ALMA MATER STUDIORUM - UNIVERSITA'DI BOLOGNA
CAMPUS DI CESENA
SCUOLA DI INGEGNERIA E ARCHITETTURA

CORSO DI LAUREA IN INGEGNERIA ELETTRONICA, INFORMATICA
E TELECOMUNICAZIONI

SPERIMENTAZIONE DI PROTOCOLLI DI ROUTING IP SU PIATTAFORME RASPBERRY PI

ELABORATO IN

APPLICAZIONI E TECNICHE DI TELECOMUNICAZIONI

RELATORE
CERRONI WALTER

PRESENTATA DA
SPARNACCI NICOLA

CORRELATORE
CALLEGATI FRANCO

II SESSIONE DI LAUREA
ANNO ACCADEMICO 2015-2016

1. Introduzione

Da qualche anno il numero di reti IP connesse ad Internet ha un tasso di crescita esponenziale: alcune stime riportano che il loro numero raddoppia ogni 9 mesi. Un problema che si delinea è il dimensionamento dei router (in particolare della memoria) poiché lo scambio di informazioni tra i gateway porta alla costruzione della routing table che, per una rete così grande, ha dimensioni ragguardevoli. Esse dovranno essere sviluppate in modo automatico dal router in modo da sopperire ai cambiamenti dovuti ad una rete che viene aggiunta piuttosto che una che viene disconnessa.

L'obbiettivo di questa tesi è quello di creare una network caratterizzata da tre aree, composte da topologie non banali di nodi routing, che instradano pacchetti su un grande quantitativo di LAN. Una volta creata, si andrà a studiarne il comportamento facendo comunicare gli host dell'area 1 con quelli dell'area 2, passando attraverso l'area 0. Ci si focalizzerà soprattutto sul comportamento che avranno i protocolli di routing come l'OSPF e come verranno influenzate le tabelle di inoltro dalle eventuali modifiche apportate alla rete.

Per simulare una rete di questo tipo, si è scelto di utilizzare macchine Linux in quanto sono le più "adatte" a lavorare su reti di calcolatori: in particolare, visto il costo, la trasportabilità e necessità di spazio, la rete sarà composta da dispositivi Raspberry Pi Model B+, dal costo di 25€ l'uno. Questa scelta è stata fatta perché per simulare una rete di questa complessità sarebbe necessario un enorme impiego di energia elettrica, di materiale, di soldi e di spazio.
Un ulteriore risparmio di risorse per creare le reti più complesse viene dato dalla virtualizzazione namespace Linux tramite l'emulatore Mininet, in grado di creare host virtuali che forniscano le stesse funzionalità di un host fisico e la creazione di switch virtuali.

In questo modo si dimostrerà che con un numero esiguo di dispositivi fisici (7 Raspberry) è possibile creare una rete funzionante con 7 LAN e 253 indirizzi disponibili ciascuna, per un totale di 1771 host potenzialmente utilizzabili.

Grazie alle piccole dimensioni dei Raspberry, la rete è stata implementata utilizzando uno spazio relativamente piccolo; infatti, se lo si desiderasse, si potrebbero trasportare tutti i dispositivi e i cavi in una piccola borsa. Questo lavoro potrebbe essere utile ai docenti universitari che insegnano il comportamento della rete Internet per fornire un esempio concreto agli studenti del corso.

Questa tesi sarà strutturata in tre capitoli in cui verranno trattati alcuni temi legati alla teoria delle reti che sfruttano protocolli, dettagliati gli strumenti utilizzati e la creazione pratica della network. Per distinguere i comandi Linux dal semplice testo verrà utilizzato il carattere `menlo` o verranno creati dei riquadri in cui verranno scritte le istruzioni da eseguire per ottenere il risultato desiderato.
Nel seguito della tesi, i comandi da inserire in linea di comando saranno preceduti dai simboli # o $. Essi hanno un preciso significato:

- # : i comandi vanno eseguiti coi privilegi di root, ottenibili col comando `sudo` "super user do" prima dell'istruzione da eseguire. Successivamente verrà richiesta l'immissione della password;
- $: i comandi possono essere eseguiti come letti senza dover digitare `sudo`.

2. Instradamento delle informazioni in Internet

2.1 Instradamento

Il compito principale dello strato 3 dell'architettura OSI è il trasferimento di unità informative tra entità di livello di rete fornendo un servizio di trasferimento delle unità dato alle entità di strato superiore, quello di trasporto. Le unità dato in questo strato sono chiamate *pacchetti.* Essi mascherano alle entità di trasporto il tipo di rete che viene interessata e le caratteristiche di commutazione della stessa. Nello strato di rete, possono essere identificate quindi due funzioni:

- *Forwarding:* quando un pacchetto arriva all'ingresso del router, questo deve spostare il pacchetto all'appropriata uscita. Si tratta di un'operazione che il router effettua in locale;
- *Routing:* è la funzione che determina il percorso che il pacchetto deve prendere per arrivare dal mittente al destinatario. Gli algoritmi che calcolano questi percorsi sono detti algoritmi di routing.

L'algoritmo di routing viene usato nel nodo di una rete a commutazione di pacchetto per determinare il nodo successivo al quale trasmettere il pacchetto ricevuto, sulla base della destinazione del pacchetto stesso. Perché si renda necessario avvalersi degli algoritmi di routing, la destinazione del pacchetto non deve trovarsi nella stessa rete del nodo: gli algoritmi di instradamento vengono usati solo per la consegna indiretta. L'algoritmo deve avere le seguenti caratteristiche:

- *semplicità*, richiesta per mantenere il costo dell'elaborazione basso;
- *robustezza e stabilità*: deve funzionare anche in caso di comportamenti anomali della rete che è in continuo cambiamento e potrebbero venir meno dei nodi o rami di network;

Per studiare un nodo gli viene associato un modello basato sulle sue caratteristiche più importanti. Le principali funzioni svolte da un nodo a commutazione di pacchetto sono:

- ricezione delle unità informative sui rami entranti e loro elaborazione;
- instradamento delle unità informative, cioè selezione del ramo uscente in funzione dell'indirizzo di destinazione dell'unità stessa, realizzando ove richiesto funzioni di controllo di flusso e di congestione;
- trasmissione delle unità informative sul canale trasmissivo uscente.

Ogni nodo a commutazione di pacchetto comprende delle unità di memoria, o buffer, che si rendono necessarie ogni volta che un stessa funzione, per esempio l'instradamento o la trasmissione sullo stesso canale trasmissivo, sia richiesta contemporaneamente da più unità informative. I buffer consentono dunque di svolgere sequenzialmente la stessa funzione richiesta da più unità informative. Occorre osservare che le funzioni svolte in un nodo sono molteplici e appartengono a tutti e tre gli strati inferiori del modello OSI, e cioè strato fisico, strato di collegamento e strato di rete. In particolare il nodo svolge tutte queste funzioni necessarie per attuare i protocolli di comunicazione per ognuno di questi tre strati sia in ricezione sia in trasmissione. Ricordando le funzioni di ogni strato, il nodo svolge a livello 2 funzioni di ricezione e trasmissione di trame sui

collegamenti entranti e uscenti, rispettivamente, mentre svolge a livello 3 funzioni di instradamento e di controllo congestione dei pacchetti ricevuti e da trasmettere.

La struttura di un generico nodo di rete a commutazione di pacchetto è rappresentata da tre stadi sequenziali, tutti dotati di unità di elaborazione, dei quali solo gli ultimi due sono dotati di memorizzazione. Il primo stadio rappresenta le funzioni di elaborazione delle unità informative ricevute (trame) per lo svolgimento di tutte le funzioni protocollari di livello 2. È ragionevole ipotizzare che queste risorse siano disponibili in ogni interfaccia di collegamento entrante. Il payload di queste unità informative, cioè i pacchetti, viene elaborato nel secondo stadio per attuare le funzioni di instradamento, ovvero la scelta del collegamento uscente dal nodo. Una volta applicato l'algoritmo di routing, il pacchetto instradato giunge nel terzo stadio dove si trova un buffer per la linea uscente.

Autonomous systems: divisione in aree
Studiando gli algoritmi di routing, si intuisce che l'ipotesi di utilizzare lo stesso algoritmo su ogni router (che realisticamente non saranno tutti uguali) è una semplificazione per almeno due ragioni:

- Scala: all'aumentare del numero di router, l'overhead coinvolto nella computazione, memorizzazione, comunicazione di informazioni di instradamento, diventano proibitivi. Immagazzinare questi overhead su ogni host richiederebbe chiaramente un grande quantitativo di memoria. Soprattutto se si considerano algoritmi iterativi come ad esempio il *distance-vector* che con un numero troppo alto di nodi potrebbero non convergere mai.

- Autonomia amministrativa: idealmente, un'azienda vorrebbe che fosse possibile creare e amministrate autonomamente la propria rete, mantenendo la possibilità di comunicare con l'esterno e la possibilità di non mostrare a terzi la struttura della loro network.

Entrambi questi problemi possono essere risolti organizzando i router in *sistemi autonomi (autonomous systems: AS)* dove ogni AS corrisponde in un gruppo di router tipicamente sotto lo stesso controllo amministrativo (ad esempio può essere la rete dell'azienda sopracitata). All'interno dell'area, tutti i router usano lo stesso algoritmo di routing, chiamato *intra-autonomous system routing protocol* e hanno informazioni sugli altri. Configurata un'area, sarà poi necessario collegare tutte le aree insieme tramite router di gateway posti nei bordi delle aree che dovranno avere un loro protocollo. Riassumendo, i protocolli si possono distinguere in due classi:

- Interior Gateway Protocol (IGP) o intra-autonomous system routing protocol: protocolli di instradamento interni al sistema autonomo, come RIP e OSPF;

- Exterior Gateway Protocol (EGP): protocolli che definiscono le modalità di comunicazione tra router appartenenti a diversi sistemi autonomi come il Border Gateway Protocol (BGP);

Qualsiasi sia l'algoritmo usato, il risultato è la definizione della tabella di instradamento che definisce le modalità di inoltro di ogni pacchetto IP. In entrambe le modalità di instradamento (diretto o indiretto) la tabella deve specificare anche l'interfaccia della macchina su cui deve essere inviato il pacchetto. Questi servizi non sono però in generale sufficienti a trasmettere un pacchetto: nel caso di reti multi-punto deve essere specificato l'indirizzo fisico di rete. Di questo fatto se ne occupa il protocollo ARP che associa da un indirizzo IP l'indirizzo MAC della sua interfaccia.

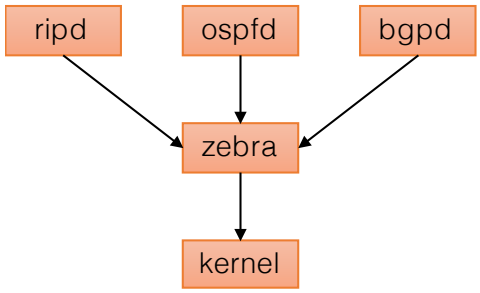
I protocolli storicamente utilizzati come intra-routing protocol sono: il Routing Information Protocol (RIP) e l'Open Shortest Path First (OSPF). Il RIP è un distance-vector protocol, utilizzato soprattutto nelle reti di piccola dimensione, che usa il conteggio di hop come calcolo del costo metrica. Il costo è indicato in hop e rappresenta la quantità di sottoreti attraversate per raggiungere la destinazione dalla sorgente.

L'OSPF è l'evoluzione del RIP e ogni router dell'area che lavora con questo protocollo si crea un grafo della topologia e, localmente, utilizza l'algoritmo Dijkstra che cerca il percorso a minor costo possibile. Un sistema autonomo OSPF può essere configurato gerarchicamente nelle aree. Ogni area esegue il proprio OSPF e ogni area ha uno o più router di bordo responsabile dell'inoltro indiretto dei pacchetti fuori dall'area. Il pacchetto OSPF è incapsulato direttamente in IP.

2.2 Quagga: protocollo di routing tra aree

Il sistema operativo Raspbian è compatibile col pacchetto quagga, un software opensource che fornisce un servizio di routing basato sul TCP/IP con protocolli come RIPv1, RIPv2, OSPFv2, BGP-4. In aggiunta ai protocolli di routing tradizionalmente IPv4, supporta i protocolli più avanzati e recenti come quelli basati su IPv6.

Basato su un'avanzata architettura software che fornisce un servizio di routing ad alta qualità, ha un'interfaccia interattiva per ogni protocollo che si adatta a molte delle esigenze di instradamento. Quagga si comporta come un router dedicato: la macchina scambia informazioni con gli altri router usando dei protocolli di comunicazione. Queste informazioni vengono usate per aggiornare le routing table del kernel in modo dinamico così che il giusto dato giunga alla giusta destinazione. La struttura d'interazione tra i processi è:



Come si intuisce dalla figura, quagga utilizza una collezione di demoni che lavorano insieme per creare le tabelle di instradamento. Tra questi, il più importante è zebra, che permette agli altri demoni di andare a modificare la kernel routing table. Questo è il software addetto alla gestione della tabella di routing ed è lui a permettere l'aggiornamento dinamico delle route. Tradizionalmente, i router basati su sistemi UNIX sono configurati tramiti i comandi `ifconfig` e `route`. A differenza di questi sistemi, dove i comandi richiedono i privilegi di root, quagga utilizza due modalità d'utente:

- modalità normale, dove è possibile vedere lo stato del sistema;
- modalità *enable*, dove è possibile agire per cambiare le impostazioni di sistema.

Il pacchetto *quagga*, che comprende tutti i protocolli necessari per rendere il Raspberry un router completo, è installabile utilizzando il comando

```
# apt-get install quagga
```

L'installazione aggiungerà la cartella quagga al percorso `/etc/quagga` . All'interno della nuova cartella si trova il file *daemons*: un file testuale che è necessario modificare per attivare correttamente i protocolli. All'apertura del file si leggerà una colonna di uguaglianze, dove accanto al nome di ogni demone è presente la stringa "no" che li mantiene disattivi. Per l'attivazione è

necessario sostituire la stringa “yes” accanto al demone scelto. Questa operazione conferisce al demone anche la priorità di esecuzione.

Relativamente al protocollo che si intende attivare è necessario creare un file di configurazione del tipo `<protocollo>d.conf` nel percorso in cui è situato il file daemons.

Il pacchetto d’installazione mette a disposizione anche della documentazione, in particolare degli esempi di file configurazione nel percorso `/usr/share/doc/quagga/examples` . Questi esempi sono molto comodi perché possono essere copiati nella cartella dei file configurazione coi comandi:

```
# cp /usr/share/doc/quagga/examples/zebra.conf.sample  /etc/quagga/zebra.conf
# cp /usr/share/doc/quagga/examples/ospfd.conf.sample  /etc/quagga/ospfd.conf
 # cp /usr/share/doc/quagga/examples/vtysh.conf.sample  /etc/quagga/vtysh.conf
```

Una volta configurati i file di configurazione, come si vedrà in seguito nel dettaglio, sarà possibile col comando `vtysh` entrare nella shell dedicata a quagga. Nella Virtual TeleYpe SHell è possibile vedere la tabella di instradamento creata dai protocolli, sia per la consegna diretta sia per la consegna indiretta (dove verrà indicato il più vicino router intermedio).
Tramite la shell è anche possibile interagire direttamente con tutti i demoni. Per fare questo è necessario passare dalla modalità normale alla modalità enable utilizzando il comando configura terminal.
Sotto il sistema operativo Debian e Raspbian, per poter visualizzare correttamente il contenuto della shell è necessario aggiungere la riga `VTYSH_PAGER=more` al file `/etc/environment`. In caso contrario verrà stampata a schermo la scritta (END) e per poter interagire con la shell si dovrà spingere il tasto “Q” ad ogni avvio.

L’interfaccia vtysh è anche molto utile per studiare e lavorare i sistemi CISCO, poiché i comandi e le impostazioni sono veramente molto simili.

3. Strumenti software e hardware utilizzati

3.1 Raspberry Pi

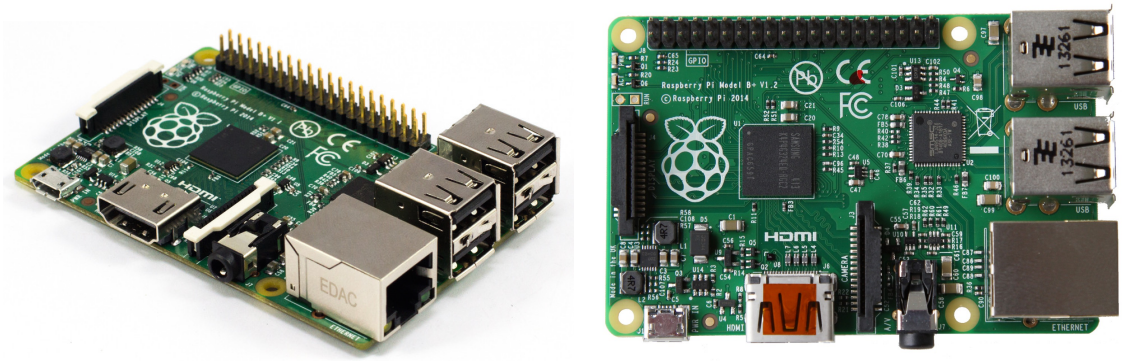
Per poter creare delle reti di calcolatori sempre più complesse è necessario sostenere costi sempre più elevati, con un alto dispendio di risorse energetiche e di manutenzione. Per ovviare parzialmente a questo problema di risorse, si è cercato di creare un computer di dimensioni e costi contenuti che però mantenga intatte le stesse funzionalità di base, sopportando anche una penalità a livello di prestazione.

Un dispositivo molto interessante sotto questi punti di vista è il Raspberry Pi, un calcolatore su una singola board (SoC) creato dalla fondazione inglese Raspberry Pi Foundation con lo scopo di promuovere l'insegnamento, soprattutto a livello scolastico, dell'informatica. Il modello più completo tra quelli disponibili è il Model B+ , basato sull'architettura ARM e con le seguenti caratteristiche tecniche:

- CPU: 700 MHz (con possibilità di Overclock) ARM1176JZF-S core (famiglia ARM11);
- GPU: Broadcom VideoCore IV, OpenGL ES 2.0, 1080p30 h.264/MPEG-4 AVC high-profile decoder;
- Memory (SDRAM): 256 Megabytes (MiB);

E interfacce Input/Output:

- Video outputs: RCA composito, HDMI;
- Audio outputs: 3.5 mm jack, HDMI;
- Slot di memoria microSD;
- Porta Ethernet 10/100 RJ45;
- 4 porte USB 2.0 hotplug e comportamento di overcurrent;
- 40 GPIO pins.



Molto importanti sono anche le sue dimensioni. Infatti, un Raspberry misura 85.60mm x 56mm x 21mm (che equivalgono circa alla grandezza di una carta di credito) e ha un peso di 45g. Queste caratteristiche ne consentono l'utilizzo anche in situazioni in cui le dimensioni e il peso richiesti siano ridotti senza avere pericoli di rotture poiché è possibile acquistare o creare un case protettivo per ripararlo da urti e agenti esterni.

Nel Febbraio 2015 è stato rilasciato un aggiornamento al modello B+ portando la CPU ad una frequenza di 900 MHz grazie al quad-core ARM Cortex-A7 e arrivando a 1 Gb di memoria RAM statica.

Questo dispositivo per il basso costo, per le piccole dimensioni e per la vasta gamma di possibili utilizzi, rappresenta un punto fermo nei sistemi embedded di nuova generazione ed è stato sostituito ai sistemi più costosi come, ad esempio, le FPGA.

Il punto di forza del sistema Raspberry Pi è, ancor più del basso costo, la flessibilità. Questa è garantita dal sistema operativo Linux supportato che, in ogni sua variante o distribuzione, permette di sfruttare a pieno tutte le potenzialità di un calcolatore. Nella fattispecie, esistono distribuzioni ad-hoc come RaspBian, una versione adattata di Debian. Come si vedrà, la sintassi dei comandi è pressoché identica a quella di un personal computer con sistema operativo Debian.

INSTALLAZIONE SISTEMA OPERATIVO

Prima di avviare la procedura d'installazione è necessario formattare la scheda SD che si andrà a montare sul Raspberry con un filesystem di tipo FAT32 e che abbia una memoria sufficiente da ospitare il sistema operativo scelto.
L'installazione del sistema operativo va eseguita in modi differenti a seconda del sistema sul quale si sta lavorando (Mac, Linux, Windows). Per ciascun sistema operativo sono presenti vari tools reperibili online che permettono di fare questa operazione in automatico ma è anche possibile installare correttamente il sistema operativo scelto per il Raspberry Pi da riga di comando. Su Mac, ad esempio, dopo aver formattato con l'applicazione Utility Disco (Applicazioni>Altro>Utility Disco), basta seguire i seguenti passi dall'applicazione Terminale (Applicazioni>Altro>Terminale):

- cercare la scheda tra i dispositivi rilevati dal computer:

```
diskutil list
```

- smontare il dispositivo per poterlo scrivere. Va scelto il disco e non la partizione. Ad esempio `disk2` va bene mentre `disk2s1` no. Per farlo, eseguire il comando:

```
diskutil unmountDisk /dev/<disk# da diskutil>
```

- selezionare l'immagine e scriverla nella scheda col comando dd:

```
sudo dd bs=1m if=<percorso_immagine>.img of=/dev/<disk# da diskutil>
```

Quest'ultimo comando richiederà alcuni minuti. Quando il processo sarà completato sarà possibile inserire la scheda SD nello slot del Raspberry e il dispositivo si avvierà con sistema operativo scelto.

Se si lavora con Linux gli step sono simili a quelli per Mac. Per formattare la scheda SD su Ubuntu si può utilizzare il comando `mkdosfs` da linea di comando utilizzando il flag "-F 32" e il percorso del file speciale associato al dispositivo, visibile tramite il comando `df`.

Come per Mac, l'immagine può essere scritta anche da linea di comando eseguendo i seguenti comandi:

```
df -h
```

- cercare la scheda tra i dispositivi correntemente montati:
- La colonna di sinistra da il nome dell'SD card che sarà scritto come `/dev/sdd1` o `/dev/mmcblk0p1` che però rappresentano le partizioni. Si deve scrivere su tutta la scheda e non solo su una partizione, perciò si devono smontare le partizioni. Per ogni partizione digitare:

```
unmount /dev/sdd# , dove # è il numero di partizione
```

- Infine, si deve scrivere su tutta la scheda SD quindi nel file destinazione non vanno messi numeri:

```
sudo dd bs=1m if=<percorso_immagine>.img of=/dev/sdd
```

Il comando dd non da informazione durante l'esecuzione quindi potrebbe sembrare che il computer si sia bloccato. Non è così, basta solo attendere qualche minuto e tutto andrà a buon fine.

Su Windows, per prima cosa si deve inserire la microSD e verificare che il suo formato si FAT32. In caso contrario deve essere formattata e per farlo si può utilizzare *SD Association's Formatting Tool* dal link https://www.sdcard.org/downloads/formatter_4/eula_windows/ , installarlo e seguire le semplici istruzioni a video per formattare.

Una volta scaricata la distribuzione voluta si può scrivere scrivere la SD utilizzando un altro tool chiamato *Win32DiskImager* scaricabile gratuitamente da http://sourceforge.net/projects/win32diskimager eseguendola da amministratore. Una volta aperta, si deve scegliere la distribuzione da installare (che deve essere in formato .img) e la microSD sulla quale si vuole scrivere. Attenzione a scegliere la scheda SD corretta perché in caso di errore i dati dell'altra scheda andrebbero persi. A questo punto, dopo aver cliccato "write" la scheda è pronta all'uso.

3.2 Virtualizzazione su Linux

La virtualizzazione è una metodologia che consente la suddivisione di risorse di un calcolatore in molteplici ambienti di esecuzioni che siano Virtual Machine o Virtual Environments. Questo garantisce:

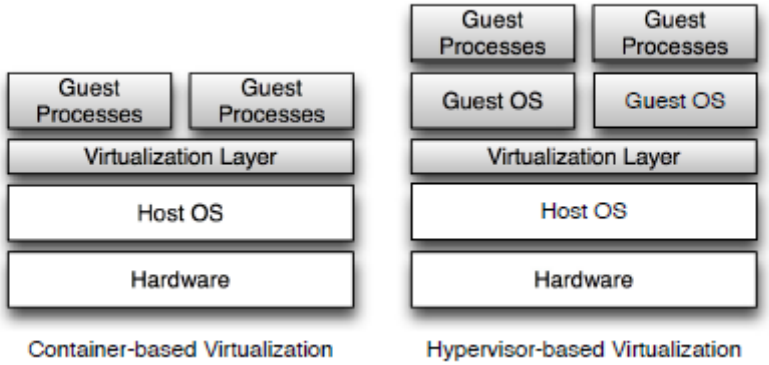
- risparmi sui costi dovuti alla riduzione dell'hardware non essendo più necessaria una macchina fisica per applicazione;
- crescente ripristinabilità di sistema in caso di guasti, poiché ogni ambiente virtuale può essere gestito come un file semplicemente trasportabile;

Esistono tre modi in cui un sistema può essere virtualizzato. La prima tecnica è l'***Emulazione*** dove vengono simulati tutti i componenti hardware. Questa tecnica permette di utilizzare un arbitrario sistema operativo ospite senza alcuna modifica al costo di un - grande - *overhead* aggiuntivo che però ne appesantisce il calcolo delle istruzioni. Per questa tecnica è richiesta la presenza di un Virtual Machine Monitor (VMM) o *Hypervisor* per analizzare il codice e renderlo sicuro in run-time.

La seconda tecnica è la ***Paravirtualizzazione*** dove la maggior parte del lavoro viene eseguito dal codice del sistema operativo ospite che viene modificato per supportare la presenza del VMM ed evitare un uso superfluo di istruzioni privilegiate. Per interfacciarsi con l'*Hypervisor* si rendono necessarie modifiche consistenti al kernel del sistema ospite (che può essere molto dispendioso in termini di tempo e difficoltà).

Queste due tecniche, come detto, introducono *overhead* ed una pesantezza nella gestione non trascurabile dovuta al fatto che ogni macchina virtuale ha il proprio sistema indipendente. Una soluzione a questo inconveniente è nella terza tecnica di virtualizzazione chiamata ***Container-based*** la quale ha come caratteristica predominante una leggerezza di virtualizzazione.

Linux permette di creare ambienti di virtualizzazione a *container*, eseguendo diversi ambienti Linux virtuali isolati tra loro su una singola macchina Linux reale. I container eseguono le istruzioni direttamente sulla CPU senza emulazione o compilazione just-in-time che, come visto per le altre due tecniche, comporterebbe un overhead e grazie all'esecuzione diretta delle istruzioni è richiesto un quantitativo di risorse basso. Il kernel e i programmi base vengono quindi condivisi da guest garantendo una limitazione dell'overhead attorno all'1-3%.



Per l'utente, ogni container è visto come un singolo sistema operativo indipendente poiché l'isolamento è garantito dal *namespace kernel.* Si ha questa illusione perché molte risorse vengono

messe in un livello di namespace, portando l'utente a pensare che il container sia il suo sistema. L'isolamento dei namespace permette di isolare tra loro gruppi di processi affinché non siano loro visibili risorse utilizzate da altri gruppi di processi.

3.3 Mininet: network emulator

Mininet è una piattaforma di emulatori di rete che sfrutta la virtualizzazione basata su processi per lanciare hosts e switch su un singolo kernel, in grado di fornire processi individuali con differenti interfacce di rete, tabelle di routing e tabelle ARP. Permette di creare kernel o user-space OpenVSwitch, controllori per controllare gli switch e host per dialogare con la rete virtuale. I nodi virtuali sono creati come processi in namespaces separati usando Linux NW namespaces e sono collegati tra loro con dei collegamenti Ethernet virtuali (veth pairs). Questo ambiente di emulazione è scritto quasi totalmente in Python ma il nucleo è scritto in C.

Su questi nodi girano i software di rete standard Linux e i suoi switch supportano il protocollo OpenFlow per un routing altamente flessibile e per la simulazione di Software Defined Network. Mininet supporta la ricerca, lo sviluppo, il testing e il debug che permettono di avere a portata di un unico calcolatore un'intera rete: questo permette di simulare topologie di rete complesse senza dover utilizzare fisicamente nessun dispositivo hardware. Mininet mette a disposizione una Commando-Line Interface per poter interagire con la rete, fare il debug o fare ampi test di funzionalità.

Le reti virtuali create da Mininet utilizzano codice "reale" che include le applicazioni standard dei sistemi Linux/Unix come il Linux kernel e la network stack. L'applicazione funziona per tutte le estensioni di kernel che sono compatibili con la virtualizzazione, già discussa, dei network namespaces e in futuro sarà possibile utilizzare Mininet su sistemi operativi diversi da Linux senza la necessità di installare una macchina virtuale. Grazie a questo, il codice sviluppato su Mininet per un controllore OpenFlow, o switch virtuale, o host, può essere trasferito a sistemi reali con dei cambiamenti minimi per test reali, valutazione delle performance e impiego. Questo significa che un progetto funzionante su Mininet è trasferibile direttamente all'hardware switch.

Quasi tutti i sistemi operativi che virtualizzano delle risorse di computing usano la *process abstraction*. L'implementazione dei *network namespace* di Linux, permette di ottenere una virtualizzazione leggera che rende un singolo sistema come una rete completa, che esegue lo stesso kernel e user-code per fornire processi individuali con separate interfacce network, tabelle di routing e tabelle ARP. Mininet può, inoltre, creare degli switch OpenVSwitch nel kernel o in diversi namespaces per avere una maggior distinzione tra nodi e gli host che vengono creati si comportano esattamente come se fossero delle macchine fisiche: essi possono quindi eseguire programmi arbitrari, anche se installati nel sistema Linux sottostante. Ogni nodo è collegato agli altri tramite dei collegamento Ethernet virtuale (veth pairs), le cui caratteristiche come la velocità, la banda, il delay, possono essere modificate per rendere più veritiere l'emulazione.

Per fare un prototipo di una rete ci sono tre differenti possibilità: testbed hardware, fare simulazione o emulare la rete. L'applicazione Mininet combina molti degli aspetti positivi degli emulatori, testbed hardware e dei simulatori le cui caratteristiche sono riportate nella seguente tabella.

Piattaforma	Vantaggi	Svantaggi
Hardware Testbed	• veloce • accurato: da i reali risultati	• costoso • difficile fare delle modifiche • non condivisibile
Simulatore	• economico • flessibile • può essere più veloce della realtà	• può dare risultati non credibili • potrebbe non eseguire codice del sistema operativo
Emulatore	• economico • flessibile • codice reale • ragionabilmente accurato • veloce e interattivo	• più lento dell'hardware • possibile inaccuratezza nel multiplexing

Infatti, comparando ai sistemi basati su virtualizzazione completa:
- Si avvia più velocemente;
- Ha una scala maggiore, considerando che è in grado di gestire centinaia di nodi invece che quantità inferiori alla decina;
- Maggiore banda concessa;
- Semplice installazione: è possibile copiare la cartella da una repository online e lanciare con i permessi di amministratore l'installazione del pacchetto (Linux) o installare la macchina virtuale;

Comparando agli hardware testbed Mininet risulta:
- molto meno costoso;
- velocemente riconfigurabile e re-installabile;
- facilmente trasportabile e condivisibile;

Se si paragona ad un simulatore di rete:
- esegue codice reale e non modificato;
- si connette facilmente alle reti reali;
- offre una performance interattiva.

Uno dei punti di forza di Mininet è la velocità: per avviare una semplice network si impiegano solo pochi secondi quindi significa che il run-edit-debug loop può essere molto veloce. Si possono creare topologie a piacere, che assomiglino alla rete internet, piuttosto che una banca dati dove ogni host è in grado di eseguire programmi dell'ambiente Linux utilizzando comandi appositi come `cmd()` o `sendcmd()`.
Grazie alla sua portabilità, è facilmente condivisibile con altri grazie al fatto che per lanciare una data rete con una data configurazione è necessario solo trasportare il file contenente il listato Python della topologia sul calcolatore destinazione, che può essere un laptop, server o macchina virtuale.
A fronte di questi lati positivi, ci sono alcuni aspetti negativi. Mininet usa un unico kernel Linux per tutti gli host virtuali; questo significa che non è possibile lanciare software che dipendono da BSD, Windows, o altri nuclei di sitemi operativi. Un altro aspetto è che differentemente da un simulatore, Mininet non ha una forte nozione di tempo virtuale: questo significa che le misurazioni temporali saranno basate sul tempo reale e che i risultati più veloci del tempo reale (e.g. 100 Gbps networks) non possono essere emulati facilmente. Questi sono casi estremi perché solitamente accade proprio il contrario: la rete virtuale è solitamente più veloce di un'equivalente reale.

3.4 Aggiungere hardware alla rete Mininet

Una volta creata la rete virtuale potrebbe risultare interessante far comunicare il mondo esterno con la network Mininet. In questo modo, sarebbe possibile creare con poche macchine Linux delle reti smisurate, collegando le varie virtual network tra loro.

Per poter connettere la rete virtuale a quella fisica è necessario un tramite che può essere lo switch OVS, al quale deve essere aggiunta la porta fisica al quale è collegato l'esterno. Questo può essere fatto in vari modi, ognuno con vantaggi e svantaggi a seconda dell'applicazione.

AGGIUNGERE LA PORTA FISICA ALLO SWITCH TRAMITE SCRIPT PYTHON

Questo primo metodo è legato alle potenzialità delle API Python, che permettono di creare topologie personalizzate, creando dei listati che verranno lanciati tramite linea di comando. Tra gli esempi che vengono scritti nel sistema operativo durante l'installazione di Mininet, ce n'è uno che implementa esattamente questa funzione. Per studiare il funzionamento di questo script, si deve andare nella cartella `examples` e scegliere il file `hwintf.py`:

```
# nano /home/mininet/mininet/examples/hwintf.py
```

Una volta aperto quello che viene stampato a schermo è il listato dello script. Si passerà ora ad analizzare ogni blocco di istruzioni, fondamentali per capirne il funzionamento.

```
import re
import sys
```

Questo primo blocco di istruzioni serve per importare i moduli che vengono usati dallo script. In particolare, il modulo `re` implementa le espressioni regolari Python, ovvero funzioni che permettono di cercare una corrispondenza tra due stringhe "pattern matching".
Invece, il modulo `sys` fornisce l'accesso ad alcune variabili usate o mantenute dall'interprete e a funzioni che interagiscono fortemente con l'interprete stesso. È sempre disponibile.

```
from mininet.CLI import CLI
from mininet.log import setLogLevel, info, error
from mininet.net import Mininet
from mininet.link import Intf
from mininet.topolib import TreeTopo
from mininet.util import quietRun
```

In questo secondo blocco, comando `from-import` vengono importate dalle classi Mininet le funzioni e le APIs che verranno usate nel listato. In questo caso viene attivata la CLI, viene scelta la modalità di Log per eventuali comunicazioni che verranno restituite dallo script e viene importata la topologia ad albero.

```
def checkIntf( intf ):
        "Make sure intf exists and is not configured."
        if ( ' %s:' %intf ) not in quietRun( 'ip link show' ):
                error( 'Error:', intf, 'does not exists!\n' )
                exit( 1 )
```

Si definisce la funzione `checkIntf` per controllare l'esistenza e lo stato dell'interfaccia fisica da aggiungere. Questa funzione consiste in un `if` dove ci si chiede se `%intf` non sia nella lista generata da `ip link show`. In caso affermativo, viene stampato un output dove viene confermato che l'interfaccia non esiste. Quando si verifica che l'interfaccia esiste, il programma prosegue.

```
def checkIntf( intf ):
        ips = re.findall( r'\d+\.\d+\.\d+\.\d+', quietRun( 'ifconfig' + intf ) )
        if ips:
                error( 'Error:', intf, 'has an IP address and is probably in use!\n'  )
                exit( 1 )
```

Successivamente, con le espressioni regolari si cerca se l'interfaccia ha un indirizzo IP con `re.findall` mettendo nella stringa `'\d+\.\d+\.\d+\.\d+'` dove `\d` significa decimale, il `+` concatenazione e il punto è la separazione dei 4 byte di IP, confrontando con la stringa che restituisce `quietRun( 'ifconfig' + interfaccia)`. Se c'è corrispondenza, allora l'interfaccia ha un IP, viene restituito un errore e lo script si chiude.

```
if __name__ == '__main__':
        setLogLevel( 'info' )
#try to get hw intf from te command line; by default, use eth1
                intfName = sys.argv[ 1 ] if len( sys.argv )  > 1 else 'eth1'
                info( '*** Connecting to hw intf: %s' %intfName )
```

Da questo punto in poi la finisce la definizione delle funzioni e inizia lo script vero e proprio. Lo si avvia con `if__name__== '__main__'` e si setta il livello di Log che si vuole.
Il modulo `sys` fornisce l'accesso ad alcune variabili usate o mantenute dall'interprete e a funzioni che interagiscono fortemente con l'interprete stesso. La funzione `sys.argv` fornisce una lista degli argomenti della riga di comando passati a uno script Python. In particolare `sys.argv[ 1 ]` copia il valore che gli viene fornito appena dopo il nome dello script: in questo caso scrivendo da linea di comando `hwintf.py eth2` verrebbe copiato `eth2`! Questo blocco, quindi, ci permette di scegliere quale interfaccia utilizzare e nel caso non venga fornito un interfaccia valida viene utilizzata di default la `eth1`.

```
                info( '*** Checking', intfName, '\n' )
                checkIntf( intfName )

                info( '*** Checking', network \n' )
                net = Mininet( topo=TreeTopo( depth=1, fanout=2 ) )
```

Viene stampato a schermo che lo script si sta connettendo all'interfaccia scelta.
Si richiama la funzione `checkIntf` definita in precedenza e si mette nel suo argomento il nome dell'interfaccia scelta; poi crea la network con una topologia ad albero.

```
                switch = net.switches[ 0 ]
                info( '*** Adding hardware interface', intfName, 'to switch', switch.name, '\n'
)
                _intf = Intf( intfName, node=switch )
                net.start()
                CLI( net )
                net.stop()
```

Questo blocco è la parte importante di questo script. È qui che viene aggiunta l'interfaccia allo switch.
Infine si avvia tutto con i comandi standard di mininet, selezionando il file.

AGGIUNGERE LA PORTA FISICA AD UN HOST TRAMITE SCRIPT PYTHON

Prendendo ad esempio lo script precedente, si potrebbe aggiungere l'interfaccia ad un host cambiando il penultimo blocco di listato come segue:

```
                host = net.hosts[ 0 ]
                info( '*** Adding hardware interface', intfName, 'to host', host.name, '\n' )
                _intf = Intf( intfName, node=host )
                net.start()
                CLI( net )
                net.stop()
```

Questi due metodi per aggiungere l'interfaccia fisica alla rete Mininet sono molto utili se non si intende lavorare troppo nella Command-Line Interface e usare due terminali, poiché una volta che lo script è pronto, per lanciare la network, dopo essersi messi nella cartella dov'è presente il file, è necessario usare il comando:

```
# python hwintf.py
```

La rete che verrà avviata sarà già pronta con la porta fisica tra le porte dello switch virtuale.

AGGIUNGERE LA PORTA FISICA ALLO SWITCH TRAMITE COMANDO OVS-VSCTL

Con questo metodo è necessario utilizzare due terminali. Questo è dovuto al fatto che l'interprete Mininet non permette di aggiungere porte allo switch perché nel caso venisse scritta la stringa "s1", verrebbe sostituita con "127.0.0.1", ovvero l'indirizzo di loopback. Per questo è utile sapere che per passare da un terminale all'altro è necessario usare la combinazione di tasti CTRL+ALT+Fn, dove n corrisponde all'n-esimo terminale che viene scelto. Verranno qui descritte le operazioni in ciascun terminale.

Terminale Mininet
In questo terminale viene avviato Mininet col comando

```
# mn --switch=ovs
```

dove con `--switch=ovs` si dichiara all'emulatore di lanciare un OpenVSwitch.

Terminale senza Mininet
Utilizzando, ad esempio, la combinazione CTRL+ALT+F2 si passa ad un altro terminale. In questo terminale si deve lavorare sullo switch per fare in modo che venga aggiunta la porta fisica alla rete virtuale. Per fare questo, col comando `ovs-vsctl`, si controlla quali switch virtuali sono attivi:

```
# ovs-vsctl list-br
s1
```

Si vede che l'unico switch OVS presente è s1, creato da mininet. Si controlla quali porte sono configurate:

```
# ovs-vsctl list-ports s1
s1-eth1
s1-eth2
```

Come si vede, ci sono solo le interfacce virtuali che crea mininet per fare il collegamento tra lo switch e gli hosts. Tramite `ovs-vsctl` è possibile aggiungere la porta fisica chiamata, ad esempio, eth0:

```
# ovs-vsctl add-port s1 eth0
```

Per controprova, si verifica che la porta si stata aggiunta alla ports-list dello switch:

```
# ovs-vsctl list-ports s1
eth0
s1-eth1
s1-eth2
```

A questo punto, per stabilire la connessione, l'interfaccia eth0 deve essere priva di indirizzo IP che dovrà essere assegnato come indirizzo secondario alla card "s1" rappresentante lo switch.

È chiaro che questa soluzione sia quella un po' più laboriosa in termini di istruzioni fornite alla linea di comando per ottenere la rete virtuale-fisica ma è anche vero che per creare uno script Python, in base alla sua complessità, può richiedere ancora un maggior tempo.

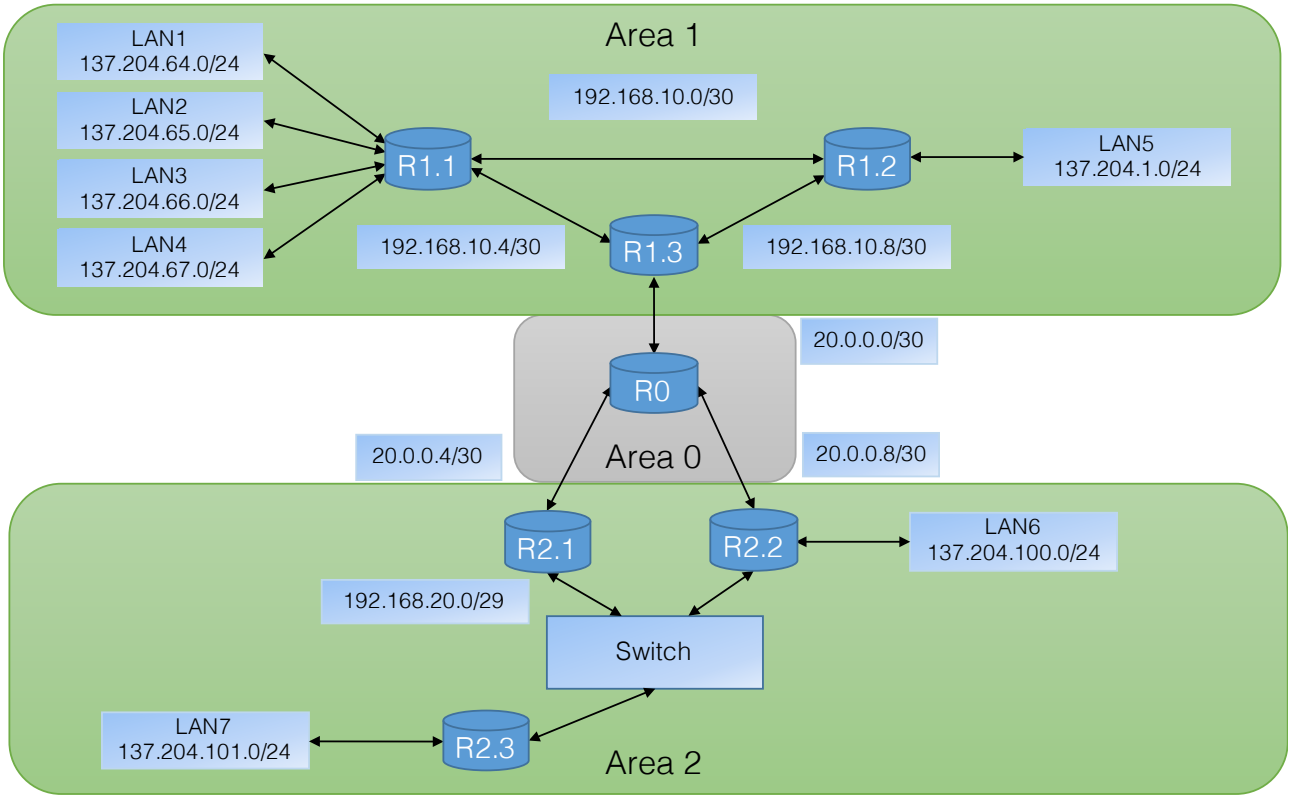
Nel seguito verrà utilizzata quest'ultima modalità di aggiunta dell'interfaccia fisica poiché è più veloce apportare modifiche alla topologia data la maggiore flessibilità della linea di comando rispetto allo script.

4. Sperimentazione di routing IP su Raspberry Pi

4.1 Descrizione della topologia

La topologia che si intende creare è costituita da tre aree formate da vari nodi di routing e svariate LAN fisiche e virtuali. In particolare, l'area 1 è composta da tre router implementati da Raspberry dove R1.1 avrà quattro segmenti LAN virtuali, R1.2 avrà un segmento LAN virtuale mentre R1.3 farà semplicemente da router IP e sarà il collegamento tra l'area 1 e l'area 0. Nell'area 0 ci sarà il router R0 che sarà la backbone della rete, al quale si collegheranno i router R2.1 e R2.2 dell'area 2. Quest'ultimi saranno connessi ad uno switch virtuale creato sull'ultimo Raspberry R2.3, all'interno del quale, oltre allo switch, sarà virtualizzata anche una LAN.

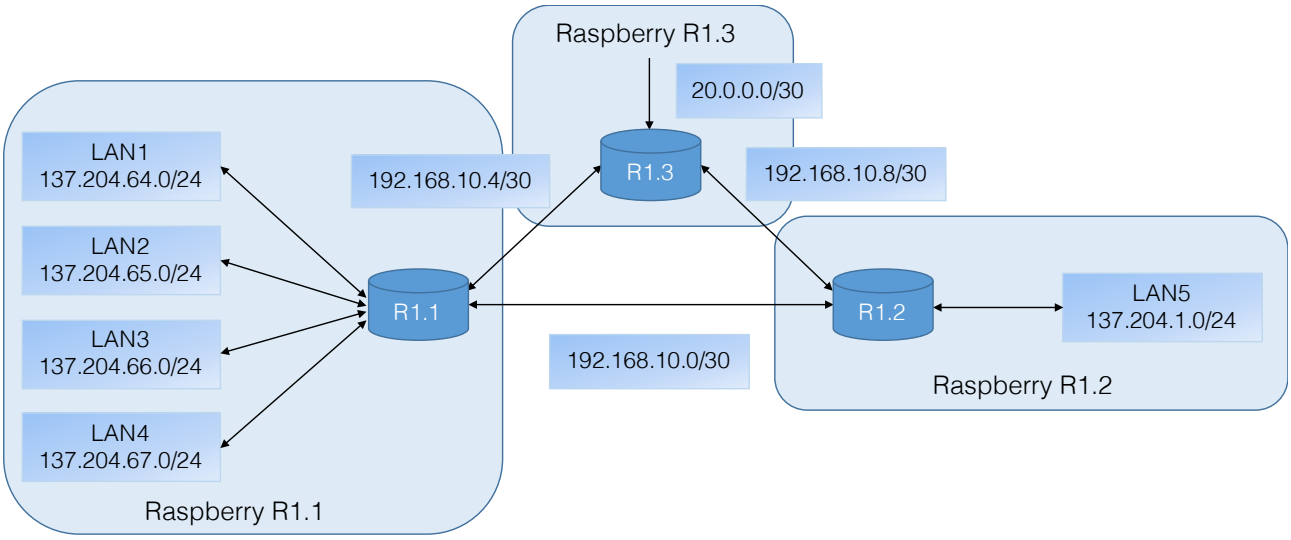
Lo schema rappresentativo è quello che segue:



Lo scopo è quello di cercare di riprodurre una rete Internet miniaturizzata per studiare nella pratica come si comportano i pacchetti che viaggiano tra i nodi.

4.2 Creazione della topologia: Area 1

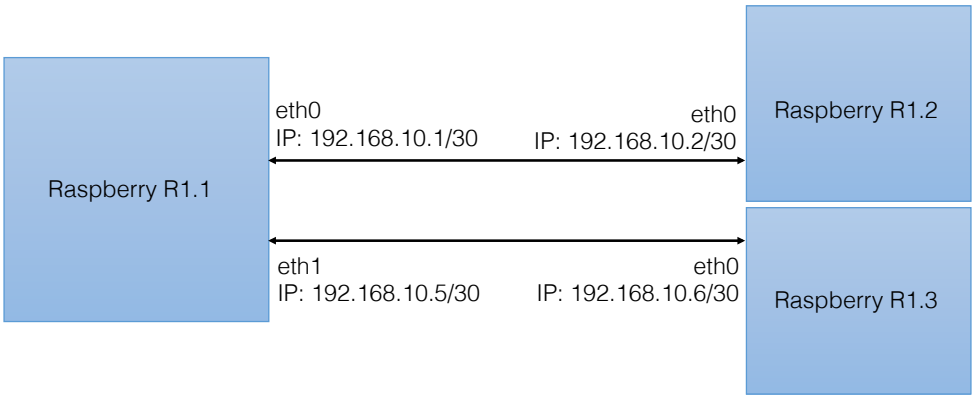
La creazione di quest'area consiste semplicemente nel settare nel modo giusto i tre dispositivi Raspberry Pi, non facendo confusione tra le reti.



CONFIGURAZIONE RASPBERRY PI R1.1

Questo nodo dovrà virtualizzare le quattro LAN e occuparsi dell'instradamento dei pacchetti che gli vengono inviati in modo da poter far comunicare un host virtuale con gli altri nodi e le altre LAN. La cosa primaria da fare è quella di assegnare alle interfacce Ethernet gli indirizzi IP ed i rispettivi gateway per verificare che i nodi possano comunicare tra loro. Si useranno in seguito:

- 192.168.10.1/30 per la eth0 con default gateway 192.168.10.2 (R1.2);
- 192.168.10.5/30 per la eth1 con default gateway 192.168.10.6 (R1.3);



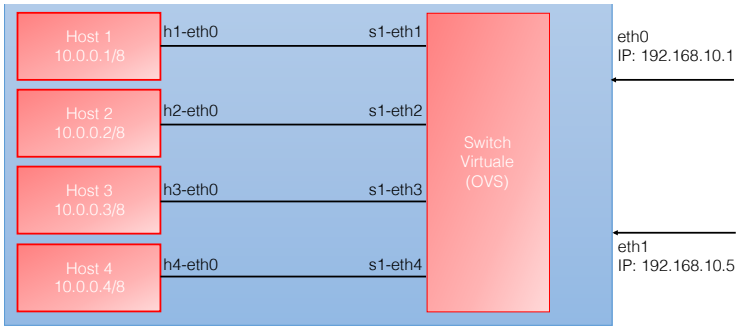
È possibile eseguire questa operazione assegnando degli indirizzi temporanei (comando ifconfig - ip addr) o indirizzi statici (modificando il file /etc/network/interfaces). Scegliendo di utilizzare ifconfig si dovrà digitare:

```
# ifconfig eth0 192.168.10.1/30
# ifconfig eth1 192.168.10.5/30
```

Se col comando ping si ottiene un dialogo con entrambi i gateway, si può procedere alla creazione delle LAN virtuali utilizzando Mininet. Si sceglierà di utilizzare un OpenVSwitch e di creare una topologia con un singolo switch al quale si collegano i quattro segmenti di rete. Tutto questo è ottenibile con:

```
# mn --switch=ovs —topo=single,4
```

La rete ottenuta corrisponde alla figura sotto, dove è stato evidenziato in rosso il carattere virtuale degli hosts e dello switch e in blu la componente hardware, rappresentata dal Raspberry Pi R1.1. Nella figura sono anche evidenziati gli indirizzi IP impostati su ciascun host e si nota che appartengono tutti alla stessa rete 10.0.0.0/8.



Per modificarne gli indirizzi degli hosts, in modo tale che si vengano a creare quattro reti distinte, esistono tre modalità: la prima consiste nell'utilizzare la Commando-Line Interface messa a disposizione da Mininet, lavorando su ciascun host come se fosse una macchina con sistema operativo Linux indipendente; la seconda sfrutta le API Python richiamate col comando `py`; la terza si basa sulla creazione di una topologia custom con indirizzi preimpostati. Il secondo e terzo aggiornano l'interprete Mininet, che sostituirà, ad esempio, ad h1 la stringa "137.204.64.1" e permetterà di verificare la connessione tra tutti gli hosts semplicemente col comando `pingall`. Le istruzione da eseguire nella CLI saranno:

```
mininet>py h1.setIP('137.204.64.1/24')
mininet>py h2.setIP('137.204.65.1/24')
mininet>py h3.setIP('137.204.66.1/24')
mininet>py h4.setIP('137.204.67.1/24')
```

Ovviamente, in questa topologia è fondamentale che i vari segmenti di rete comunichino tra loro. È quindi necessario aggiungere ad ogni host un IP di default gateway che verrà poi assegnato allo switch virtuale. Per convenzione si sceglie di usare come gateway l'indirizzo .254, ovvero l'ultimo disponibile prima dell'indirizzo broadcast. Su ogni host va digitato:

```
mininet>h1 route add default gw 137.204.64.254
mininet>h2 route add default gw 137.204.65.254
mininet>h3 route add default gw 137.204.66.254
mininet>h4 route add default gw 137.204.67.254
```

Successivamente, si deve agire sullo switch. È necessario aggiungere le interfacce fisiche eth0 e eth1 allo switch virtuale come detto in precedenza. Il metodo migliore per questa topologia, se non si è creato uno script specifico, è quello che prevede l'uso della classe di comandi `ovs-vsctl` passando ad un altro terminale.

Terminale senza Mininet

```
# ovs-vsctl add-port s1 eth0
# ovs-vsctl add-port s1 eth1
    # ifconfig eth0 0.0.0.0
    # ifconfig eth1 0.0.0.0
```

Si provvede a togliere l'indirizzo IP alle porte eth0 e eth1 e assegnare allo switch sia gli indirizzi delle interfacce fisiche sia gli indirizzi di gateway di ciascun host e si deve abilitare il forwarding.

```
# ip addr add 137.204.64.254/24 dev s1
# ip addr add 137.204.65.254/24 dev s1
# ip addr add 137.204.66.254/24 dev s1
# ip addr add 137.204.67.254/24 dev s1
  # ip addr add 192.168.10.1/30 dev s1
  # ip addr add 192.168.10.5/30 dev s1
     # sysctl -w net.ipv4.ip_forward=1
```

Ora la connessione è stabilita. Per verificare che tutti gli hosts comunichino tra loro è possibile utilizzare il comando ping da ciascun host provando tutte le combinazioni. Però, se si è usato il comando `py` di Mininet per assegnare gli IP, è possibile usare il comando `pingall` che dovrà dare un risultato del tipo:

```
mininet>pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

Questo risultato può essere ottenuto creando una topologia "custom" che crei gli host, gli switch, i collegamenti e che ne imposti gli indirizzi e gateway all'avvio. Per creare una rete virtuale come quella creata tramite script Python si può utilizzare il seguente listato:

```
from mininet.topo import Topo

class MyTopo( Topo ):
   “Topologia con uno switch collegato a quattro LAN."

   def __init__( self ):
      “Creazione topologia."

      # Inizializzazione topologia
            Topo.__init__( self )

      # Aggiunta di host e switch
       Host1 = self.addHost( 'h1', ip='137.204.64.1/24', defaultRoute='via
137.204.64.254' )

      Host2 = self.addHost( 'h2', ip='137.204.65.1/24', defaultRoute='via 137.204.65.254' )

      Host3 = self.addHost( 'h3', ip='137.204.66.1/24', defaultRouter='via
137.204.66.254' )

      Host4 = self.addHost( 'h4', ip='137.204.67.1/24', defaultRouter='via
137.204.67.254' )

      Switch1 = self.addSwitch( 's1' )

      # Aggiunta collegamenti
      self.addLink( Host1, Switch1 )
      self.addLink( Host2, Switch1 )
      self.addLink( Host3, Switch1 )
      self.addLink( Host4, Switch1 )


topos = { 'topo1': ( lambda: MyTopo() ) }
```

Per richiamare la topologia custom bisogna indicare il percorso del file dopo l’opzione `–custom` e dopo l’opzione `–topo` andrà inserito il nome che si è scelto per la topologia, che può essere modificato nell’ultima linea del listato.

```
# mn —custom /home/pi/topo1.py —topo=topo1
```

Nel seguito della tesi verrà utilizzato l’approccio precedente, per rendere più chiara l’idea di come è impostata rete.

Essendo una rete composta da molti nodi, è necessario sfruttare degli algoritmi di routing adatti. Per questo si andrà a sollecitare il pacchetto quagga: in particolare si utilizzeranno i protocolli zebra e ospfd.

La configurazione deve partire dalla modifica del file daemons in `/etc/quagga`, andando ad attivare il protocollo necessario sostituendo la stringa “yes” come segue:

```
# nano /etc/quagga/daemons
zebra=yes
bgpd=no
ospfd=yes
ospf6d=no
ripd=no
ripngd=no
isisd=no
```

Il passo successivo sarà quello di aggiungere nella stessa cartella i file di configurazione per ciascuno protocollo. Si noti che in questo modo le configurazioni saranno permanenti e persisteranno anche al riavvio del nodo di routing. I file possono essere copiati dagli esempi messi a disposizione dalla documentazione:

```
# cp /usr/share/doc/quagga/examples/zebra.conf.sample  /etc/quagga/zebra.conf
# cp /usr/share/doc/quagga/examples/ospfd.conf.sample  /etc/quagga/ospfd.conf
 # cp /usr/share/doc/quagga/examples/vtysh.conf.sample  /etc/quagga/vtysh.conf
```

In seguito, si andrà a modificare il file zebra.conf per adattarlo alle esigenze della rete da implementare. Il processo di configurazione di zebra e ospfd deve essere fatto su tutti i router presenti nella network, in modo che i nodi possano comunicare tra loro tramite il pacchetto “hello” del protocollo OSPF. Impostando il nodo R1.1 si ottiene un file come segue:

```
! -*- zebra -*-

hostname Router11
password zebra
enable password zebra

interface lo
description loopback
ip address 127.0.0.1/8
ip forwarding

interface s1
ip address 192.168.10.1/30
ip address 192.168.10.5/30
ip address 137.204.64.254/24
ip address 137.204.65.254/24
ip address 137.204.66.254/24
ip address 137.204.67.254/24
ip forwarding

log file /var/log/quagga/zebra.log
```

dove si sono impostati gli indirizzi che deve avere s1, l'indirizzo di loopback e la stringa "ip forwarding" per l'inoltro dei pacchetti.

Si passa ora ad esaminare il file ospfd.conf dove si andrà ad indicare la porzione di rete raggiungibile direttamente dal nodo attraverso le sue interfacce. Si otterrà un file di questo tipo:

```
! -*- ospf -*-

hostname Router11
password zebra

interface s1

router ospf
network 192.168.10.0/30 area 1
network 192.168.10.4/30 area 1
network 137.204.64.0/24 area 1
network 137.204.65.0/24 area 1
network 137.204.66.0/24 area 1
network 137.204.67.0/24 area 1

log stdout
log file /var/log/quagga/ospfd.log
```

In questo file vanno indicate le interfacce che sono coinvolte nel processo di forwarding, in questo caso solo s1, e le reti al quale il nodo è in grado di effettuare la consegna diretta dei pacchetti.

Quando le modifiche sono state ultimate, per renderle effettive, è necessario riavviare il servizio quagga digitando da terminale:

```
# /etc/init.d/quagga restart
```

Una volta completati questi passaggi, sarà possibile lanciare la CLI di `vtysh` che permette di controllare gli effetti delle impostazioni fatte. Una volta completato questo processo su tutti i router della rete, passando alla CLI il comando `show ip route` si ottengono tutte le reti raggiungibili direttamente e quelle raggiungibili attraverso altri nodi. Il risultato dovrebbe essere simile a quanto segue:

```
raspberrypi# show ip route

Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

C>* 127.0.0.0/8 is directly connected, lo
O   137.204.64.0/24 [110/10] is directly connected, s1, 00:00:17
C>* 137.204.64.0/24 is directly connected, s1
O   137.204.65.0/24 [110/10] is directly connected, s1, 00:00:17
C>* 137.204.65.0/24 is directly connected, s1
O   137.204.66.0/24 [110/10] is directly connected, s1, 00:00:17
C>* 137.204.66.0/24 is directly connected, s1
O   137.204.67.0/24 [110/10] is directly connected, s1, 00:00:17
C>* 137.204.67.0/24 is directly connected, s1
O   192.168.10.0/30 [110/10] is directly connected, s1, 00:00:17
C>* 192.168.10.0/30 is directly connected, s1
O   192.168.10.4/30 [110/10] is directly connected, s1, 00:00:17
C>* 192.168.10.4/30 is directly connected, s1
```

È facilmente distinguibile ogni network collegata al nodo e all'interfaccia s1. La simbologia indica con la lettera C la route direttamente connessa al nodo e con la lettera O che indica che quella è la strada generata dal router tramite i protocolli di routing per consegna indiretta, come sarà visto nel nodo R0. Quando saranno configurati anche gli altri nodi, sarà possibile testare la comunicazione tra ogni host e l'esterno, semplicemente utilizzando il comando `ping` verso tutti gli indirizzi presenti nella network.

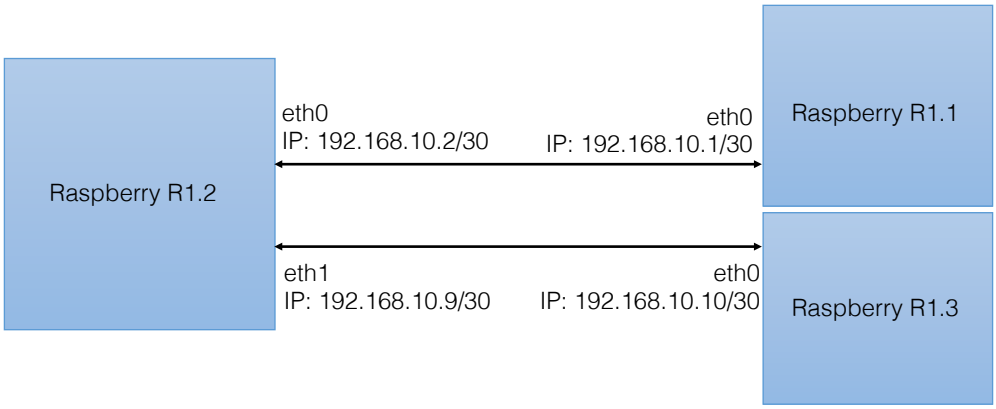
Il servizio quagga impostato come spiegato per il router R1.1, permette l'assegnazione statica degli indirizzi IP e dei gateway. Di conseguenza è possibile effettuare l'assegnazione solo attraverso i file di configurazione senza utilizzare comandi `ifconfig` e `ip addr`, come verrà fatto dal router R1.3 in poi.

CONFIGURAZIONE RASPBERRY PI R1.2

Questo nodo si dovrà comportare in modo simile al nodo R1.1, con la piccola differenza che le LAN virtuali non saranno quattro ma una sola con indirizzo 137.204.1.0/24 e anche questo nodo dovrà eseguire il forwarding, comportandosi da router.

In questo nodo gli indirizzi delle interfacce saranno:

- 192.168.10.2/30 per la eth0 con default gateway 192.168.10.1 (R1.1);
- 192.168.10.9/30 per la eth1 con default gateway 192.168.10.10 (R1.3)



Per creare la LAN virtuale si utilizza Mininet con la topologia a singolo switch al quale verrà aggiunto un solo host:

```
# mn --switch=ovs —topo=single,1
```

Una volta avviato si assegnano allo host h1 l'indirizzo IP 137.204.1.1/24 e il default gateway 137.204.1.254 che verrà assegnato allo switch virtuale come indirizzo secondario.

```
mininet>py h1.setIP('137.204.1.1/24')
mininet>h1 route add default gw 137.204.1.254
```

Si aggiungono ora le porte fisiche e gli indirizzi IP allo switch e si attiva il forwarding passando ad un terminale senza Mininet.

Terminale senza Mininet

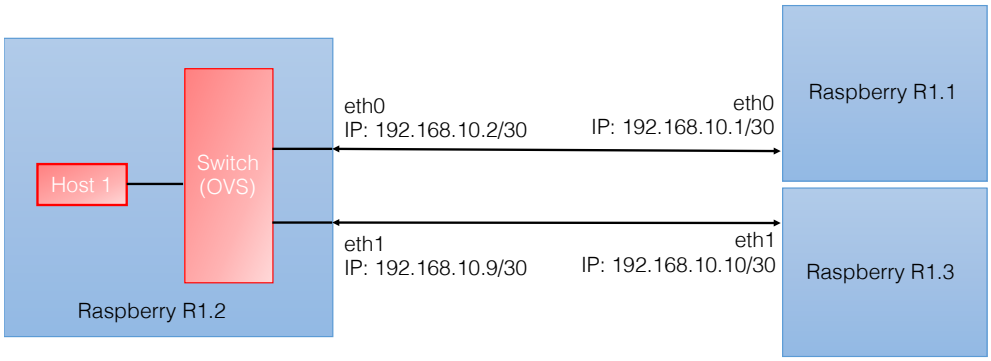
```
# ovs-vsctl add-port s1 eth0
# ovs-vsctl add-port s1 eth1
# ifconfig eth0 0.0.0.0
# ifconfig eth1 0.0.0.0
# ip addr add 137.204.1.254/24 dev s1
# ip addr add 192.168.10.2/30 dev s1
# ip addr add 192.168.10.9/30 dev s1
# sysctl -w net.ipv4.ip_forward=1
```

Si va poi ad impostare i file di configurazione zebra e ospfd considerando che le interfacce a cui sono collegati i cavi Ethernet sono associate a s1.

```
! -*- zebra -*-

hostname Router12
password zebra
enable password zebra

interface lo
description loopback
ip address 127.0.0.1/8
ip forwarding

interface s1
ip address 192.168.10.2/30
ip address 192.168.10.9/30
ip address 137.204.1.254/24
ip forwarding

log file /var/log/quagga/zebra.log
```

```
! -*- ospf -*-

hostname Router12
password zebra

interface s1

router ospf
network 192.168.10.0/30 area 1
network 192.168.10.8/30 area 1
network 137.204.1.0/24 area 1

log stdout
log file /var/log/quagga/ospfd.log
```

Effettuate le impostazione si riavvii il servizio quagga e con `vtysh` si può controllare il risultato, controllando anche quali reti vengono aggiunte tramite il router R1.1. Il risultato ottenuto è indicato in figura.



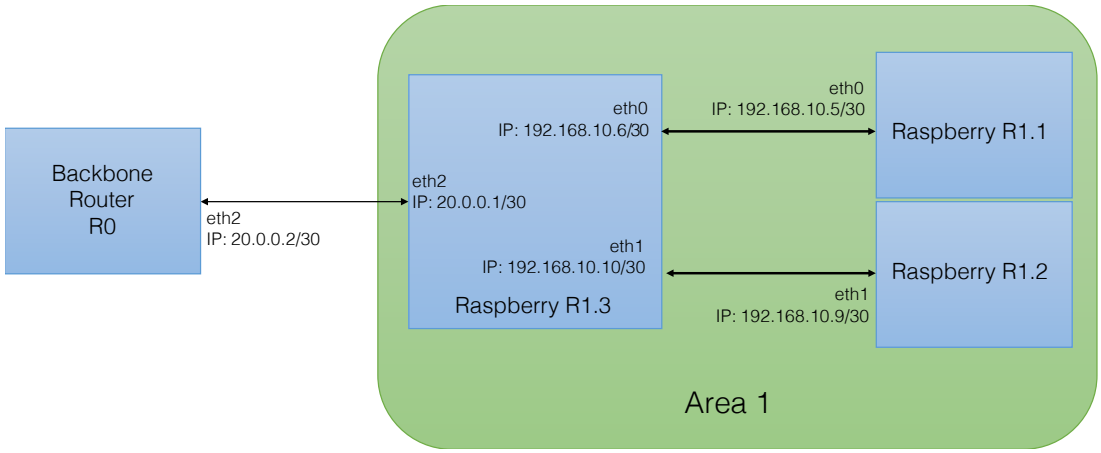
È interessante ora provare a far comunicare l'host virtuale nella LAN appena creata con uno di quelli virtuali creati nel nodo R1.1. Col comando `traceroute` è possibile vedere tutti gli hops che vengono "toccati" dal pacchetto inviato alla destinazione ed è un ottimo metodo per studiare il funzionamento dei collegamenti, ad esempio eseguendo il `traceroute` tra h3 di R1.1 e h1 di R1.2.

CONFIGURAZIONE RASPBERRY PI R1.3

A differenza degli altri dispositivi, in questo nodo non è necessario creare una rete virtuale quindi il setting si basa semplicemente sull'impostazione degli IP delle interfacce e sull'attivazione del forwarding. Va anche assegnato l'indirizzo ad una terza interfaccia che collega l'Area 1 al backbone router che a sua volta si collegherà anche all'Area 2. Gli IP adottati saranno:

- 192.168.10.6/30 per connettersi al router R1.1;
- 192.168.10.10/30 per connettersi al router R1.2;
- 20.0.0.1/30 per connettere R1.3 a R0.

In questo router si utilizza direttamente l'assegnazione delle impostazione IP tramite quagga. Si creano i file configurazione e li si impostano considerando che, a differenza dei casi precedenti, le interfacce da impostare saranno eth0, eth1 e eth2. Inoltre, modificando l'ospfd si deve segnalare che eth2 è connesso ad un nodo nell'area di backbone che verrà indicata come area 0.



L'area di backbone è letteralmente "la colonna vertebrale" della rete poiché ogni pacchetto che l'area 1 vuole scambiare con l'area 2 deve passare per forza attraverso l'area di backbone e viceversa.

```
! -*- zebra -*-

hostname Router13
password zebra
enable password zebra

interface lo
description loopback
ip address 127.0.0.1/8
ip forwarding

interface eth0
description to_Router11
ip address 192.168.10.6/30
ip forwarding

interface eth1
description to_Router12
ip address 192.168.10.10/30
ip forwarding

interface eth2
description to_Router0
ip address 20.0.0.1/30
ip forwarding

log file /var/log/quagga/zebra.log
```

```
! -*- ospf -*-

hostname Router13
password zebra

interface eth0
interface eth1
interface eth2

router ospf
network 192.168.10.4/30 area 1
network 192.168.10.8/30 area 1
network 20.0.0.0/30 area 0

log stdout
log file /var/log/quagga/ospfd.log
```

NOTE

DHCP

Il DHCP (Dynamic Host Configuration Protocol) è un protocollo di rete che permette ai dispositivi collegati ad una rete locale di ottenere automaticamente una configurazione corretta per poter comunicare tramite protocollo Internet. Questo servizio assegna un indirizzo IP tra quelli ancora disponibili in rete e imposta automaticamente DNS e gateway.
Quando si crea la rete virtuale con Mininet e il processo che fornisce il servizio DHCP è attivo, è possibile che vengano assegnati automaticamente degli indirizzi IP alle interfacce virtuali dello switch (ad esempio s1-eth2), che rischiano di far saltare la comunicazione. Per evitare che questo accada, è possibile disattivare sin dall'avvio il processo togliendo i permessi di esecuzione al file `dhcpcd`:

```
# chmod -x /etc/init.d/dhcpcd
```

dove il comando `chmod` modifica i permessi e il flag `-x` toglie l' "eseguibilità" a `dhcpcd`.

Impostare indirizzi IP statici

Se si intende utilizzare i dispositivi "headless", ovvero facendoli lavorare in modo automatico, è possibile utilizzare la configurazione statica degli indirizzi di rete modificando il file testuale `/etc/network/interfaces` . È sufficiente modificare la stringa "dhcp" in "static" e mettere indirizzo, netmask e gateway. Se però si lavora su una rete di grandi dimensioni su cui sono attivi i protocollo di routing quagga, è possibile configurare solo i file di configurazione quagga per assegnare anche gli indirizzi ogni volta che il processo viene lanciato (come all'avvio).

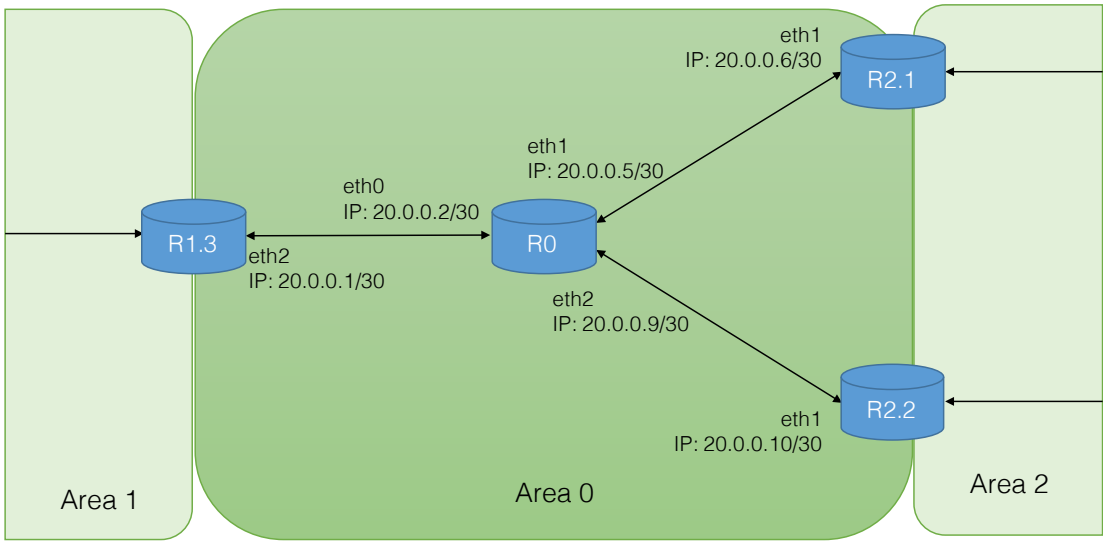
Impostare attivazione automatica del forwarding all'avvio

Grazie al file `rc.local` messo a disposizione da Debian, è possibile far eseguire al dispositivo dei comandi in automatico. L'esecuzione dei comandi avviene appena dopo che il sistema ha effettuato il boot di tutte le sue componenti.
Per evitare di ripetere il comando di forwarding ogni volta che si vuole simulare la rete, si va a modificare il file `/etc/rc.local` inserendo il comando `sysctl -w net.ipv4.ip_forward=1` nel testo senza andare a modificare la stringa exit(0) o creare dei loop infiniti, altrimenti non si avvierebbe il dispositivo.

4.3 Creazione della topologia: Area 0

Quest'area si occupa della connessione tra l'area 1 e l'area 2. Essa comprende un router centro-stella collegato alle interfacce dei border routers che appartengono alle aree adiacenti.

Quest'area, in virtù della sua funzione di giunzione tra le due aree, è detta backbone area. La sua primaria funzione è quella di indirizzare il traffico attraverso gli altri sistemi autonomi.



CONFIGURAZIONE RASPBERRY PI R0

Considerato che questo router non deve virtualizzare nessuna rete, ma si occupa solo della funzione di routing, si passa al settaggio del protocollo quagga. Gli indirizzi utilizzati per questo nodo sono:

- 20.0.0.2/30 che connette R0 al router R1.3;
- 20.0.0.5/30 per connettersi al router dell'area 2 R2.1;
- 20.0.0.9/30 per connettersi al router dell'area 2 R2.2.

La procedura non cambia. Si creano i file configurazione e si sentano i file come mostrato nella pagina seguente.
Dopo aver impostato già qualche nodo è più interessante andare a vedere la tabella di routing col comando `show ip route` dalla shell vtysh. Verranno riportate solo le linee più interessanti, ovvero quelle della consegna indiretta:

O>* 192.168.10.0/30 [110/30] via 20.0.0.1, eth0
O>* 192.168.10.4/30 [110/30] via 20.0.0.1, eth0
O>* 192.168.10.8/30 [110/30] via 20.0.0.1, eth0

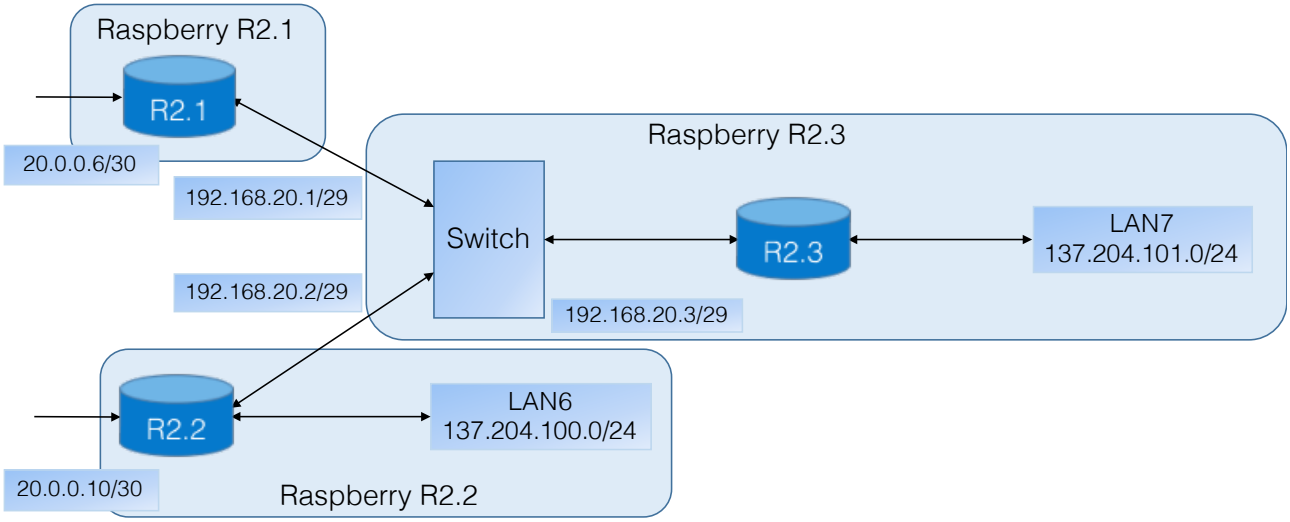
Si vede che le 3 reti 192.168.10 sono raggiunte dai pacchetti inviati da R0 tramite l'indirizzo IP 20.0.0.1 che corrisponde esattamente al router R1.3, come ci si aspettava. Inoltre si distingue la lettera O, che indica il protocollo OSPF, e il simbolo >, che sta ad indicare che quella è la route scelta dal router per contattare la rete.

```
! -*- zebra -*-

hostname Router0
password zebra
enable password zebra

interface lo
description loopback
ip address 127.0.0.1/8
ip forwarding

interface eth0
description to_Router13
ip address 20.0.0.2/30
ip forwarding

interface eth1
description to_Router21
ip address 20.0.0.5/30
ip forwarding

interface eth2
description to_Router22
ip address 20.0.0.9/30
ip forwarding

log file /var/log/quagga/zebra.log
```

```
! -*- ospf -*-

hostname Router0
password zebra

interface eth0
interface eth1
interface eth2

router ospf
network 20.0.0.0/30 area 0
network 20.0.0.4/30 area 0
network 20.0.0.8/30 area 0

log stdout
log file /var/log/quagga/ospfd.log
```

4.4 Creazione della topologia: Area 2

L'area 2 si compone di tre router (R2.1, R2.2 e R2.3) che vanno a collegarsi ad uno switch. In tutto i dispositivi fisici saranno tre, poiché le LAN e lo switch saranno virtualizzate con Mininet.



CONFIGURAZIONE RASPBERRY PI R2.1

Il Raspberry R2.1 dovrà occuparsi del solo inoltro, essendo il border router dell'area 2 che va a collegarsi alla backbone e non essendo connesso a nessun host. Verranno utilizzati i seguenti IP:

- 192.168.20.1/29 per connettersi ai router dell'area 2 R2.2 e R2.3 attraverso eth0;
- 20.0.0.6/30 che connette R0 al router R2.1 tramite eth1;

È sufficiente andare a impostare il servizio quagga che automaticamente al suo avvio fornirà gli indirizzi IP alle interfacce fisiche del nodo. Si modificano i file di configurazione come si è fatto per tutti gli altri dispositivi.

```
! -*- zebra -*-

hostname Router21
password zebra
enable password zebra

interface lo
description loopback
ip address 127.0.0.1/8
ip forwarding

interface eth0
description to_s2
ip address 192.168.20.1/29
ip forwarding

interface eth1
description to_Router0
ip address 20.0.0.6/30
ip forwarding

log file /var/log/quagga/zebra.log
```

```
! -*- ospf -*-

hostname Router21
password zebra

interface eth0
interface eth1

router ospf
network 192.168.20.0/29area 2
network 20.0.0.4/30 area 2

log stdout
log file /var/log/quagga/ospfd.log
```

CONFIGURAZIONE RASPBERRY PI R2.2

Questo nodo è analogo al router R1.2 che ospita una LAN virtuale creata con Mininet ed è collegato ad altri due nodi. L'unica differenza sta nelle aree che va a toccare questo router, poiché è un router di bordo che intercorrente area 0 e area 2.

Gli indirizzi che si andranno ad usare sono:

- 192.168.20.2/29 che connette la rete interna al nodo col la parte di network connessa a R2.3 tramite eth0;
- 137.204.100.254/24 per fare da default gateway all'host virtuale nella LAN 137.204.100.0/24;
- 20.0.0.10/30 per collegare il nodo tramite l'interfaccia Ethernet eth1 a R0.

Questi indirizzi andranno assegnati all'interfaccia dello switch virtuale s1 dopo che sono stati tolti gli indirizzi alle porte fisiche.
È possibile utilizzare il solito comando Mininet per creare una LAN connessa ad un unico host al quale vengono assegnati IP e default gateway:

```
# mn --switch=ovs —topo=single,1
mininet> py h1.setIP('137.204.100.1/24')
mininet>h1 route add default gw 137.204.100.254
```

Col comando `ovs-vsctl` vengono aggiunte le interfacce fisiche come porte allo switch s1 per permettere la comunicazione con l'esterno degli host.

```
# ovs-vsctl add-port s1 eth0
# ovs-vsctl add-port s1 eth1
```

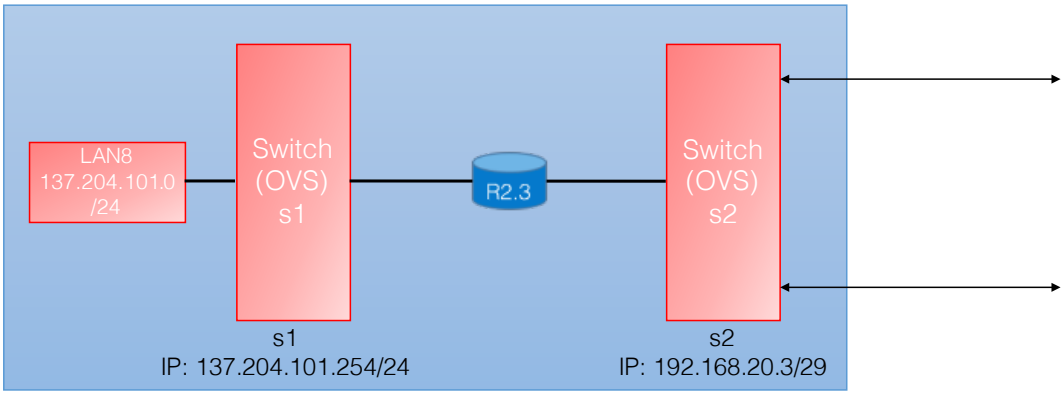
Di nuovo, si conclude la configurazione aggiungendo le impostazioni ai file quagga:

```
! -*- zebra -*-

hostname Router22
password zebra
enable password zebra

interface lo
description loopback
ip address 127.0.0.1/8
ip forwarding

interface s1
description LAN
ip address 137.204.100.254/24
ip address 20.0.0.10/30
ip address 192.168.20.2/29
ip forwarding

log file /var/log/quagga/zebra.log
```

```
! -*- ospf -*-

hostname Router22
password zebra

interface s1

router ospf
network 137.204.100.0/24 area 2
network 192.168.20.0/29 area 2
network 20.0.0.8/30

log stdout
log file /var/log/quagga/ospfd.log
```

CONFIGURAZIONE RASPBERRY PI R2.3

Il router R2.3 è un nodo che dovrà virtualizzare una rete non banale al suo interno. Saranno presenti due switch virtuali: uno sarà posto tra la LAN virtuale e il router (che è un nodo simbolico visto che la funzione sarà fatta dal kernel Linux), mentre l'altro sarà posto tra le interfacce fisiche e il router. Una visione più chiara si ha guardando l'immagine che si focalizza sulla rete all'interno del Raspberry R2.3.



Raspberry R2.3

Gli indirizzi che verranno utilizzati in questo nodo sono:

- 192.168.20.3/29 che connette la rete interna al nodo col resto della network esterna al router R2.3;
- 137.204.101.254/24 per fare da default gateway all'host virtuale nella LAN 137.204.101.0/24;

Per creare la rete virtuale si può cominciare partendo da Mininet, che in un unico passaggio permette di creare il primo switch virtuale s1 al quale sarà collegato l'host h1. Come nei casi di R1.2 e R2.2 si andrà a impostare manualmente l'indirizzo dell'host e il suo gateway.

```
# mn --switch=ovs —topo=single,1
mininet>py h1.setIP('137.204.101.1/24')
mininet>h1 route add default gw 137.204.101.254
```

Una volta configurati LAN e switch s1, si passa al secondo switch. Quest'ultimo dovrà essere dotato di indirizzo IP tale che permetta di comunicare coi due nodi esterni vicini: gli si assegnerà l'indirizzo 192.168.20.3/29.

Si sfruttano i comandi ovs-vsctl per creare lo switch s2 e per aggiungere alle sue porte le interfacce fisiche:

```
# ovs-vsctl add-br s2
# ovs-vsctl add-port s2 eth0
# ovs-vsctl add-port s2 eth1
```

Infine, come al solito, si dovranno impostare i protocolli di routing OSPF e zebra andando a scrivere nei file di configurazione che le addette al forwarding sono le interfacce virtuali s1 e s2.

```
! -*- zebra -*-

hostname Router23
password zebra
enable password zebra

interface lo
description loopback
ip address 127.0.0.1/8
ip forwarding

interface s1
description LAN
ip address 137.204.101.254/24
ip forwarding

interface s2
description to_Router21_and_Router22
ip address 192.168.20.3/29
ip forwarding

log file /var/log/quagga/zebra.log
```

```
! -*- ospf -*-

hostname Router23
password zebra

interface s1
interface s2

router ospf
network 137.204.101.0/24 area 2
network 192.168.20.0/29 area 2

log stdout
log file /var/log/quagga/ospfd.log
```

La rete è stata impostata correttamente. Adesso è possibile effettuare dei test per capire come funziona una rete simil-Internet che sfrutta i protocolli di routing.

4.5 Routing tables e test connettività

Nel seguito di questo sotto-capitolo verranno elencate le tabelle di routing a regime di ciascun router per verificare la consistenza della rete configurata nel corso di questo capitolo.
Router R1.1:

```
raspberrypi# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

O>*    20.0.0.0/30 [110/20] via 192.168.10.6, s1, 01:24:02
O>*    20.0.0.4/30 [110/30] via 192.168.10.6, s1, 01:24:02
O>*    20.0.0.8/30 [110/30] via 192.168.10.6, s1, 01:24:02
C>*    127.0.0.0/8 is directly connected, lo
O>*    137.204.1.0/24 [110/30] via 192.168.10.6, s1, 00:03:14
O      137.204.64.0/24 [110/10] is directly connected, s1, 00:00:17
C>*    137.204.64.0/24 is directly connected, s1, 00:00:17
O      137.204.65.0/24 [110/10] is directly connected, s1, 00:00:17
C>*    137.204.65.0/24 is directly connected, s1, 00:00:17
O      137.204.66.0/24 [110/10] is directly connected, s1, 00:00:17
C>*    137.204.66.0/24 is directly connected, s1, 00:00:17
O      137.204.67.0/24 [110/10] is directly connected, s1, 00:00:17
C>*    137.204.67.0/24 is directly connected, s1, 00:00:17
O>*    137.204.100.0/24 [110/30] via 192.168.10.6, s1, 00:00:14
O>*    137.204.101.0/24 [110/50] via 192.168.10.6, s1, 00:09:32
O>*    192.168.10.0/30 [110/10] is directly connected, s1, 00:00:17
O>*    192.168.10.4/30 [110/10] is directly connected, s1, 00:00:17
O>*    192.168.10.8/30 [110/20] via 192.168.10.6, s1, 00:10:49
O>*    192.168.20.0/29 [110/40] via 192.168.10.6, s1, 00:09:32
```

Router R1.2:

```
raspberrypi# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

O>*    20.0.0.0/30 [110/20] via 192.168.10.10, s1, 00:01:57
O>*    20.0.0.4/30 [110/30] via 192.168.10.10, s1, 00:01:57
O>*    20.0.0.8/30 [110/30] via 192.168.10.10, s1, 00:01:57
C>*    127.0.0.0/8 is directly connected, lo
O      137.204.1.0/24 [110/10] is directly connected, s1, 00:00:17
C>*    137.204.1.0/24 is directly connected, s1
O>*    137.204.64.0/24 [110/30] via 192.168.10.2, s1, 00:01:57
O>*    137.204.65.0/24 [110/30] via 192.168.10.2, s1, 00:01:57
O>*    137.204.66.0/24 [110/10] via 192.168.10.2, s1, 00:01:57
O>*    137.204.67.0/24 [110/10] via 192.168.10.2, s1, 00:01:57
O>*    137.204.100.0/24 [110/30] via 192.168.10.10, s1, 00:00:14
O>*    137.204.101.0/24 [110/50] via 192.168.10.10, s1, 00:09:32
O      192.168.10.0/30 [110/10] is directly connected, s1, 00:03:22
C>*    192.168.10.0/30 is directly connected, s1
O>*    192.168.10.4/30 [110/10] via 192.168.10.10, s1, 00:03:22
O>*    192.168.10.8/30 [110/20] is directly connected, s1, 00:10:49
C>*    192.168.10.8/30 is directly connected, s1
O>*    192.168.20.0/29 [110/40] via 192.168.10.10, s1, 00:01:57
```

Router R1.3:

```
raspberrypi# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

O        20.0.0.0/30 [110/10] is directly connected, eth2, 02:39:13
C>*      20.0.0.0/30 is directly connected, eth2
O>*      20.0.0.4/30 [110/20] via 20.0.0.2, eth2, 02:38:31
O>*      20.0.0.8/30 [110/20] via 20.0.0.2, eth2, 02:25:35
C>*      127.0.0.0/8 is directly connected, lo
O>*      137.204.1.0/24 [110/10] via 192.168.10.9, eth1, 01:16:31
O>*      137.204.64.0/24 [110/20] via 192.168.10.5, eth0, 01:16:31
O>*      137.204.65.0/24 [110/20] via 192.168.10.5, eth0, 01:16:31
O>*      137.204.66.0/24 [110/20] via 192.168.10.5, eth0, 01:16:31
O>*      137.204.67.0/24 [110/20] via 192.168.10.5, eth0, 01:16:31
O>*      137.204.100.0/24 [110/30] via 20.0.0.2, eth2, 00:03:14
O>*      137.204.101.0/24 [110/40] via 20.0.0.2, eth2, 00:02:09
O>*      192.168.10.0/30 [110/20] via 192.168.10.5, eth0, 00:22:15
O        192.168.10.4/30 [110/10] via 192.168.10.10, eth0, 00:03:22
C>*      192.168.10.4/30 is directly connected, eth0
O        192.168.10.8/30 [110/10] is directly connected, eth1, 00:03:26
C>*      192.168.10.8/30 is directly connected, eth1
O>*      192.168.20.0/29 [110/30] via 20.0.0.2, eth2, 00:02:09
```

Router 0:

```
raspberrypi# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

O        20.0.0.0/30 [110/10] is directly connected, eth0, 00:47:15
C>*      20.0.0.0/30 is directly connected, eth0
O        20.0.0.4/30 [110/10] is directly connected, eth1, 01:44:00
C>*      20.0.0.4/30 is directly connected, eth1
O        20.0.0.8/30 [110/10] is directly connected, eth2, 00:33:09
C>*      20.0.0.8/30 is directly connected, eth2
C>*      127.0.0.0/8 is directly connected, lo
O>*      137.204.64.0/24 [110/30] via 20.0.0.1, eth0, 01:15:42
O>*      137.204.65.0/24 [110/30] via 20.0.0.1, eth0, 01:15:42
O>*      137.204.66.0/24 [110/30] via 20.0.0.1, eth0, 01:15:42
O>*      137.204.67.0/24 [110/30] via 20.0.0.1, eth0, 01:15:42
O>*      137.204.100.0/24 [110/30] via 20.0.0.6, eth1, 00:01:20
O>*      137.204.101.0/24 [110/30] via 20.0.0.6, eth1, 00:01:20
O>*      192.168.10.0/30 [110/30] via 20.0.0.1, eth0, 01:15:42
O>*      192.168.10.4/30 [110/20] via 20.0.0.1, eth0, 01:24:47
O>*      192.168.10.8/30 [110/20] via 20.0.0.1, eth0, 00:02:36
O>*      192.168.20.0/29 [110/20] via 20.0.0.6, eth1, 00:01:20
```

Router R2.1:

```
raspberrypi# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

O>*     20.0.0.0/30 [110/20] via 20.0.0.5, eth1, 00:15:07
O       20.0.0.4/30 [110/10] is directly connected, eth1, 00:15:57
C>*     20.0.0.4/30 is directly connected, eth1
O>*     20.0.0.8/30 [110/20] via 20.0.0.5, eth1, 00:15:07
C>*     127.0.0.0/8 is directly connected, lo
O>*     137.204.64.0/24 [110/40] via 20.0.0.5, eth1, 00:15:07
O>*     137.204.65.0/24 [110/40] via 20.0.0.5, eth1, 00:15:07
O>*     137.204.66.0/24 [110/40] via 20.0.0.5, eth1, 00:15:07
O>*     137.204.67.0/24 [110/40] via 20.0.0.5, eth1, 00:15:07
O>*     137.204.100.0/24 [110/20] via 192.168.20.2, eth0, 00:05:45
O>*     137.204.101.0/24 [110/20] via 192.168.20.3, eth0, 00:05:45
O>*     192.168.10.0/30 [110/40] via 20.0.0.5, eth0, 00:15:07
O>*     192.168.10.4/30 [110/30] via 20.0.0.5, eth0, 00:15:07
O>*     192.168.10.8/30 [110/40] via 20.0.0.5, eth0, 00:15:07
O       192.168.20.0/29 [110/20] is directly connected, eth0, 00:14:58
C>*     192.168.20.0/29 is directly connected, eth0
```

Router R2.2:

```
raspberrypi# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

O>*     20.0.0.0/30 [110/20] via 20.0.0.9, s1, 00:02:34
O>*     20.0.0.4/30 [110/20] via 20.0.0.9, s1, 00:02:34
O       20.0.0.8/30 [110/10] is directly connected, s1, 00:03:24
C>*     20.0.0.8/30 is directly connected, s1
C>*     127.0.0.0/8 is directly connected, lo
O>*     137.204.64.0/24 [110/40] via 20.0.0.9, s1, 00:02:34
O>*     137.204.65.0/24 [110/40] via 20.0.0.9, s1, 00:02:34
O>*     137.204.66.0/24 [110/40] via 20.0.0.9, s1, 00:02:34
O>*     137.204.67.0/24 [110/40] via 20.0.0.9, s1, 00:02:34
O       137.204.100.0/24 [110/10] is directly connected, s1, 00:03:24
C>*     137.204.100.0/24 is directly connected, s1
O>*     137.204.101.0/24 [110/20] via 192.168.20.3, s1, 00:02:34
O>*     192.168.10.0/30 [110/40] via 20.0.0.9, s1, 00:02:34
O>*     192.168.10.4/30 [110/30] via 20.0.0.9, s1, 00:02:34
O>*     192.168.10.8/30 [110/40] via 20.0.0.9, s1, 00:02:34
O       192.168.20.0/29 [110/10] is directly connected, s1, 00:03:24
C>*     192.168.20.0/29 is directly connected, s1
```

Router 2.3:

```
raspberrypi# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

O>*    20.0.0.0/30 [110/30] via 192.168.20.1, s2, 00:05:04
O>*    20.0.0.4/30 [110/20] via 192.168.20.1, s2, 00:05:04
O>*    20.0.0.8/30 [110/30] via 192.168.20.1, s2, 00:05:04
C>*    127.0.0.0/8 is directly connected, lo
O>*    137.204.64.0/24 [110/50] via 192.168.20.1, s2, 00:05:04
O>*    137.204.65.0/24 [110/50] via 192.168.20.1, s2, 00:05:04
O>*    137.204.66.0/24 [110/50] via 192.168.20.1, s2, 00:05:04
O>*    137.204.67.0/24 [110/50] via 192.168.20.1, s2, 00:05:04
O>*    137.204.100.0/24 [110/10] via 192.168.20.2, s2, 00:05:04
O      137.204.101.0/24 [110/10] is directly connected, s1, 00:11:19
C>*    137.204.101.0/24 is directly connected, s1
O>*    192.168.10.0/30 [110/50] via 192.168.20.1, s2, 00:05:04
O>*    192.168.10.4/30 [110/40] via 192.168.20.1, s2, 00:05:04
O>*    192.168.10.8/30 [110/40] via 192.168.20.1, s2, 00:05:04
O      192.168.20.0/29 [110/10] is directly connected, s2, 00:05:04
C>*    192.168.20.0/29 is directly connected, s2
```

Come si nota dalle tabelle, tutta la rete è conosciuta e raggiungibile da ciascun nodo.

Applicando il comando `traceroute`, che permette di vedere il percorso intrapreso e i router intercettati dei pacchetti inviati, tra un host virtuale h1 della LAN interna al Router2.3 e uno degli host virtuali presenti nel Router1.1 si ottiene un risultato di questo tipo:

```
mininet>h1 traceroute -n 137.204.64.1
traceroute to 137.204.64.1 (137.204.64.1) 30 hops max, 60 byte packets
1      137.204.101.254     46.700ms      46.710ms      45.833ms
2      192.168.20.1        51.019ms      52.514ms      51.385ms
3      20.0.0.5            51.506ms      65.532ms      65.087ms
4      20.0.0.1            51.338ms      51.478ms      50.081ms
5      192.168.10.5        67.237ms      54.925ms      90.833ms
7      137.204.64.1        80.348ms      81.543ms      80.798ms
```

In questa configurazione di rete i pacchetti che partono da h1 (in R2.3) vengono consegnati tramite s1 al router che attraverso lo switch s2 invia i pacchetti all'esterno. Successivamente il pacchetto viene consegnato al router scelto dal protocollo OSPF, in questo caso R2.1 con 192.168.20.1 che, a sua volta, lo consegnerà al router di backbone R0 (20.0.0.5). R0 lo consegna al router di bordo dell'area 1 R1.3 (20.0.0.1) che trasmette il pacchetto all'ultimo router rappresentato da R1.1 che, internamente, consegna il pacchetto alla destinazione h1. Si è dimostrato praticamente che c'è connessione tra LAN1 e LAN7.

Interessante è anche il test del percorso che un pacchetto fa per essere consegnato da una LAN virtuale ad un'altra. Ad esempio, facendo il tra un host della LAN1 e uno della LAN2 si ottiene:

```
mininet>h1 traceroute -n h2
traceroute to 137.204.65.1 (137.204.65.1) 30 hops max, 60 byte packets
1      137.204.65.1        16.373ms     24.016ms     14.973ms
```

Si nota che il pacchetto non incontra nessun router poiché sono entrambi nello stesso dominio di broadcast quindi il pacchetto è come se fosse consegnato in maniera diretta, giungendo subito alla destinazione.

È possibile verificare il percorso intrapreso dai pacchetti che partono dalla rete virtuale Mininet del router R2.1 e giunge nella rete virtuale di R1.1:

```
mininet>h1 traceroute -n 137.204.64.1
traceroute to 137.204.64.1 (137.204.64.1) 30 hops max, 60 byte packets
1      137.204.100.254     17.798ms     16.379ms     18.799ms
2      20.0.0.5            54.780ms     51.012ms     60.571ms
3      20.0.0.1            63.820ms     62.532ms     57.962ms
4      192.168.10.5        75.950ms     77.002ms     86.176ms
5      137.204.64.1        87.200ms     90.925ms     90.833ms
```

In questo caso i dati arrivano allo switch (137.204.100.254). Esso li manda al kernel che funge da router e provvede a spedirli verso l'esterno. Il pacchetto arriva quindi al router vicino (20.0.0.5 - R0) e attraverso R1.3 (20.0.0.1) e R1.1 (192.168.10.5) arriva alla rete virtuale di destinazione.

Si verifica la connessione anche tra la LAN5 e la LAN1, restando sempre nella stessa area:

```
mininet>h1 traceroute -n 137.204.64.1
traceroute to 137.204.64.1 (137.204.64.1) 30 hops max, 60 byte packets
1      137.204.1.254       27.148ms     153.822ms    125.661ms
2      192.168.10.1        266.646ms    258.326ms    284.837ms
3      137.204.64.1        287.186ms    288.000ms    288.548ms
```

Si nota che il pacchetto esce dal nodo R1.2 e arriva al router R1.1 nella porta eth1 (192.168.10.10) e entra nella rete virtuale arrivando a destinazione.

5. Conclusioni

Grazie alla creazione di questa rete composta da 7 nodi Raspberry Pi si è in grado di studiare il principio di funzionamento della rete Internet o sperimentare-implementare-testare nuovi protocolli di routing che non sarebbe possibile testare sulla rete Internet.

Un'altra utilità di questa rete potrebbe essere quella di permettere ai professori e agli studenti di testare ciò che viene spiegato nella teoria del routing IP su Internet, grazie alla sua trasportabilità. Infatti grazie alle piccole dimensioni dei Raspberry è stato necessario uno spazio minimo se si rapporta allo spazio occupato da 7 personal computer.

Questa rete è adatta anche per affacciarsi sul mondo Mininet e alle sue enormi potenzialità, sia per scopi didattici che di progetto di reti caratterizzate da una certa complessità.

Un aspetto sul quale ci si deve soffermare sono i costi sostenuti. Per creare una rete che, come già detto nell'introduzione, per come è impostata potrebbe funzionare con 1771 dispositivi, sono stati utilizzati 7 Raspberry Pi Model B+ (25€ circa per pezzo), 7 alimentatori (8€ l'uno), 7 schede SD (5.50€ l'una) e 9 adattatori Ethernet-USB (4€ ad adattatore). In totale sono stati utilizzati materiali per circa 305.50€ che è molto meno di quanto si spenderebbe per un solo personal computer.

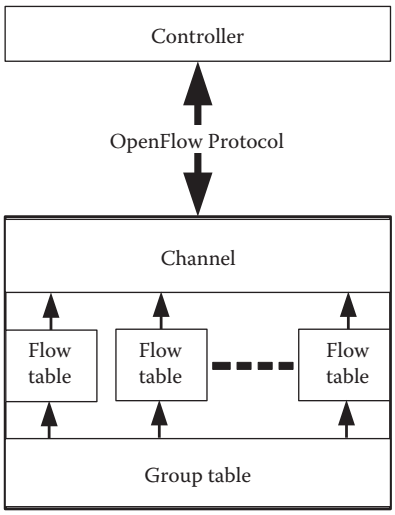
6. Appendice

6.1 SDN e OpenFlow

Una SDN (Software Defined Network) si basa sull'idea di separare la funzione di network forwarding, svolta dal data layer, e la funzione di controllo rete realizzata dal control plane. Questo permette un più semplice e flessibile controllo della rete oltre alla possibilità di virtualizzazione.

OpenFlow è un'implementazione SDN, dove il controllore della rete comunica con gli switch OpenFlow usando uno specifico protocollo attraverso un canale sicuro. Grazie a questa comunicazione, il controller è in grado di modificare le tabelle di instradamento dello switch.

Una semplice architettura di OpenFlow è data dalla figura seguente:



Open flow Switch

I vantaggi di tecnologie SDN basate su OpenFlow sono:
· adattare semplicemente le funzioni della rete a diversi business;
· indirizzare l'elevata larghezza di banda, cioè la natura dinamica delle reti del computer;
· ridurre drasticamente la complessità delle operazioni e della manutenzione.

Grazie a queste tipologie di reti, nel futuro non dovremo più programmare, configurare e fare il debug di ogni dispositivo di rete individualmente ma sarà possibile farlo in una postazione centralizzata e modificare la rete in base alle esigenze, che siano sperimentazione scientifica, controllo della rete casalinga piuttosto che un rete aziendale.

Le reti, che rappresentano la colonna portante di Internet, devono adattarsi ai cambiamenti senza essere troppo laboriose nel cambiamento di hardware e software. Aggiornando gli switch così che il control plane e il data plane possano essere separati, un controllore centralizzato può creare delle routes ottimizzate per instradare appropriatamente il traffico, rendendo le reti più efficienti e nettamente più flessibili.

Col nuovo protocollo OpenFlow, che sta diventando uno standard affermato, le SDN sono diventate molto più semplici da implementare. SDN disaccoppia control plane e data plane, quindi permette agli switch di dividere le due funzioni principali:

- il control plane genera le tabelle di instradamento;
- il data plane, usando le tabelle di instradamento generate dal control plane, determina dove i pacchetti devono essere mandati.

Il network è diventato una parte della critica infrastruttura del nostro business, delle nostre case e delle nostre scuole. Oggi, non c'è quasi nessun modo pratico di sperimentare con nuovi protocolli network (e.g. nuovi protocolli routing o alternative all'IP) in impostazioni sufficientemente realistiche (e.g. in scala reale del traffico di trasporto) per ottenere la fiducia necessaria per la loro vasta distribuzione. I network programmabili virtualizzati potrebbero ridurre le barriere d'entrata per nuove idee, aumentando il tasso di innovazione nelle infrastrutture network. Ma i piani per le strutture nazionali sono ambiziosi e ci vorranno anni prima che siano distribuite.

L'idea di base è semplice: si sfrutta il fatto che la maggior parte degli Ethernet switches moderni contengano dei flow-tables che eseguono a line-rate per implementare i firewall, NAT, QoS e per raccogliere statistiche. Mentre il flow table di ciascun venditore è diverso, abbiamo identificato un set comune interessante che funziona in molti switch e routers.
OpenFlow fornisce un protocollo aperto per programmare i flow-tables in switch e router differenti. Un amministratore di un network può ripartire il traffico in produzioni e research flows. I ricercatori possono controllare i propri flow scegliendo le strade che seguiranno i loro pacchetti e il processo che ricevono. In questo modo i ricercatori possono provare nuovi protocolli routing, modelli di sicurezza, schemi di indirizzo, anche alternativi all'IP.

Il percorso dei dati di uno switch OpenFlow consiste in una OpenFlow Table e un'azione associata ad ogni entrata di flow. L'insieme di azioni supportate da uno switch OpenFlow è estensibile, ma di seguito si descrivono i requisiti minimi per tutti gli switch.
Uno switch OpenFlow consiste di almeno tre parti: (1) Una Flow Table, con un'azione associata a ciascun flow in entrata, per comunicare allo switch come processare il flow. (2) un Secure Channel che connetta lo switch al remote control process (chiamato controller), permettendo ai comandi e ai pacchetti di essere inviati fra il controller e lo switch usando (3) il protocollo OpenFlow, che fornisce una via aperta e standardizzata di comunicazione fra il controller e lo switch.
Uno switch dedicato ad OpenFlow è un elemento datapath muto che invia i pacchetti fra le porte, come definito da un processo di controllo remoto. In questo contesto, i flussi sono definiti in senso lato, e sono limitati solo dalle capacità della particolare implementazione del Flow Table. Per esempio, un flusso potrebbe essere una connessione TCP, o tutti i pacchettidi un particolare indirizzo MAC o indirizzo IP, o tutti i pacchetti con lo stesso VLAN tag, o tutti i pacchetti dallo stesso switch-port.
Ogni flow in entrata è associato a una semplice azione; le tre azioni di base (che tutti gli switch OpenFlow devono supportare) sono:

1. L'invio di questi pacchetti flow a una data porta (o porte). Questo permette ai pacchetti di essere instradati attraverso il network. Nella maggior parte degli switch ci si aspetta che questo accada alla velocità di linea.
2. Incapsulare e inviare questi pacchetti di flow a un controller. Un pacchetto è inviato a Secure Channel, dove esso è incapsulato e spedito al controller. Tipicamente usato per il primo pacchetto

in un nuovo flow, così che un controller possa decidere se il flusso debba essere aggiunto al Flow Table. O in alcuni esperimenti, potrebbe essere usato per inviare tutti i pacchetti a un controller che li processi.
3. Lasciare questi pacchetti di flow. Possono essere usati per sicurezza, per smentire gli attacchi di servizio, o per ridurre una trasmissione a traffico scoperta simulata da end-hosts.

OpenFlow necessita di componenti essenziali sia nel data plane che nel controller plane. Un nodo di forwarding è fondamentale nel data plane e questo può essere sia un nodo HW-based sia una implementazione SW-based. Invece, il control plane consiste nel controller OpenFlow.

La separazione permette inoltre di rendere le reti astratte, semplificandone l'analisi. Infatti, molte Companies usano il protocollo OpenFlow per organizzare i loro data center per semplificare le operazioni che, nel caso di grandi quantità di dati, diventano velocemente molto complesse.
Un ulteriore e significante aspetto positivo del protocollo OpenFlow è la possibilità di creare semplicemente degli ambienti di emulazioni di reti, come di tools di sviluppo, che permetto all'utente di prendere confidenza con le reti tramite un'esperienza hands-on.

Le SDN implicano un cambio di paradigma su come le reti sono progettate e come lavorano. Esse sono il risultato di un trend dove la funzionalità hardware viene rimpiazzata da una implementazione software. Questa evoluzione è originata non solo dal continuo aumento di potenza dei calcolatori ma anche dall'evoluzione nello sviluppare software con l'aiuto di tools easy-to-use.

Uno dei più popolari SW-based OpenFlow switch è l'OpenVSwitch (OVS) che è spesso integrato nel kernel di Linux. Ci sono altri controllori open-source per ogni ambiente di sviluppo come ad esempio:
- Pox: scritto in Python;
- Nox: scritto in C;
- Floodlight: scritto in Java;

7. Bibliografia

Libri

Fei H., *Network Innovation through OpenFlow and SDN. Principles and Design*, Auerbach, 2014

Kurose J. F., Ross K. W., *Computer Networking. A Top-Down Approach*, Pearson, 2013

Lamm T., Odom S., *CCNP Routing Study Guide,* Alameda, Sybex, 2001

Pattavina A., *Reti di telecomunicazioni. Networking e Internet,* Milano, McGraw-Hill, 2003

Articoli

Bob Lantz, Brandon Heller, Nick McKeown, *A Network in a Laptop: Rapid Prototyping for Software-Defined Networks,* in Hotnets '10*,* October 20–21, 2010, Monterey, CA, USA.

Bob Lantz, Brian O'Connor, Brandon Heller, Nikhil Handigol, Vimal Jeyakumar, *Mininet: An instant Virtual Network on your Laptop,*

Marcel Großmann, Stephan J.A. Schuberth, *Auto-Mininet: Assessing the Internet Topology Zoo in a Software-Defined Network Emulator*, in 7ter Workshop Leitungs, Amburgo, 05-06.09.13

Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, Jonathan Turner, *OpenFlow: Enabling Innovation in Campus Networks*, 14 Marzo 2008

Online

Introduction to Mininet. Disponibile online: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet

Mininet Python Style. Disponibile online: https://github.com/mininet/mininet/wiki/Mininet-Python-Style

OpenFlow Tutorial. Disponibile online: http://archive.openflow.org/wk/index.php/OpenFlow_Tutorial

Software-Defined Networking (SDN). Disponibile online: https://en.wikipedia.org/wiki/Software-defined_networking