2017

RELAZIONE PROGETTO ONOS

STUDENTE: MARCO GRASSIA (O55000330)

OBIETTIVO DEL PROGETTO

L'obiettivo del progetto è quello di realizzare un'applicazione per ONOS che gestisca le comunicazioni tra host in un datacenter. L'applicazione deve rispettare tre requisiti fondamentali:

1. Ogni coppia di host deve comunicare adoperando sempre lo shortest path tra i due. Nel caso esistano più path con la stessa lunghezza, deve essere scelto, di volta in volta, uno tra questi in maniera casuale
2. Le VM appartenenti allo stesso tenant devono essere in grado di comunicare solo tra di loro
3. Deve essere possibile migrare le VM in esecuzione su un host verso un nuovo host e ciò deve essere trasparente a livello IP

Inoltre, il funzionamento dell'applicazione deve essere verificato su una rete Mininet emulata che presenti topologia Clos.

AMBIENTE DI LAVORO

Primo step, necessario alla realizzazione dell'applicazione, è quello del setup dell'ambiente di lavoro.

AMBIENTE

Si è lavorato in ambiente Linux (VM Ubuntu del tutorial o su una installazione "reale" di Linux Mint, a seconda della disponibilità). Sono stati usati Atom e l'IDE IntellJ Idea per la scrittura del codice, la cui gestione è stata affidata a Git.

MACCHINA VIRTUALE

La macchina virtuale del tutorial su ONOS presenta problemi all'installatore di pacchetti. Per risolvere è bastato seguire le istruzioni di recupero riportate negli stessi errori.

```
sudo dpkg --configure -a
sudo apt-get -f install
```

SETUP

È stato installato quanto necessario per procedere con lo sviluppo.

Alcuni di questi passaggi sono già stati svolti nella macchina virtuale del tutorial, ma sono stati necessari avendo lavorato, in un primo momento, su partizione fisica e non in VM.

MININET

L'installazione di Mininet è stata effettuata a partire dal codice nel repository git e non dai repository di Ubuntu. Da console:

```
git clone git://github.com/mininet/mininet
cd ./mininet
git checkout -b 2.2.2
./util/install.sh -nfv
```

Lavorando su Linux Mint è stato necessario bypassare, commentandolo, il check della distro nell'install.sh. Ciò non costituisce un problema dato che LM è Ubuntu based.

Una volta installato, per verificarne il corretto funzionamento, da console:

```
mn -version              # must be >= 2.2.1
sudo mn --test pingall   # test if working
```

ONOS

Il setup di ONOS ha richiesto diversi passaggi:

- Installazione dell'SDK di Java e set come versione di default per il sistema

```
sudo apt-get install software-properties-common -y && \
sudo add-apt-repository ppa:webupd8team/java -y && \
sudo apt-get update && \
echo "oracle-java8-installer shared/accepted-oracle-license-v1-1 select true" | sudo debconf-set-
selections && \
sudo apt-get install oracle-java8-installer oracle-java8-set-default -y
```

- Download del source di ONOS da GitHub

```
git clone https://gerrit.onosproject.org/onos
```

- Build del source

```
export ONOS_ROOT=/home/marco/Scrivania/ONOS/onos/
cd $ONOS_ROOT
git checkout 1.10.3
$ONOS_ROOT/tools/build/onos-buck build onos --show-output
source $ONOS_ROOT/tools/dev/bash_profile
# $ONOS_ROOT/tools/build/onos-buck test
$ONOS_ROOT/tools/build/onos-buck run onos-local -- clean debug
```

- Apertura della console di ONOS

```
export ONOS_ROOT=/home/marco/Scrivania/ONOS/onos/
$ONOS_ROOT/tools/test/bin/onos localhost
```

- Apertura della WebUI per test del corretto funzionamento:
  http://localhost:8181/onos/ui
  Credenziali di default: onos / rocks

SCRIPT TOPOLOGIA CLOS

Il primo punto della consegna è quello di scrivere uno script Python che crei una rete a topologia Clos in Mininet. La configurazione possibile riguarda il numero di core switch e il fanout per ogni nodo.

È stato usato lo scheletro di script fornito via email, al quale è stato aggiunto il codice necessario per la creazione della topologia richiesta. In particolare, viene calcolato il numero di host e di switch (core, aggregation ed edge) necessari, sulla base del numero di core e del fanout. Vengono, quindi, creati gli elementi ed i link necessari.

Durante il testing iniziale era, però, osservabile la costante mancanza di uno switch – appartenente, di volta in volta, o al livello edge o all'aggregation – nonostante in Mininet fossero tutti presenti, inclusi i relativi link. L'analisi dell'output nella console di ONOS ha permesso di individuare il problema. Qui veniva, infatti, segnalato un conflitto tra due switch a causa del DPID duplicato. Il problema è stato risolto fornendo tale parametro al metodo di creazione degli switch ed eliminando l'assegnazione automatica da parte di Mininet.

Lo script può essere avviato tramite console con la seguente sintassi:

```
sudo python clos_topo_builder.py -c CORE -f FANOUT
```

Nel nostro caso, il numero di core switch desiderati è due, così come il fanout. Il comando di avvio è, dunque:

```
sudo python clos_topo_builder.py -c 2 -f 2
```

DATACENTER

Uno dei due punti della consegna è quello di creare un'applicazione per ONOS che gestisca i tenant e le migrazioni delle VM da un host ad un altro ed in modo trasparente.

APPLICAZIONE DI BASE

Per l'implementazione è stata usata come base l'applicazione FWD, inclusa nel pacchetto di apps esempio della cartella apps/, disponibile nella root del sorgente di ONOS stesso.

Le modifiche iniziali per rendere l'applicazione Datacenter indipendente sono:

- Copia della cartella onos/apps/fwd → onos/apps/datacenter
- [Non necessario, in quanto ora viene usato BUCK] Aggiunto il modulo nel file pom.xml in apps/
- Aggiunta in modules.defs della root:
    - Nella sezione ONOS_APPS
      `'//apps/datacenter:onos-apps-datacenter-oar',`
    - Nella sezione APP_JARS
      `'//apps/datacenter:onos-apps-datacenter',`

  Questo permette il build dell'applicazione insieme ad ONOS

- Rename del package, tramite Refactor, da org.onosproject.fwd → emarco.datacenter
- Ricerca per altre ricorrenze del nome del package e sostituzione delle stesse, come sopra
- Cambiamenti nell'iml di config e rename dell'applicazione a onos-app-datacenter

SCELTA CASUALE DEL PATH

L'applicazione FWD, usata come base per l'applicazione, interpella il *TopologyService* tramite metodo *getPaths* per ottenere i cammini minimi disponibili, scegliendo di fatto il primo tra questi.

La consegna richiede, invece, una scelta randomica all'interno del set. Si è proceduto, dunque, alla modifica del metodo *pickForwardPathIfPossible*, delegato alla scelta del Path, che adesso restituisce un elemento scelto (pseudo)randomicamente.

La scelta di usare un generatore di numeri pseudo-random piuttosto che random segue, ovviamente, da considerazioni di tipo prestazionale.

TENANTS

OBIETTIVI

L'obiettivo è quello di permettere la comunicazione tra soli host appartenenti allo stesso tenant.
È però necessario cercare di minimizzare il delay dovuto all'elaborazione, del quale una componente importante è data dal tempo di accesso alla struttura dati che mantiene in memoria le associazioni tra host e tenant, anche a scapito di una maggiore occupazione di memoria.

STRUTTURA DATI

A seguito di qualche ricerca a riguardo, la scelta della struttura dati è ricaduta sulle HashMap, poiché il tempo di accesso, a seguito di un primo calcolo dell'hash della chiave, è costante ed è dell'ordine di O(1).

In particolare, per migliorare ulteriormente è stata adoperata la ConcurrentHashMap, la quale divide in sottocomponenti la mappa e utilizza più thread per l'accesso.

IMPLEMENTAZIONE TENANTS + FIREWALL

Di seguito gli step dell'implementazione:

- Creazione del servizio manager dei tenant, in modo da potervi accedere tramite comandi da CLI
    - Aggiunta dell'interfaccia *TenantsMapService*
    - Aggiunta della relativa implementazione *TenantsMapProvider*
    - Il servizio include e gestisce la mappa-cache contenente le associazioni host-tenant

- [BONUS] Comando di update dei tenant:
    - Implementazione:
        - Creazione della classe handler *TenantsMapCommand* nella cartella ${APP_ROOT}/src/main/java/emarco/datacenter/cli
        - Registrazione del comando tramite l'aggiunta del file ${APP_ROOT}/src/main/resources/OSGI-INF/blueprint/shell-config.xml e relativo contenuto
    - Il comando accetta come unico parametro la locazione del file e richiama la classe principale *ReactiveForwarding* per segnalare l'aggiornamento. Questa si occupa di eseguire il purge delle Flow Rules precedentemente settate dall'applicazione stessa, evitando che possano consentire il bypass del blocco

- Modifica del metodo *process*:
    - Viene ora confrontato il tenant di appartenenza degli host, ottenuto dal servizio apposito tramite IP
        - Se i tenant dei due host corrispondono si procede all'elaborazione regolare del pacchetto. Ciò vale anche nel caso in cui i tenant non siano settati. Infatti, host non appartenenti a nessun tenant potranno comunicare solo tra loro
        - In caso contrario, viene settata una regola permanente con *TrafficTreatment* di tipo drop

- All'activate dell'applicazione viene mostrato nella console quale è la directory di default dove posizionare il file per l'auto load che avviene in questa fase. È comunque possibile caricare un file qualsiasi in un secondo momento.

TESTING

Il corretto funzionamento del sistema è stato testato, inizialmente, ricorrendo a dei semplici ping tra host di tenant diversi.
Successivamente, è stato introdotto un secondo file di configurazione (con pattern di divisione degli host 0-7, 8-15; il primo alternava, invece, tra pari e dispari) e si è provato ad aggiornare le assegnazioni durante un pingall. Il risultato di seguito:

```
h0 -> X h2 X h4 X h6 X h8 X h10 X h12 X h14 X
h1 -> h0 h2 h3 h4 h5 h6 h7 X X X X X X X
```

Il sistema si comporta, dunque, come voluto.

COMANDI DI ESEMPIO

- Da bash:
    - Symlink al file dei tenants nella directory di default, per permettere il load dei tenants all'activate dell'applicazione:

      ```
      ln -s '${PROJECT_ROOT}/tenants.csv' '/tmp/onos-1.10.0/apache-karaf-3.0.8'
      ```
- Da console di ONOS:
    - Update dei tenant a partire dal file nella posizione di default:

      ```
      tenantsmap
      ```
    - Update dei tenants a partire da file:

      ```
      tenantsmap PATH/FILE
      ```

HOST MIGRATION

OBIETTIVO

L'obiettivo di questa sezione è quello di rendere trasparente la migrazione di una macchina virtuale da un host sorgente ad un altro di destinazione, come nell'esempio seguente:

Dati tre host (A, B e C) appartenenti allo stesso tenant:

> Vogliamo eseguire un ping da A verso B e, successivamente, tramite un comando da tastiera su console di ONOS, cancellare l'instradamento attuale e fare in modo che tutti i pacchetti IP diretti verso B subiscano la modifica dell'IP di destinazione in modo che arrivino a C, e viceversa tutti i pacchetti provenienti da C siano modificati in modo che ad A sembrino arrivare da B.
>
> Il risultato finale è che il ping non dovrebbe risentire del cambiamento di host e il tutto dovrebbe avviene senza perdita di pacchetti.

IMPLEMENTAZIONE

Il redirect trasparente è stato ottenuto come segue:

- È stato aggiunto, in maniera similare a quanto fatto per i tenant, un comando da console specifico per la migrazione
- È stato aggiunto, ancora in modo analogo al sistema dei tenant, un servizio per la migrazione che tenga traccia degli IP degli Host migrati e delle relative destinazioni. Questo presenta e gestisce due Map, nuovamente ConcurrentHashMap per ragioni prestazionali, che contengono, rispettivamente, le associazioni:
    - <IP hostMigrato, IP hostDestinazione> per verificare se è necessario il redirect verso una nuova destinazione
    - <IP hostDestinazione, IP hostMigrato> per verificare se è necessario il modificare il campo IP sorgente all'IP originario. Questa mappa è ridondante, ma permette di evitare un'operazione di ricerca della chiave (Value -> Key) a partire da un certo valore, molto costosa dal punto di vista computazionale, a fronte di una maggiore occupazione di memoria
- La classe *ReactiveForwarding* ha subito le seguenti modifiche:
    - Il codice è stato modificato, ove opportuno, per snellirlo, migliorandone la modificabilità e la riusabilità
    - Sono stati aggiunti metodi relativi a:
        - Migrazione
        - Ricerca di host by IP

        - Ricerca e rimozione delle flow rules che presentano come selettore del traffico un IP sorgente o destinazione. Il metodo si occupa di controllare anche se le regole sono state settate usando come selettore l'indirizzo MAC dello stesso host
        - Ricerca del path verso un host tramite IP
- Cambiamenti al metodo *process*:
    - Analisi dei pacchetti:
        - [Una volta controllato che gli host appartengano allo stesso tenant]
        - Identificazione dei pacchetti diretti verso host migrati
            - Set dell'IP di destinazione a quello della nuova destinazione, noto dal comando utente
            - Set del MAC di destinazione a quello della nuova destinazione, se noto all'*HostService*. ~~In caso contrario viene settato al valore di broadcast e il pacchetto mandato in flood senza installazione di regole associate~~. [A causa dell'implementazione non completa del gestore del context di OpenFlow, quanto tagliato non avviene. Il metodo *send* del pacchetto associato al context del gestore pacchetti adopera attualmente solo il *TrafficTreatment* relativo alla porta di out. Per la migrazione è, dunque, richiesta la conoscenza dell'host da parte dell'*HostService*, che avviene non appena genera qualsiasi tipo di traffico.]
        - Identificazione dei pacchetti uscenti da Host destinazione di VM migrate
            - Set dell'IP sorgente a quello dell'host originario
            - Set del MAC a quello dell'host originario
    - Identificazione della porta corretta dello switch su cui mandare il pacchetto per ottenere il cammino minimo, scelto ancora randomicamente tra quelli disponibili
    - Installazione della regola e uscita del pacchetto

TESTING

Il test del corretto funzionamento del redirect ha seguito i seguenti step:

- Test della connettività tra i due host scelti e appartenenti allo stesso tenant tramite ping, in modo da assicurarci che la flow rule venga installata
- Migrazione dell'host:
  ```
  migratehost sourceIP destinationIP
  ```
- Visualizzazione del flusso di ping sugli host:
    - Dalla console di Mininet, per aprire le console degli host
      ```
      xterm pingHostSource migratedHost destinationHost
      ```
    - Nelle console apertesi:
      ```
      tcpdump ip proto \\icmp
      ```
- Ignore dei ping sull'host migrato:
  ```
  echo 1 >/proc/sys/net/ipv4/icmp_echo_ignore_all
  ```
- Set a down dell'interfaccia di rete dell'host migrato
  ```
  ip link set hX-eth0 down
  ```

COMANDI DI ESEMPIO

- Da shell di ONOS:
    - Migrazione di un host:
      ```
      migratehost sourceIP destinationIP
      ```

CODICE

Il codice dell'applicazione è commentato, ove necessario, ed è disponibile nel repository git insieme allo script per la creazione della topologia Clos e ad altri ausiliari usati durante l'implementazione.

Per quanto riguarda le differenze con l'applicazione FWD di partenza, queste sono osservabili tramite git diff.

```
git diff -M fcfe42899511f5058a8fca374e4e7ffda5edb080
```

o, per utilizzare tool esterni di diff (eg. Meld)

```
git difftool -M fcfe42899511f5058a8fca374e4e7ffda5edb080
```

La flag -M risulta necessaria poiché alcuni file sono stati spostati senza ricorrere al tracciamento di git.