vde_autolink assicura il collegamento.

Uno dispositivo di rete fisico riesce con buona approssimazione a stabilire se un cavo può essere usato per trasmettere dati semplicemente controllando la presenza o meno di tensione. Nel mondo virtuale non siamo così fortunati, bisogna quindi ricorrere ad altri metodi.

Ho individuato due situazioni che si possono verificare quando un cavo cessa di funzionare correttamente:

- il processo relativo al cavo *termina* come conseguenza di gravi errori nell'invio e nella ricezione di dati sui socket aperti;
- il processo relativo al cavo *"resta appeso"*, ovvero non termina in seguito ad errori nell'invio e nella ricezione di dati sui socket aperti, ma non è più in grado di veicolare i frame ethernet dello switch.

Riparare il link nel primo caso è molto semplice: vde_autolink lancia i cavi come processi figli. Quando un processo termina vde_autolink raccoglie il segnale SIGCHLD e prova a ri-lanciare un cavo dello stesso tipo od uno nuovo (in base alla configurazione).

Riparare il link nel secondo caso non è invece immediato: serve un meccanismo di monitoraggio che invii e riceva periodicamente dei messaggi aventi la funzione di keep-alive e che possa notificare errori in ricezione o in trasmissione. Fortunatamente un sistema similare è già presente in vde_switch ed è la parte del Rapid Spanning-Tree Protocol che si occupa degli hello-BPDU.

A questo punto lo schema di funzionamento di vde_autolink è già definibile:

- all'avvio si apre un canale di comunicazione (per ora doppio) con vde_switch da utilizzare per riservare le porte dei link e per il controllo delle notifiche riguardanti la mancata ricezione di pacchetti hello-BPDU;
- quindi il programma entra in un ciclo infinito aspettando
  - richiesta di una connessione di management;
  - dati su una console di management aperta da un utente o da uno script (con unixtermcmd) che possono interrogare la situazione attuale dei cavi o cambiare le configurazione;
  - notifiche da vde_switch per pacchetti hello non ricevuti;
  - SIGCHLD per la terminazione di un processo relativo ad un cavo.

Quando un utente definisce più cavi e più host questi vengono utilizzati uno ad uno in modo circolare: prima si fanno ruotare tutti i cavi su un host, quindi si cambia host.

vde_autolink decide di cambiare cavo se tra due notifiche di non funzionamento è trascorso un intervallo di tempo minore rispetto ad un valore configurabile.

Per evitare attività non necessarie in caso di assenza di connessione se un ciclo completo di prove cavi-host è durato meno di un determinato intervallo di tempo allora vde_autolink mette sospensione temporanea il link utilizzando un processo dummy che simula un cavo attivo.

3.3 unixtermcmd

unixtermcmd è il secondo programma che ho scritto per gestire i cavi VDE.

A livello pratico questo software non interviene direttamente sui componenti, è solo un software ausiliario che permette di inviare e ricevere comandi a qualsiasi applicativo della suite VDE direttamente da uno script.

Nel framework è già presente *unixterm,* un sistema per interfacciarsi alle console di management che però si limita ad aprire una sorta di terminale in cui l'utente può inviare comandi ai componenti e ricevere informazioni.

Vediamo un esempio di utilizzo. Supponiamo di avere un vde_autolink non configurato in ascolto sul socket "/tmp/test.alink":

```
$ ./unixtermcmd /tmp/test.alink runninglinks
$ echo $?
0
$ ./unixtermcmd /tmp/test.alink showlinks
no autolink defined
$ echo $?
0
$ ./unixtermcmd /tmp/test.alink asdasasd
```

```
$ echo $?
38
$
```

La sintassi è semplice:
unixtermcmd <*socket-management*> <*comando*>.

All'avvio unixtermcmd si connette al socket di management specificato, invia il comando, memorizza l'output e chiude la connessione. A questo punto effettua una scansione dei dati ricevuti attraverso la console di management scartando le righe non necessarie. Alla fine dell'analisi se vi sono informazioni utili queste vengono stampate su STDOUT, dopodichè l'applicazione esce utilizzando come valore di ritorno il valore di ritorno del comando eseguito all'interno della console di management, opportunamente convertito.

Sarebbe perfettamente possibile integrare con poco sforzo unixtermcmd in unixterm inserendo un controllo sui parametri passati all'eseguibile: se viene passato solo il socket di management si esegue il codice di unixterm, altrimenti si esegue il codice di unixtermcmd. Tuttavia, in accordo con gli altri sviluppatori di VDE, non ho intrapreso questa strada poichè una volta ultimate le librerie per l'utilizzo della console di management (descritte nel capitolo sugli Sviluppi Futuri) sia unixterm che unixtermcmd andranno riscritti da capo.

3.4 Il Sistema

Dopo aver descritto i software proposti per il framework VDE studiamo tre programmi che possono essere usati per risolvere, almeno sotto Linux, la parte di problema legata al sistema operativo.

3.4.1 guessnet

guessnet[2] è un sistema scritto da Enrico Zini con l'aiuto di Thomas Hood. Si utilizza per il rilevamento delle reti quando un computer viene spostato tra vari ambienti che non offrono necessariamente il servizio DHCP. guessnet si aspetta in input una lista di possibili profili di rete, ognuno dei quali include una descrizione dei test da effettuare; quindi esegue tutti i test in parallelo e stampa il nome del profilo il cui test ha dato per primo un risultato positivo.

La differenza principale tra guessnet ed altri tool come whereami, intuitively, laptop-net, divine.. sta nel fatto che il primo si preoccupa solamente di stabilire il profilo di rete e non di configurare il sistema. Questa importante caratteristica, che lo rende un modulo generico e flessibile, mi ha fatto scegliere guessnet come parte della soluzione.

I tipi di test che guessnet può eseguire sono:

- Controlli ARP per controllare la presenza di host conosciuti all'interno della rete;
- Controlli sul link-beat;
- Controlli sulle reti Wireless, per rilevare eventuali access point con waproamd;
- Controlli PPPOE, per rilevare la presenza di un concentratore accessibile;
- Controlli basati su script arbitrari, definibili dall'utente.

3.4.2 ifplugd - udev

ifplugd[3] è un demone scritto da Lennart Poettering in grado di rilevare la presenza di segnale di rete su un'interfaccia ethernet. In seguito al

[2] http://guessnet.alioth.debian.org/
[3] http://0pointer.de/lennart/projects/ifplugd/

cambiamento di stato di un'interfaccia (connessa/disconnessa) ifplugd può esegue i comandi specificati nel file di configurazione.

udev[4] è un demone che fornisce una directory dei dispositivi dinamica contenente solo i file che rappresentano i device attualmente presenti. È in grado di creare o rimuovere i file dei dispositivi in "/dev" o di rinominare le interfacce di rete. Quando un dispositivo viene collegato o rimosso dal systema il kernel notifica udev tramite un "uevent". In seguito alla ricezione di un evento udev confronta gli attributi dei device presenti nel sysfs con una serie di regole configurabili. Le regole se hanno corrispondenza con il device possono impostare molte proprietà della rappresentazione userspace del dispositivo e, cosa che ci interessa in particolare, possono eseguire degli script specificati come "action".

3.4.3 Composizione dei software

Dovrebbe essere ormai chiara la composizione della catena di programmi: ifplugd è in grado di capire quando un device ethernet può fruire di una connessione, udev può rilevare la presenza di altri tipi di dispositivi di rete (non ethernet). Entrambi possono lanciare guessnet che cerca di capire in che ambiente ci si trova e di configurare le interfacce di conseguenza. Infine vde_autolink assicura un link di rete virtuale non appena la rete fisica diviene funzionante.

[4]http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev.html

Capitolo 4

SVILUPPI FUTURI

Ci sono molte possibilità di espandere e migliorare il progetto che non ho potuto affrontare per mancanza di tempo. Le seguenti sezioni presentano alcuni spunti di miglioramento.

4.1 Asincronia

Come già accennato nel capitolo precedente una delle funzionalità su cui mi metterò a lavorare appena consegnata la tesi è il supporto dei messaggi asincroni all'interno di vde_switch.

Vari progetti relativi a VDE, tra cui il mio, hanno bisogno di notifiche da parte di vde_switch nel caso in cui si verifichino determinate condizioni, differenti per ogni progetto. Il professor Davoli, coordinatore dei vari progetti, ha notato questa necessità comune e ha quindi proposto di implementare all'interno dello switch un sistema che permetta di mettere in modalità di debugging vari aspetti dei sottosistemi dello switch.

Per rendere l'idea si immagini un utente che imposti il debugging relativo, ad esempio, alla ricezione dei pacchetti hello-bpdu tramite una console di management. Egli si vedrà stampato un messaggio di notifica sulla console ogni volta che lo switch riceve un frame hello-bpdu.

4.2 libvdemgmt

Il successivo problema che si presenta è la creazione di una libreria che permetta di facilitare la scrittura di programmi che vogliono interfacciarsi con la console di management dello switch vde.

Sto collaborando in questo senso con Alessio Caprari per definire l'Application Program Interface della libreria ed anche alcune logiche di funzionamento.

Siccome la libreria dovrà comprendere anche il supporto per i messaggi asincroni, questo punto è intimamente legato con il precedente.

4.3 Cavi VDE

Nel progetto iniziale avevo inserito anche un studio per rendere VDE un sistema "ip over everything", ma non ho avuto tempo per affrontarlo e produrre qualcosa di significativo.

Su questo argomento ho solo due vaghe idee. La prima è un elenco di possibili transport e metodi da usare per riuscire a veicolare il traffico ip. La seconda è cercare di definire uno scheletro generico per facilitare il lavoro di uno sviluppatore che vuole creare il proprio cavo, in modo che egli debba solamente implementare alcune funzioni da passare ad una routine che poi si occupa di tutto il resto.

4.4 Action

Ritengo che vde_autolink sia un programma abbastanza generico e versatile, ma per renderlo ancora più flessibile volevo introdurre la possibilità di inserire delle "action" da intraprendere quando un link torna disponibile dopo un periodo di down e viceversa.

4.5 Esempi

A corredo del software ritengo di non aver inserito abbastanza script di esempio per l'utenza. Siccome questi script devono insegnare e proporre delle metodologie non devono essere numerosi ma devono al contempo coprire più casi possibili. Creare esempi simili richiede tempo per riflettere e familiarizzare con il progetto.

4.6 Bi-mobile

Un'idea recente, scaturita in seguito ad una chiacchierata con un collega, è la possibilità di aggiungere un meccanismo che consenta a due switch in movimento di avere un collegamento diretto tra loro, senza nessuno dei due abbia a disposizione informazioni dettagliate sull'ambiente in cui si trova l'altro. Per ora non dico altro :)

Capitolo 5

CONCLUSIONI

Questo lavoro di tesi ha analizzato la possibilità di realizzare un sistema che permetta di configurare in modo automatico un collegamento tra un vde_switch mobile e uno in una locazione statica.

Come primo passo è stato necessario analizzare sia la filosofia di sviluppo del framework VDE e del progetto Virtual Square che tutte le parti dello switch virtuale prima a livello teorico e poi studiandone direttamente l'implementazione C.

Il lungo lavoro di analisi preliminare si è rivelato molto utile perchè da un lato mi ha fornito una metodologia di progetto dinamica e dall'altro lato mi ha permesso di orientare la mia soluzione per farla poggiare su parti già scritte e testate, senza però forzare la soluzione o stravolgere la logica del problema.

Nella fase di progettazione ho cercato di attenermi a criteri di modularità e riusabilità. Ritengo di aver rispettato questi principi poichè alcune parti software sviluppate sono state già riutilizzate in altri progetti differenti.

Per motivi di tempo non sono riuscito a sviluppare soluzioni per alcuni problemi che, anche se non intimamente legati con l'oggetto di questa dissertazione, avrebbero abbellito ed arricchito ulteriormente il mio progetto.

FINE :)

Bibliografia

[1] R. Davoli. VDE: Virtual Distributed Ethernet. In *Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities (TRIDENTCOM'05)*. http://ieeexplore.ieee.org/iel5/9524/30172/01386197.pdf.

[2] R. Davoli. *A Very Virtual Seminar about Virtual Square*. In Web-MINDS final meeting of the project Genoa, October 27 2006. http://web-minds.consorzio-cini.it/activities/pdfriunionefinale/WP5_davoli.pdf.

[3] R. Davoli. *Virtual Square.* *Homepage: http://www.virtualsquare.org/*.

[4] R. Davoli. Virtual Square. In *Séminaire - PREUVES, PROGRAMMES et SYSTÉMES, September 2006*. http://www.pps.jussieu.fr/seminaire/sem2006/abstracts/davoli.html.

[5] IEEE. *Standard 802.1D - Local and metropolitan area networks Media Access Control (MAC) Bridges*. 2004.

[6] Cisco. *Understanding Spanning-Tree Protocol*.

[7] IEEE. Standard 802.1W - *Rapid Reconfiguration of Spanning Tree*. 2005.

[8] Cisco. *Understanding Rapid Spanning Tree Protocol*.

[9] Wikipedia. http://en.wikipedia.org/wiki/Main_Page.

[10] E. Zini. guessnet - Documentation.
Homepage: http://guessnet.alioth.debian.org/.

[11] L. Poettering. ifplugd - Documentation.
Homepage: http://0pointer.de/lennart/projects/ifplugd/.

[12] D. Drake. *Writing udev rules*.
http://www.reactivated.net/writing_udev_rules.html.