ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
SEDE DI CESENA

Scuola di Ingegneria ed Architettura

Corso di Laurea in Ingegneria dei Sistemi Elettronici per lo Sviluppo Sostenibile

IL PROTOCOLLO OVSDB PER LA GESTIONE DI SWITCH ETHERNET VIRTUALI

Tesi di Laurea di:
LUCA FALCO

Relatore:
Prof. Ing.
CERRONI WALTER

SESSIONE III
ANNO ACCADEMICO 2017–2018

PAROLE CHIAVE

Software-Defined Networking

Open vSwitch

Protocollo OVSDB

ovsdb-server

Contents

Acronyms

IEEE Institute of Electrical and Electronics Engineers.

IP Internet Protocols.

IPSEC Internet Protocol Security.

IT Information Technology.

LAN Local Area Network.

MAC Media Access Control.

OF OpenFlow.

OVS Open Virtual Switch.

OVSDB Open vSwitch Database.

QOS Quality of Service.

RFC Request for Comments.

SND Software Defined Networking.

SSL Secure Sockets Layer.

TCP Transmission Control Protocol.

VLAN Virtual LAN.

VM Virtual Machine.

Introduzione

Negli ultimi anni si è potuto vedere come, a causa dell'aumento esponenziale di utenti connessi alla rete internet, le infrastrutture server sono a mano a mano cresciute e con esse la complicatezza nel gestirle. Basti pensare che, solo in Italia, gli utenti connessi ad internet sono 43 milioni, mentre quelli attivi su mobile sono oltre 38 milioni (report *Global Digital 2018*). La crescente necessità di poter accedere ai contenuti aziendali e alle applicazioni da qualunque luogo, in ogni momento e con qualsiasi dispositivo ha generato un'inevitabile rischiesta alle infrastrutture e alle risorse IT di più sicurezza, conformità e controllo. Per tale motivo si è cercato un modo per rendere le reti abbastanza elastiche in modo da supportare le esigenze del traffico.

Un primo approcio a questo problema lo si ha avuto con la diffusione di tecnologie di "*cloud computing*". Con "*cloud computing*" ci riferiamo a quella serie di principi rivolti alla fornitura di risorse per le infrastrutture di elaborazione, rete e storage, servizi, piattaforme e applicazioni, accessibili on-demand e attraverso qualsiasi rete. Tali risorse di infrastrutture, servizi e applicazioni sono messe a disposizione dai cloud, ovvero pool di risorse virtuali orchestrati da software di gestione e automazione. Tra i vari tipi di servizi cloud troviamo diverse categorie: la piattaforma distribuita come un servizio (*PaaS*, *Platform as a Service*), i software come un servizio (*SaaS*, *Software as a Service*) e l'infrastruttura distribuita come un servizio (*IaaS*, *Infrastructure as a Service*). Nel contesto del cloud computing, tutti i server vengono virtualizzati e distribuiti sotto forma di una macchine virtuali o VM (virtual machine).

E' proprio l'utilizzo della virtualizzazione nelle reti che ha portato all'emergere di varie soluzioni basate su software: per esempio, nelle reti dei data center vengono usati switch virtuali come *Open vSwitch* (OVS) per interconnettere macchine virtuali. Tale soluzione permette di offrire una buona flessibilità oltre ad avere un controllo centrale, dinamico e facilmente scalabile della rete. Di contro sono necessarie risorse hardware prestanti e un'infrastruttura

in certi casi ridondante per poterne garantire l'affidabilità.

Questa tesi si concentrerà nell'analizzare la *Software-Defined Networking* e in particolare il protocollo *OVSDB* (*Open vSwitch Database*) il quale ha permesso di avere un'interfaccia di gestione moderna e programmatica per la gestione e l'automazione delle reti. Vedremo come, grazie ad esso, è possibile gestire le implementazioni Open vSwitch attraverso degli appositi comandi da remoto i quali, sfruttando i metodi JSON-RPC, ci consentono di modificare il database di un server.

Nel Capitolo 1 andremo ad esaminare il *Software Defined Networking*, i vantaggi che il suo utilizzo comporta (rispetto al networking tradizionale) e i termini che spesso vengono correlati ad esso (*OpenFlow* e *Open vSwitch*). Vedremo poi di analizzare il protocollo OpenFlow.

Open vSwitch, spesso abbreviato con *OVS*, è una implementazione open source di uno switch virtuale. Nel Capitolo 2 viene analizzata la sua struttura e le operazioni *OVSDB* che utilizza al suo interno. Successivamente (nel Capitolo 3) approfondiremo il protocollo *OVSDB*, cardine di questa tesi. Tratteremo le sue specifiche, in particolare come viene impostata la connessione tra client e server, le operazioni che possiamo svolgere sul database e i principali metodi usati dal protocollo.

Dopo questa attenta analisi, nel Capitolo 4 è presentato un caso pratico di utilizzo di *OVSDB* in cui, dopo una prima parte in cui è descritto il testbed e la configurazione delle macchine usate, vedremo come vengono scambiati i messaggi tra client e server e il loro contenuto JSON.

Le conclusioni di questa tesi sono infine discusse nel Capitolo 5.

Chapter 1

Software-Defined Networking

Oggigiorno, mentre nelle reti tradizionali è possibile trovare i tre piani dati, di controllo e di gestione implementati insieme nei firmware di router e switch, con il *Software Defined Networking* il control plane (nel quale troviamo i meccanismi di calcolo e di segnalazione che indicano alla rete quale sia, ad esempio, il percorso che i dati devono effetuare) e il data plane (la parte che fisicamente instrada i pacchetti) vengono separati tra loro per permettere un controllo più flessibile nell'amministrazione delle reti. Infatti mentre nel networking tradizionale i due piani sono eseguiti sullo stesso dispositivo fisico (per esempio uno switch che calcola il percorso del pacchetto e lo instrada sulle sue opportune porte), con il SDN le funzioni di control plane sono svolte da un software (il controller) che viene eseguito su un server centrale. È lui che decide il percorso dei pacchetti scegliendo non solo in funzione della destinazione e della topologia della rete ma anche da altre informazioni a cui lo switch tradizionale non può accedere (per esempio se è necessario che certi pacchetti abbiano la priorità rispetto ad altri per un certo periodo di tempo). Il SDN permette quindi di avere sia un controllo dinamico del traffico sia una funzione di automazione, caratteristiche molto cercate dai provider di rete.

Tra i possibili vantaggi troviamo:

- Le risorse di rete sono fornite in modo più rapido;
- Maggiore automazione e riduzione delle spese operative;
- Più flessibilità e personalizzazione della configurazione di rete;
- Più sicurezza.

Ciò non toglie che implementare le reti basate su software richieda un investimento iniziale ingente (per la sostituzione degli apparati e la revisione complessiva del network).

1.1 Relazione tra OpenFlow e Open vSwitch

Poiché il Software Defined Networking è diventato popolare in applicazioni dinamiche e ad elevata larghezza di banda (ad esempio, il cloud computing), termini correlati come Open vSwitch e OpenFlow sono molto discussi dai tecnici IT. Sebbene siano stati introdotti da un po', Openv Switch e OpenFlow ancora confondono le persone in alcuni aspetti. E la domanda più frequente è quale sia la relazione e la differenza tra OpenvSwitch e OpenFlow.

Tradizionalmente, gli hardware di rete di diversi fornitori spesso dispongono di speciali sistemi di configurazione e gestione, che limitano l'interazione tra router e switch di produttori diversi. Per risolvere questo problema, OpenFlow è stato creato come protocollo di rete programmabile aperto per la configurazione e la gestione degli switch di rete Gigabit di vari fornitori. Consente di scaricare il piano di controllo di tutti gli switch su un controller centrale e consente a un software centrale di definire il comportamento della rete (permettendo agli amministratori di rete di gestire e controllare il flusso di traffico tra apparecchiature di vari produttori). La programmabilità del piano di controllo permette di ridurre i costi operativi spostando l'onere della configurazione e della gestione dalle persone alla tecnologia tramite l'automazione.

Open vSwitch e OpenFlow sono entrambi utilizzati per applicazioni SDN. OpenFlow è uno dei primi standard SDN. Open vSwitch, invece, è una implemetazione open source di uno switch virtuale che lavora su più livelli della pila protocollare OSI (*Open Systems Interconnection*). E' inoltre un componente OpenStack SDN; quest'ultimo è un sistema operativo cloud che permette di controllare vasti pool di risorse di elaborazione, storage e di rete in un data center, permettendo agli amministratori di gestire il tutto tramite una dashboard. Per quanto riguarda la loro relazione, OpenvSwitch è una delle implementazioni più popolari di OpenFlow. Oltre a OpenFlow, OpenvSwitch supporta anche altri protocolli di gestione degli switch come OVSDB (Open vSwitch Database Management Protocol). Vediamo ora di analizzare un pò più a fondo il protocollo Openflow.

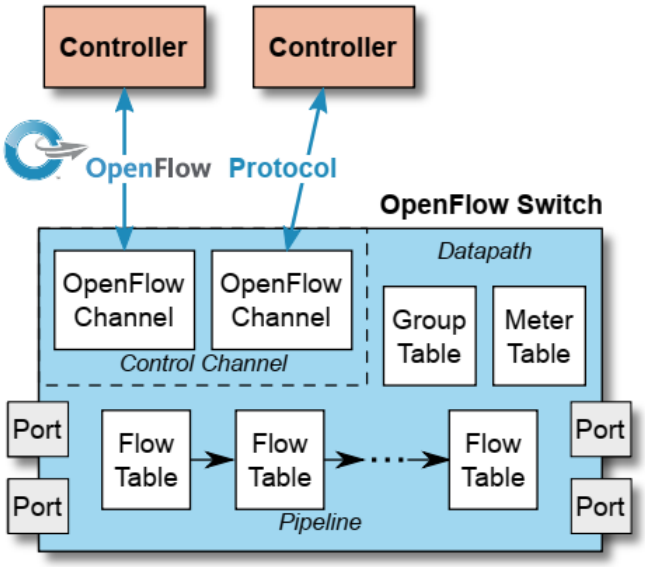

Figure 1.1: Componenti principali di uno switch OpenFlow (*OpenFlow Switch Specifications*).

1.2 Come funziona il protocollo OpenFlow

OpenFlow è generalmente costituito da tre componenti: il controller OpenFlow, il protocollo OpenFlow e una catena di tabelle di flusso configurate sullo switch OpenFlow (Figura 1.1). Il protocollo OpenFlow è come un supporto per il controller che comunica in modo sicuro con lo switch OpenFlow. Il controller OpenFlow può impostare regole sui comportamenti di trasmissione dei dati di ogni dispositivo di inoltro attraverso il protocollo OpenFlow. Le tabelle di flusso installate sullo switch spesso memorizzano una raccolta di flow entries. Pertanto, quando un pacchetto di dati arriva allo switch OpenFlow, lo switch cercherà le flow entries corrispondenti nelle tabelle di flusso ed eseguirà le azioni necessarie. Se non viene trovata alcuna corrispondenza in una tabella di flusso, il risultato dipende dalla configurazione della voce di flusso table-miss: ad esempio, il pacchetto può essere inoltrato ai controller sul canale OpenFlow, lasciato cadere o può continuare alla successiva tabella di flusso.

Figure 1.2: Componenti principali di una flow entry in una tabella di flusso.

1.2.1 Le tabelle di flusso

Una tabella di flusso consiste in una serie di voci di flusso (Figura 1.2). Ogni voce della tabella di flusso contiene:

- **match fields**: campi da abbinare ai pacchetti. Questi consistono nelle porte di ingresso e nelle intestazioni dei pacchetti;
- **priority**: corrisponde alla priorità delle voci di flusso;
- **counters**: contatori aggiornati quando i pacchetti sono abbinati;
- **instructions**: per modificare il set di azioni o l'elaborazione della pipeline;
- **timeouts**: tempo massimo o tempo di inattività prima che il flusso venga fatto scadere dallo switch;
- **cookie**: valore di dati "opachi" scelti dal controller. Può essere usato dal controller per filtrare le flow entries interessate da richieste di statistiche, di modifica o di eliminazione del flusso;
- **flags**: i flags alterano il modo con cui le voci di flusso sono gestite.

1.2.2 Matching

Quando arriva un pacchetto nella pipeline (il percorso composto da più tabelle di flusso), esso viene elaborato dalla tabella di flusso zero (questa è la prima tabella e deve sempre essere presente in ogni switch). Non essendo ancora stati processati da nessuna tabella, i pacchetti avranno metadata e action set vuoti; i campi di match e di priorità verrano poi confrontati con le flow entries per cercare una corrispondenza. Se non è presente nessuna corrispondenza, il pacchetto viene eliminato. Nel caso si abbia invece corrispondenza con una o più voci, viene processata quella con priorità maggiore. Vengono inoltre aggiornati i contatori associati a quella voce, viene eseguita l'istruzione presente nel campo *instruction* e infine il pacchetto è inoltrato

alla porta d'uscita o ad una tabella (che può essere una group table, una meter table o una flow table successiva nella pipeline).

Nel caso si abbia un'unica corrispondenza con la voce table miss, il pacchetto viene processato seguendo l'azione ad essa associata. Tra le varie azioni possiamo avere l'inoltro ad una flow table successiva nella pipeline, la cancellazione del pacchetto o l'invio al controller (che deciderà se creare una nuova entry o cancellare il pacchetto).

E' importante notare come non sia possibile indirizzare il flusso verso una flow table precedente a quella attuale (l'inoltro è possibile solo verso tabelle successive della pipeline). Una volta che il flusso viene diretto alla porta d'uscita, l'action set del flusso che si è accumulato fino ad ora viene eseguito e il pacchetto è messo in coda per uscire.

1.2.3 Action set

Un action set è la lista di azioni associata ad un pacchetto. Queste vengono aggiunte durante l'elaborazione delle tabelle e vengono eseguite quando il pacchetto esce dal processo della pipeline. Questo set è vuoto di default. Una flow entry può modificare l'action set usando un'istruzione *Write-Action* o un'istruzione *Clear-Action* associata con un particolare match. L'action set è trasportato tra le tabelle di flusso. Quando un instruction set di una flow entry non contiene una istruzione *GoTo-Table*, il processamento della pipeline è interrotto e le azioni nell'action set del pacchetto vengono eseguite. Un action set contiene al massimo un'azione per ogni tipo. Le azioni servono per descrivere le operazioni di inoltro di pacchetti, di modifica dei pacchetti, dell'action set e/o alla pipeline.Le azioni possibili definite dalla specifica OpenFlow sono le seguenti:

- **Set-queue**: imposta l'ID della coda del pacchetto corrispondente alla porta di uscita (viene usata per lo scheduling e l'inoltro dei pacchetti);
- **Push-tag, Pop-tag**: inserisce o toglie il campo tag da un pacchetto VLAN o MPLS;
- **Drop**: è un'azione non esplicita (risultato o di un instruction set vuoto di una pipeline o di un'istruzione clear-actions);
- **Group**: processa il pacchetto attraverso un gruppo;
- **Output**: inoltra i pacchetti verso la porta di uscita.

Chapter 2

Open vSwitch

2.1 Caratteristiche e componenti

Open vSwitch è uno switch software multilayer con licenza Open Source Apache 2.0. Il suo obiettivo è quello di implementare una piattaforma di switch di rete multistrato virtuale, che supporti le interfacce di gestione standard, e aprire al controllo e alla programmabilità delle funzioni di inoltro. Open vSwitch è adatto a funzionare come switch virtuale negli ambienti con VMs. Oltre ad esporre le interfacce di controllo standard e di visibilità al layer di networking virtuale, è stato progettato per supportare la distribuzione su più server fisici. Open vSwitch supporta diverse tecnologie di virtualizzazione tra cui Xen/XenServer, KVM e VirtualBox.

La maggior parte del codice è scritta in linguaggio C non dipendente dalla piattaforma ed è facilmente trasferibile in altri ambienti. La versione corrente di Open vSwitch supporta le seguenti funzionalità:

- Modello standard VLAN 802.1Q con trunk e porte di accesso;
- Collegamento NIC con o senza LACP (*Link Aggregation Control Protocol*, che consente di raggruppare più porte fisiche per avere un unico canale logico) su interruttore a monte;
- NetFlow, sFlow® e mirroring (sono dei protocolli e tecniche di monitoraggio della rete) per una maggiore visibilità;
- Configurazione QoS (Quality of Service), oltre a policing;
- OpenFlow 1.0 più numerose estensioni.

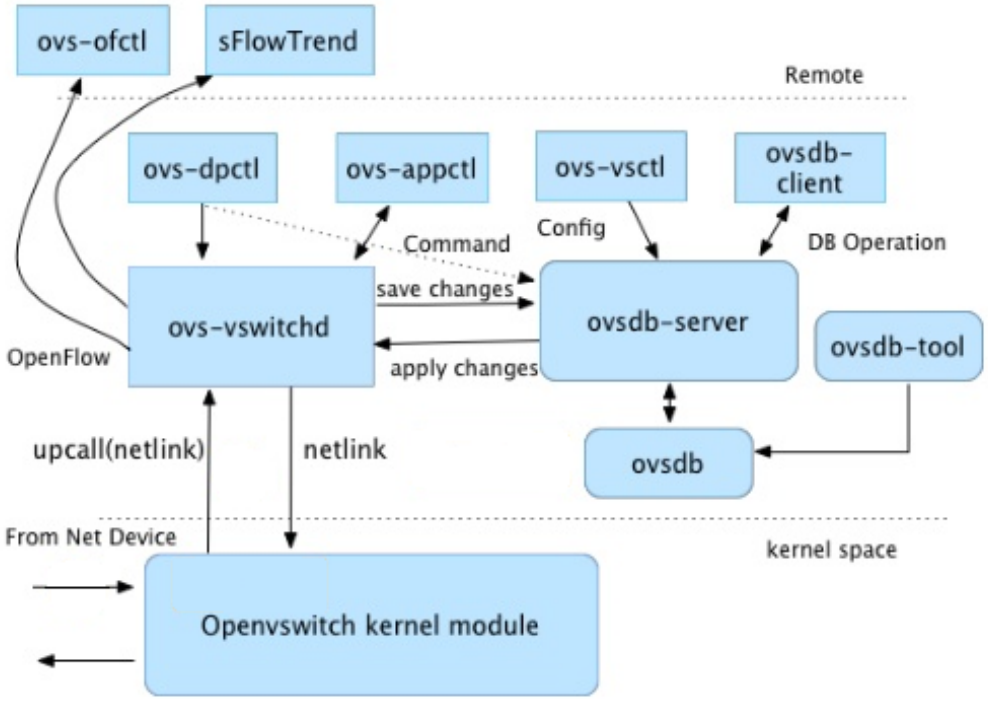

Figure 2.1: Open vSwitch Internals.

Open vSwitch può anche operare interamente dall'userspace senza assistenza da un modulo del kernel.
Questa implementazione dell'userspace dovrebbe essere più semplice da trasferire rispetto allo switch kernel-based. OVS in userspace può accedere a dispositivi Linux o DPDK.

SDN con Open vSwitch apporta numerosi vantaggi, specialmente se utilizzato insieme alle macchine virtuali:

1. Gli strati della rete possono essere identificati facilmente;

2. Le reti e il loro stato live possono essere spostati da un host ad un altro;

3. Le dinamiche di rete sono rintracciabili e il software esterno può essere abilitato per rispondere ad esse;

4. Open vSwitch implementa il protocollo GRE (Generic Routing Encapsulation). Ciò consente, ad esempio, di connettere tra loro le reti VM

private.

2.1.1 La struttura di Open vSwitch

Come è possibile vedere in Figura 2.1, i principali componenti di questa distribuzione sono:

- ovs-vswitchd: demone che implementa lo switch, insieme ad un modulo complementare del kernel Linux per la commutazione basata sul flusso;
- ovsdb-server: database server interrogato da ovs-vswitchd per ottenere la sua configurazione;
- ovs-dpctl: strumento per configurare il modulo dello switch kernel;
- ovs-vsctl: utility per interrogare e aggiornare la configurazione di ovs-vswitchd;
- ovs-appctl: utility che invia comandi a demoni Open vSwitch operativi;
- ovs-ofctl: utility per interrogare e controllare switch OpenFlow e controllers;
- ovs-pki: utility per la creazione e la gestione dell'infrastruttura a chiave pubblica.

La Figura 2.2 illustra i componenti principali di Open vSwitch e le interfacce verso un cluster di controllo e gestione. Un'istanza OVS comprende un database server (ovsdb-server), un demone vswitch (ovs-vswitchd) e, opzionalmente, un modulo che esegue il fast-path forwarding. Il "cluster di gestione e controllo" è composto da alcuni manager e controllori. I manager utilizzano il protocollo di gestione OVSDB per gestire le istanze di OVS. Un'istanza OVS è gestita da almeno un manager. I controller utilizzano OpenFlow per istanziare flow state negli switch OpenFlow. Un'istanza OVS può supportare più datapath logici, indicati come "bridges". C'è almeno un controller per ogni bridge OpenFlow. L'interfaccia di gestione OVSDB viene utilizzata per eseguire operazioni di gestione e configurazione sull'istanza OVS. Rispetto a OpenFlow, le operazioni di gestione OVSDB si verificano in un intervallo di tempo relativamente lungo. Esempi di operazioni supportate da OVSDB includono:

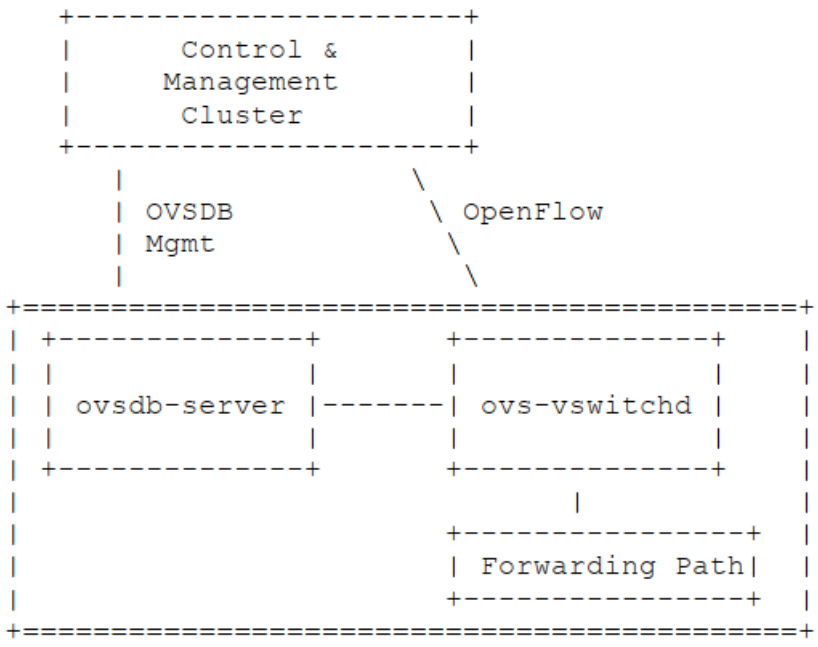

Figure 2.2: Open vSwitch Interfaces.

- Creazione, modifica ed eliminazione di datapaths OpenFlow (bridges), di cui possono esserci molti in una singola istanza OVS;
- Creazione, modifica ed eliminazione di porte di datapaths OpenFlow;
- Creazione, modifica ed eliminazione di interfacce tunnel di datapaths OpenFlow;
- Creazione, modifica ed eliminazione di queues (sono code di pacchetti assegnate ad una switch port che implementano una o più policy QoS; permettono la regolarizzazione del flusso di pacchetti);
- Configurazione di un set di controllori a cui l'OpenFlow datapath si connette;
- Configurazione di policies QoS (Quality of Service);
- Collezione di statistiche.

OVSDB non compie operazioni per-flow, lasciando quest'ultime a OpenFlow.

Chapter 3

Il Protocollo OVSDB

OVSDB, il database Open vSwitch, è un sistema di database accessibile dalla rete. Gli schemi in OVSDB specificano le tabelle in un database e i loro tipi di colonne e possono includere dati, unicità e vincoli di integrità referenziale. OVSDB offre transazioni atomiche, coerenti, isolate e durature. RFC 7047 specifica il protocollo basato su JSON-RPC utilizzato dai client OVSDB e dai server per comunicare.

Il protocollo OVSDB è adatto per la sincronizzazione dello stato perché consente a ciascun client di monitorare i contenuti di un intero database o di un sottoinsieme di esso. Ogni volta che una porzione monitorata del database cambia, il server comunica al client quali righe sono state aggiunte, modificate (inclusi i nuovi contenuti) o eliminate. Così i client OVSDB possono facilmente tenere traccia dei contenuti più recenti di qualsiasi parte del database.

Mentre OVSDB è general-purpose e non particolarmente specializzato per l'uso con Open vSwitch, Open vSwitch lo usa per molteplici scopi. L'uso principale di OVSDB è la configurazione e il monitoraggio di ovs-vswitchd, il demone dello switch Open vSwitch, che utilizza lo schema documentato in ovs-vswitchd.conf.db.

Le specifiche del protocollo OVSDB consentono di sviluppare implementazioni indipendenti e interoperabili di OVSDB. Open vSwitch include un'implementazione del server OVSDB denominata ovsdb-server, che supporta diverse estensioni di protocollo e un client OVSDB di base da linea di comando denominato ovsdb-client, nonché librerie client OVSDB per C e per Python. La documentazione di Open vSwitch spesso parla di queste implementazioni OVSDB in Open vSwitch come semplicemente "OVSDB",

anche se è distinto dal protocollo OVSDB.

Oltre a questi strumenti generici per server e client OVSDB, Open vSwitch include strumenti per lavorare con database che hanno schemi specifici: ovs-vsctl funziona con il database di configurazione ovs-vswitchd, vtep-ctl funziona con il database VTEP (*VXLAN Tunnel End Point*, tecnologia utilizzata per creare una rete che colleghi VMs e server fisici facendoli credere di trovarsi sulla stessa Layer 2 network), ovn-nbctl funziona con il database Northbound OVN (interfaccia tra la Open Virtual Network e i sistemi di management cloud) e così via.

RFC 7047 specifica il protocollo OVSDB ma non specifica un formato di archiviazione su disco. Open vSwitch include ovsdb-tool per lavorare con i propri formati di database su disco. La caratteristica più notevole di questo formato è che ovsdb-tool rende facile per gli utenti stampare le transazioni che sono cambiate in un database dall'ultima volta che è stato compattato. Questa funzione è spesso utile per la risoluzione dei problemi.

3.1 Modelli di servizio

OVSDB supporta tre modelli di servizio per i database: standalone, backup attivo e cluster. I modelli di servizio offrono diversi compromessi tra coerenza, disponibilità e tolleranza delle partizioni. Differiscono anche nel numero di server richiesti e in termini di prestazioni. I modelli di servizio di database standalone e di backup attivo condividono un formato su disco e i database in cluster utilizzano un formato diverso, ma i programmi OVSDB funzionano con entrambi i formati.

RFC 7047, che specifica il protocollo OVSDB, non richiede o specifica alcun particolare modello di servizio.

Vediamo ora di analizzare il caso in cui si usi un solo database.

3.1.1 Modello di servizio a database autonomo

Un database standalone esegue un singolo server. Se il server smette di funzionare, il database diventa inaccessibile e, se la memoria del server viene persa o danneggiata, il contenuto del database viene perso. Questo mod-

ello di servizio è appropriato quando il database controlla un processo o un'attività a cui è collegato tramite "fate-sharing". Ad esempio, un'istanza OVSDB che controlla un demone di uno switch virtuale Open vSwitch (ovs-vswitchd) è un database autonomo perché un guasto del server eliminerebbe sia il database che lo switch virtuale.

Per impostare un database standalone si utilizza il comando *ovsdb-tool create* per creare un file di database, quindi si esegue il comando *ovsdb-server* per avviare il servizio del database.

Per configurare un client, come ovs-vswitchd o ovs-vsctl, per utilizzare un database standalone, si configura il server in modo che ascolti un "metodo di connessione" che il client può raggiungere, quindi si indirizza il client al metodo di connessione.

3.2 Metodo di connessione

Un metodo di connessione OVSDB è una stringa che specifica come eseguire una connessione JSON-RPC tra un client OVSDB e un server. I metodi di connessione fanno parte dell'implementazione Open vSwitch di OVSDB e non sono specificati da RFC 7047. Ovsdb-server utilizza metodi di connessione per specificare come dovrebbe ascoltare le connessioni dai client e ovsdb-client li utilizza per specificare come deve connettersi a un server. Sono anche possibili connessioni nella direzione opposta, in cui ovsdb-server si connette a un client configurato per l'ascolto di una connessione in ingresso.

I metodi di connessione sono classificati come attivi o passivi. Un metodo di connessione attivo effettua una connessione in uscita a un host remoto; un metodo di connessione passivo ascolta le connessioni da host remoti. La soluzione più comune consiste nel configurare un server OVSDB con metodi di connessione passivi e client con quelli attivi, ma l'implementazione OVSDB in Open vSwitch supporta anche la disposizione opposta.

OVSDB supporta i seguenti metodi di connessione attivi:

- ssl: <host>: <porta>
  La <porta> SSL o TLS specificata sul dato <host>.

- tcp: <host>: <porta>
  La <porta> TCP specificata sul dato <host>.

- unix: <file>
  Sui sistemi di tipo Unix, si connette al socket del domain server Unix denominato <file>. Su Windows, si connette a una local named pipe che è rappresentata da un file creato nel percorso <file> per imitare il comportamento di un socket di dominio Unix.

- <Method1>,<method2>,...,<methodN>
  Affinché un servizio di database cluster sia altamente disponibile, un client deve essere in grado di connettersi a qualsiasi server del cluster. Per fare ciò, si specifica i metodi di connessione per ciascuno dei server separati da virgole (e spazi facoltativi).

OVSDB supporta i seguenti metodi di connessione passiva:

- pssl: <porta>: [:<IP>]
  Ascolta sulla <porta> TCP per le connessioni SSL o TLS. Per impostazione predefinita, le connessioni non sono collegate a un particolare indirizzo IP locale. Specificando <ip> limita le connessioni a quelle dell'indirizzo IP specificato.

- ptcp: <porta>: [:<IP>]
  Ascolta sulla <porta> TCP. Per impostazione predefinita, le connessioni non sono collegate a un particolare indirizzo IP locale. Specificando <ip> si limita le connessioni a quelle dell'indirizzo IP specificato.

- punix: <file>
  Sui sistemi di tipo Unix, ascolta per connessioni sul socket del dominio Unix denominato <file>. Su Windows, ascolta su una nominata pipe locale, creando una pipe chiamata <file> per imitare il comportamento di un socket di dominio Unix.

Tutti i metodi di connessione basati su IP accettano indirizzi IPv4 e IPv6. Per specificare un indirizzo IPv6, bisogna racchiuderlo tra parentesi quadre, ad es. ssl: [::1]: 6640. I metodi di connessione passivi basati su IP per impostazione predefinita ascoltano solo le connessioni IPv4; è necessario usare [::] come indirizzo per accettare entrambe le connessioni IPv4 e IPv6, ad es.

PSSL: 6640: [::].

<porta> può essere omesso dai metodi di connessione che usano un numero di porta. La <porta> predefinita per i metodi di connessione basati su TCP è 6640. In versioni Open vSwitch precedenti alla 2.4.0, la porta predefinita è la 6632.

I metodi di connessione ssl e pssl richiedono un'ulteriore configurazione tramite le opzioni della riga di comando –private-key, –certificate e –ca-cert. Open vSwitch può essere creato senza il supporto SSL: in quel caso questi metodi di connessione non sono supportati.

3.3 Life cycle del database

3.3.1 Creare un database

A seconda del modello di servizio utilizzato, cambia il modo per creare e accendere un servizio per il database. Per quanto ci riguarda, essendoci interessati solo del caso "a database autonomo", possiamo utilizzare il comando *ovsdb-tool create* per generare un file di database.

3.3.2 Fare il backup e ripristinare un database

OVSDB viene spesso utilizzato in contesti in cui i contenuti del database non sono particolarmente preziosi. Ad esempio, in molti sistemi, il database per la configurazione di ovs-vswitchd viene essenzialmente ricostruito da zero al momento dell'avvio. Non vale la pena eseguire il backup di questi database.

Quando OVSDB viene utilizzato per dati preziosi, è consigliabile prendere in considerazione una strategia di backup. Un modo è utilizzare la replica del database, che conserva una copia online e aggiornata di un database, possibilmente su un sistema remoto. Ciò funziona con tutti i modelli di servizio OVSDB.

Una strategia di backup più comune consiste nel prendere e memorizzare periodicamente un'istantanea. Per i modelli di servizio standalone e di backup attivo, eseguire una copia del file di database, ad es. usando il comando cp, crea effettivamente un'istantanea e poiché i file di database

OVSDB sono append-only, funziona anche se il database viene modificato quando si verifica l'istantanea. Questo approccio non funziona per i database in cluster.

Un altro modo per eseguire un backup, che funziona con tutti i modelli di servizio OVSDB, consiste nell'utilizzare *ovsdb-client backup*, che si collega a un database server in esecuzione e genera un'istantanea atomica del suo schema e contenuto, nello stesso formato utilizzato per i database standalone e active-backup.

Sono disponibili anche più opzioni quando arriva il momento di ripristinare un database da un backup. Per i modelli di servizio standalone e di active-backup, un'opzione è quella di arrestare il server (o i server) del database, sovrascrivere il file del database con il backup (ad es. con il comando cp) e quindi riavviare i server. Un altro modo, che funziona con qualsiasi modello di servizio, consiste nell'utilizzare *ovsdb-client restore*, che permette di collegarsi a un server di database in esecuzione e sostituire i dati in uno dei suoi database con uno snapshot fornito. Il vantaggio di *ovsdb-client restore* è che causa zero tempi di inattività per il database e il suo server. Ha lo svantaggio che gli UUID delle righe nel database ripristinato differiscono da quelli nello snapshot, poiché il protocollo OVSDB non consente ai client di specificare la righe degli UUID.

Nessuno di questi approcci salva e ripristina i dati in colonne che lo schema indica come effimere. Questo è in base alla progettazione: il progettista di uno schema contrassegna una colonna come effimera solo se è accettabile che i dati vengano persi al riavvio di un server di database.

Il clustering e il backup hanno scopi diversi. Il clustering aumenta la disponibilità, ma non protegge dalla perdita di dati se, ad esempio, un client OVSDB dannoso o malfunzionante elimina o altera i dati.

3.3.3 Cambiare il modello di servizio del database

Si può utilizzare il comando *ovsdb-tool create-cluster* per creare un database clustered dai contenuti di un database standalone e il comando *ovsdb-tool backup* per creare un database autonomo dal contenuto di un database cluster.

3.3.4 Aggiornare o declassare un database

L'evoluzione di un software può richiedere modifiche agli schemi dei database che utilizza. Ad esempio, nuove funzionalità potrebbero richiedere nuove tabelle o nuove colonne nelle tabelle esistenti, oppure modifiche concettuali potrebbero richiedere la riorganizzazione di un database in altri modi. In alcuni casi, il modo più semplice per gestire una modifica in uno schema di database consiste nell'eliminare il database esistente e ricominciare con il nuovo schema, soprattutto se i dati nel database sono facili da ricostruire. In molti altri casi, è meglio convertire il database da uno schema a un altro.

L'implementazione OVSDB in Open vSwitch ha il supporto integrato per alcuni semplici casi di conversione di un database da uno schema a un altro. Questo supporto può gestire le modifiche che aggiungono o rimuovono colonne/tabelle del database (ad esempio permette di cambiare una colonna che deve avere esattamente un valore con una che ha uno o più valori). Può anche gestire modifiche che aggiungono vincoli o renderli più rigidi, ma solo se i dati esistenti nel database soddisfano i nuovi vincoli. Ad esempio, è possibile cambiare una colonna che ha uno o più valori in una con esattamente un valore, se ogni riga nella colonna ha esattamente un valore. La conversione integrata può causare la perdita di dati in modi ovvi (se il nuovo schema rimuove le tabelle o le colonne) oppure in modo indiretto (per esempio eliminando le righe non referenziate nelle tabelle contrassegnate dal nuovo schema per garbage collection. E' quindi consigliabile fare un backup in anticipo.

Per utilizzare il supporto integrato di OVSDB per la conversione dello schema con un database autonomo o di active-backup, bisogna arrestare prima il server (o i server) del database, quindi utilizzare il comando *ovsdb-tool convert* per convertirlo nel nuovo schema e infine riavviare il server del database.

OVSDB supporta anche la conversione dello schema del database online per qualsiasi modello di servizio del database. Per convertire un database online si utilizza il comando *ovsdb-client convert*. La conversione è atomica, coerente, isolata e duratura. Ovsdb-server disconnette tutti i client connessi quando la conversione ha luogo (eccetto i client che usano l'estensione Open vSwitch set_db_change_aware). Al momento della riconnessione, i client scopriranno che lo schema è cambiato.

Le versioni e i checksum dello schema (vedere gli Schemas) possono fornire suggerimenti su come convertire un database in un nuovo schema.

3.4 RPC method

Con il termine *RPC method* è indicata una chiamata di procedura remota (*Remote Procedure Call*) che si compone, come descritto nelle specifiche *JSON-RPC 1.0*, in una richiesta e in una risposta. La richiesta è formata da una stringa che contiene il nome del metodo invocato, un array di parametri da passare al metodo e un *request ID* che può essere usato per abbinare la risposta alla richiesta. La risposta, invece, è formata da un *result object* (non nullo nel caso che l'invocazione del metodo abbia successo), da un *error object* (non nullo nel caso vi sia un errore) e l'*ID* della richiesta abbinata.

Mentre un server OVSDB **deve** implementare tutti i metodi che andremo ad analizzare qui sotto, un client OVSDB **deve** implementare il metodo "*Echo*" mentre è libero di implementare qualunque metodo a seconda delle necessità.

I metodi usati sono:

- **List Databases**: recupera un array i cui elementi sono i nomi dei databases a cui è possibile fare accesso tramite *OVSDB*;
- **Get Schema**: recupera un <*database-schema*> che descrive il database ospitato <*db-name*>;
- **Transact**: permette di eseguire su un certo database una serie di operazioni nell'ordine da noi specificato;
- **Cancel**: il seguente metodo è una notifica *JSON-RPC*, non richiede quindi una risposta. Istruisce il database server a completare o cancellare la richiesta "*transact*" il cui "*id*" coincide con il valore "*params*" della notifica *JSON-RPC*;
- **Monitor**: abilita un client a replicare delle tables o dei sottoinsiemi di tables contenute in un database *OVSDB*. Per fare ciò richiede notifica di avvenute modifiche di tali tables e riceve lo stato iniziale di una table o di un suo sottoinsieme;
- **Update Notification**: inviato da un server ad un client per notificare una modifica in alcune tables che sono monitorate a seguito di una richiesta "*monitor*";

- **Monitor Cancellation**: cancella una precedente richiesta di "*monitor*";
- **Lock Operations**: sono presenti tre metodi *RPC*, "*lock*", "*steal*" e "*unlock*" che provvedono ad eseguire operazioni di *locking* su un database. Il database server supporta un arbitrario numero di *locks*, ognuno identificato da un ben definito client-ID. Il preciso uso di un *lock* è determinato dal client: per esempio un certo set di clients potrebbero concordare che una tabella sia scrivibile solo dal proprietario di un certo *lock* (OVSDB stesso non impone alcuna restrizione su come i *locks* devono essere usati; si assicura semplicemente che un *lock* abbia al massimo un solo proprietario);
- **Locked Notifications**: usato per notificare a un client che gli è stato concesso un *lock* precedentemente richiesto con un metodo "*lock*";
- **Stolen Notification**: usato per notificare ad un client che la proprietà di un suo *lock*, precedentemente ottenuto, è stato "rubato" da un altro client;
- **Echo**: usato sia dai clients che dai servers per verificare se una connessione ad un database è ancora attiva. E' necessario che questo metodo sia implementato sia dai clients che dai servers.

Come sarà possibile vedere nel capitolo successivo, il metodo *transact* è quello più usato in quanto permette di svolgere tutta una serie di operazioni sul database. Ognuna delle operazioni è un oggetto JSON che può essere incluso nell'array "*params*" (è uno degli elementi presenti nella richiesta "*transact*" discussa qui sopra).

Chapter 4

Utilizzo del protocollo OVSDB

4.1 Testbed e configurazione iniziale

Le prove sperimentali sul protocollo sono state svolte nel Laboratorio 3.2 (Lele) in cui sono presenti sedici postazioni di lavoro equipaggiate con workstation Linux CentOS 7.2 dotate di interfacce di rete multiple e di numerosi pacchetti software open-source relativi a servizi avanzati di rete, che spaziano dai protocolli di instradamento del traffico in reti complesse, ai principali servizi applicativi più diffusi in Internet, quali server web, VoIP, ecc. Le postazioni sono ideali per lo studente che vuole configurare, studiare e sperimentare servizi di rete di base e avanzati su un ambiente Linux in piena autonomia. Il laboratorio è dotato anche di apparati commerciali per reti di telecomunicazioni, quali router IP Cisco e switch Ethernet 3Com.

Nel nostro caso specifico sono state usate due macchine (un client e un server) con indirizzo IP della porta eth0 rispettivamente 192.168.8.15/24 (netlab15) e 192.168.8.16/24 (netlab16), mentre per la porta eth1 10.0.0.29/30 (netlab15) e 10.0.0.30/30 (netlab16). Sulle macchine era già installata una versione di Open vSwitch (versione 2.6.1).

Dopo aver avviato le macchine, si è proceduto ad abilitare su entrambe il software di gestione di Open vSwitch tramite i seguenti comandi (da eseguire con i diritti di amministratore inserendo *sudo* all'inizio):

systemctl start openvswitch → avvio di Open vSwitch
systemctl status openvswitch → visualizza status Open vSwitch
ovs-vsctl show → versione corrente Open vSwitch

```
[luca.falco@netlab15 ~]$ sudo systemctl status openvswitch
● openvswitch.service - Open vSwitch
   Loaded: loaded (/usr/lib/systemd/system/openvswitch.service; disabled; vendor
 preset: disabled)
   Active: active (exited) since Wed 2019-02-13 10:41:43 CET; 1 weeks 4 days ago
  Process: 3978 ExecStart=/bin/true (code=exited, status=0/SUCCESS)
 Main PID: 3978 (code=exited, status=0/SUCCESS)
   CGroup: /system.slice/openvswitch.service

Feb 13 10:41:43 netlab15 systemd[1]: Starting Open vSwitch...
Feb 13 10:41:43 netlab15 systemd[1]: Started Open vSwitch.
Feb 14 08:46:58 netlab15 systemd[1]: Started Open vSwitch.
```

Figure 4.1: Open vSwitch avviato.

Se i passaggi precedenti sono stati eseguiti in modo corretto e il sistema non ha generato errori, otterremo i seguenti messaggi (Figura 4.1 e Figura 4.2).

È necessario, in seguito, controllare che sia stato avviato anche il processo *ovsdb-server* tramite i comandi *systemctl start ovsdb-server* e *systemctl status ovsdb-server*(Figura 4.1).

Questi passaggi vengono compiuti su entrambe le macchine. Come già visto nei capitoli precedenti, Open vSwitch provvede di default a creare il file di database sui computer all'indirizzo */ovs/var/run/openvswitch/db.sock* (nel nostro caso il file conf.db era presente in */etc/openvswitch*). Allo stesso modo OVS genera il PID (*Process IDentifier*, permette di identificare in modo univoco un processo) di ovsdb-server e ovs-vswitchd (il PID è presente sulle macchine all'indirizzo */var/run/openvswitch*).

Occorre successivamente impostare il server in modo tale che accetti i comandi che riceverà dal client. Come visto nel capitolo "*Metodi di connessione*", il server e il client possono connettersi tra loro con una connessione di tipo attivo (il server "ascolta" sulla porta 6640 e aspetta che il client si connetta) o passivo (il contrario). In questo caso si è usato un metodo di connessione attivo: per impostarlo si è digitato nel server il comando *sudo ovs-vsctl set-manager ptcp:192.168.8.15:6640*.

Successivamente si è controllata la buona riuscita dell'operazione con *sudo ovs-vsctl show*(Figura 4.2).

```
[luca.falco@netlab16 ~]$ sudo ovs-vsctl show
92f648e0-77c4-4351-a3a8-2686ca2f9bc2
    Manager "ptcp:6640"
```

Figure 4.2: Manager impostato correttamente.

```
Bridge table
_uuid                                auto_attach controller datapath_id       d
atapath_type datapath_version external_ids fail_mode flood_vlans flow_tables ipf
ix mcast_snooping_enable mirrors name   netflow other_config ports
                    protocols rstp_enable rstp_status sflow status stp_enable
------------------------------------ ----------- ---------- ------------------ -
------------ ---------------- ------------ --------- ----------- ----------- ---
-- --------------------- ------- ------ ------- ------------ --------------------
------------------- --------- ----------- ----------- ----- ------ ----------
c4395e6b-47cf-457c-8be3-786848d81052 []          []         "00006a5e39c47c45" "
"            "<unknown>"      {}          []         []          {}          []
   false                 []      "br54" []      {}          [516fc265-43fb-466a
-98a0-d588872a3d99] []        false       {}          []    {}     false
flea29d9-3938-461b-be0b-7c781cb20365 []          []         "0000da29eaf11b46" "
"            "<unknown>"      {}          []         []          {}          []
   false                 []      "br89" []      {}          [c2b29779-ad40-4f07
-bd8a-dfc4ba2d96b9] []        false       {}          []    {}     false
```

Figure 4.3: Tabella Bridge.

4.2 Prove svolte

Giunti a questo punto (fino ad ora tutti i comandi sono stati inseriti in locale nelle due macchine) è finalmente possibile interagire da remoto sul server: per comunicare con esso viene usato *ovsdb-client* che è un client da linea di comando per interagire con il processo ovsdb-server. Per esempio, usando il comando ovsdb-client dump tcp:192.168.8.16:6640 - -pretty è possibile stampare su schermo le tabelle OVSDB presenti sul server (- *-pretty* permette di ottenere una visualizzazione più leggibile). Un esempio di visualizzazione del codice ricevuto sul client è visibile nella Figura 4.3 e nella Figura 4.4.

Andiamo ora ad analizzare nel dettaglio come sia possibile aggiungere o rimuovere bridge, porte e interfacce da remoto con il comando *sudo ovs-vsctl –db=tcp:192.168.8.16:6640*. Questo comando utilizza OVS. Come visto nel capitolo "*Differenza tra Open vSwitch e OpenFlow*", OVS già implementa al suo interno il protocollo OVSDB. Per avere un'ulteriore conferma di questa caratteristica, si è provveduto ad analizzare i pacchetti dati inviati tra client e server dopo l'invio del comando. I due programmi usati per lo scopo sono stati Wireshark e tcpdump: il primo è un sofware che cattura e presenta i

```
Port table
_uuid                                bond_active_slave bond_downdelay bond_fake_
iface bond_mode bond_updelay external_ids fake_bridge interfaces
            lacp mac name   other_config qos rstp_statistics rstp_status statis
tics status tag trunks vlan_mode
------------------------------------ ----------------- -------------- ----------
----- --------- ------------ ------------ ----------- --------------------------
------------ ---- --- ------ ------------ --- --------------- ----------- ------
---- ------ --- ------ ---------
516fc265-43fb-466a-98a0-d588872a3d99 []                 0              false
      []        0            {}           false       [49bcd537-09e6-43f0-9ffa-1
63385fc9e12] []   []  "br54" {}           []  {}              {}          {}
     {}     []  []     []
c2b29779-ad40-4f07-bd8a-dfc4ba2d96b9 []                 0              false
      []        0            {}           false       [37214c64-8655-42f5-8760-d
589fb292fd7] []   []  "br89" {}           []  {}              {}          {}
     {}     []  []     []
```

Figure 4.4: Tabella Port.

dati che attraversano una determinata rete in modo leggibile, mentre il secondo svolge una funzione simile a Wireshark, ma è possibile usarlo tramite linea di comando.

Per prima cosa abbiamo avviato tcpdump sul server tramite comando *sudo tcpdump -s 0 -i eth1 -w tcpdump.pcap* (con -s si può impostare la massima lunghezza in bytes di ogni pacchetto, con -i si seleziona la porta da monitorare e con -w si abilita la compatibilità del file con Wireshark). A questo punto è stato inviato dal client il comando per generare un nuovo bridge di nome *br89* sul server (*sudo ovs-vsctl - -db=tcp:192.168.8.16:6640 add-br br89*). Fatta la cattura, si è proceduto a trasferire il file tcpdump.pcap su un computer con installato Wireshark (per fare ciò si è dovuto prima dare il permesso di copia al file tramite comando *sudo chmod 644 tcpdump.pcap*). Una volta importato il file su Wireshark è stato possibile visualizzare il contenuto del file: nella Figura 4.5 è visibile lo scambio di pacchetti TCP tra le due macchine.

Selezionando la trasmissione che si è svolta (dal pacchetto 28 al pacchetto 54) e visualizzandola in formato ASCII, se è potuto constatare come il comando appartenente al protocollo OVS ha inviato effettivamente un JSON-RPC appartenente al protocollo OVSDB (visibile in Figura 4.6). Dopo una richiesta iniziale da parte del client di ottenere lo schema del database del server, attraverso il codice

```
"id":0,"method":"get_schema","params":["Open_vSwitch"]
```

i messaggi inviati dal client sono due.

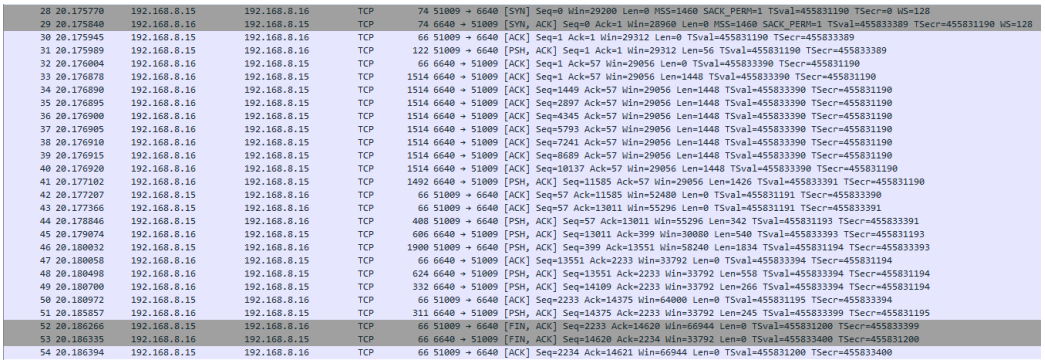

Figure 4.5: Cattura Wireshark.

```
{
  "id": 1,
  "method": "monitor_cond",
  "params": [
    "Open_vSwitch",
    "01528a65-2e42-4cbc-bdc8-8bd569a8de28",
    {
      "Port": {
        "columns": [
          "fake_bridge",
          "interfaces",
          "name",
          "tag"
        ]
      },

    [...]

    }
  ]
}
```

Nel primo (presente qui sopra) si vede una richiesta di "*monitor_cond*": questo metodo non è presente nella documentazione OVSDB poichè implementata in Open vSwitch dalla versione 2.6. Corrisponde, con poche differenze, al metodo "*monitor*" già visto in precedenza. E' formato da tre membri:

- "*method*": "monitor";
- "*params*": [<db-name>,<json-value>,<monitor-requests>];
- "*id*": <nonnull-json-value>.


d588872a3d99"]}}},"Open_vSwitch":{"92f648e0-77c4-4351-a3a8-2686ca2f9bc2":{"initial":{"bridges":["uuid","c4395e6b-47cf-457c-8be3-786848d81052"],"cur_cfg":58}}}},"error":null}
{"id":2,"method":"transact","params":["Open_vSwitch",{"rows":[{"bridges":["uuid","c4395e6b-47cf-457c-8be3-786848d81052"]}],"until":"==","where":[["_uuid","==",
["uuid","92f648e0-77c4-4351-a3a8-2686ca2f9bc2"]]],"columns":["bridges"],"timeout":0,"op":"wait","table":"Open_vSwitch"},{"rows":[{"interfaces":
["uuid","49bcd537-09e6-43f0-9ffa-163385fc9e12"]]],"until":"==","where":[["_uuid","==",["uuid","516fc265-43fb-466a-98a0-d588872a3d99"]]],"columns":["interfaces"],"timeout":
0,"op":"wait","table":"Port"},{"rows":[{"ports":["uuid","516fc265-43fb-466a-98a0-d588872a3d99"]}],"until":"==","where":[["_uuid","==",
["uuid","c4395e6b-47cf-457c-8be3-786848d81052"]]],"columns":["ports"],"timeout":0,"op":"wait","table":"Bridge"},{"uuid-name":"row42ef0a14_7f8a_4bba_a3e6_5493e06906d2","row":
{"name":"br89","type":"internal"},"op":"insert","table":"Interface"},{"where":[["_uuid","==",["uuid","92f648e0-77c4-4351-a3a8-2686ca2f9bc2"]]],"row":{"bridges":["set",
[["named-uuid","row36dda593_2dd8_4f23_b2b1_cd6ec80b35b4"],["uuid","c4395e6b-47cf-457c-8be3-786848d81052"]]]},"op":"update","table":"Open_vSwitch"},{"uuid-
name":"rowd81dfdba_0d53_4b96_92f5_f006123b2942","row":{"name":"br89","interfaces":["named-uuid","row42ef0a14_7f8a_4bba_a3e6_5493e06906d2"]},"op":"insert","table":"Port"},
{"uuid-name":"row36dda593_2dd8_4f23_b2b1_cd6ec80b35b4","row":{"name":"br89","ports":["named-
uuid","rowd81dfdba_0d53_4b96_92f5_f006123b2942"]},"op":"insert","table":"Bridge"},{"mutations":[["next_cfg","+=",1]],"where":[["_uuid","==",["uuid","92f648e0-77c4-4351-
a3a8-2686ca2f9bc2"]]],"op":"mutate","table":"Open_vSwitch"},{"where":[["_uuid","==",["uuid","92f648e0-77c4-4351-a3a8-2686ca2f9bc2"]]],"columns":
["next_cfg"],"op":"select","table":"Open_vSwitch"},{"comment":"ovs-vsctl (invoked by sudo): ovs-vsctl --db=tcp:192.168.8.16:6640 add-br br89","op":"comment"}]}

Figure 4.6: Esempio codice JSON-RPC.

Nel secondo messaggio (Figura 4.6) il client richiama il metodo RPC *transact*; come documentato nel capitolo "*RPC method*", *transact* è usato per eseguire una serie di operazioni nell'ordine specificato su un database. In questo caso viene creato un bridge (nominato br89) a cui viene aggiunta, di default, anche una porta e una interfaccia. Qui sotto viene riportato una parte del codice (nell'Appendice A.1 è presente il codice per intero).

```
{
  "id": 2,
  "method": "transact",
  "params": [
    "Open_vSwitch",
    {
      "rows": [
        {
          "bridges": [
            "uuid",
            "c4395e6b-47cf-457c-8be3-786848d81052"
          ]
        }
      ],

    [...]
```

Altri metodi che possiamo trovare nel messaggio inviato sono "*wait*", "*insert*", "*update*" e "*mutate*". Il metodo "wait" aspetta che vengano fatte le opportune modifiche al database, "update" e "mutate" permettono di modificare delle righe in una table (in questo caso "Open_vSwitch") e "insert" crea nuove righe nella table selezionata (nel codice agisce sulle tables "Bridge", "Port" e "Interface").

Vediamo qui sotto come compaiono nel codice:

Wait:

```
{
"until": "==",
      "where": [
        [
          "_uuid",
          "==",
          [
            "uuid",
            "92f648e0-77c4-4351-a3a8-2686ca2f9bc2"
          ]
        ]
      ],
      "columns": [
        "bridges"
      ],
      "timeout": 0,
      "op": "wait",
      "table": "Open_vSwitch"
    },

    [...]
```

Update

```
"row": {
        "bridges": [
          "set",
          [
            [
              "named-uuid",
              "row36dda593_2dd8_4f23_b2b1_cd6ec80b35b4"
            ],
            [
              "uuid",
              "c4395e6b-47cf-457c-8be3-786848d81052"
            ]
          ]
        ]
      },
      "op": "update",
      "table": "Open_vSwitch"
    },

    [...]
```

Mutate

```
{
      "mutations": [
        [
          "next_cfg",
          "+=",
          1
        ]
      ],
      "where": [
        [
          "_uuid",
          "==",
          [
            "uuid",
            "92f648e0-77c4-4351-a3a8-2686ca2f9bc2"
          ]
        ]
      ],
      "op": "mutate",
      "table": "Open_vSwitch"
    },

    [...]
```

Insert

```
{
      "uuid-name": "row36dda593_2dd8_4f23_b2b1_cd6ec80b35b4",
      "row": {
        "name": "br89",
        "ports": [
          "named-uuid",
          "rowd81dfdba_0d53_4b96_92f5_f006123b2942"
        ]
      },
      "op": "insert",
      "table": "Bridge"
    },

    [...]
```

In Appendice A.2 è presentato il codice inerente la creazione di una port su un bridge: anche in questo caso è stato inviato un messaggio dal client (riportato qui sotto), catturato sul server con tcpdump e poi visualizzato con l'uso del software Wireshark.

```
sudo ovs-vsctl --db=tcp:192.168.8.16:6640 add-port br89 eth0
```

E' anche possibile inviare messaggi dal client al server attraverso l'uso del comando ovsdb-client. Come visto in precedenza, facciamo uso di un client da linea di comando che interagisce con ovsdb-server. Come al solito viene usato il metodo "*transact*"; il codice da digitare è il seguente: *ovsdb-client -v transact tcp:192.168.8.16:6640 '["Open_vSwitch",qui è inserito il codice JSON*]'. Con questo metodo è quindi possibile comporre messaggi specifici in JSON; vista la difficoltà nell'editare tali messaggi da linea di comando, è consigliato l'uso di editor di testo.

Chapter 5

Conclusioni

L'obiettivo di questa tesi era analizzare il protocollo OVSDB e come esso ci permetta di interagire con implementazioni Open vSwitch presenti su un server attraverso appositi comandi.

La possibilità di interagire da remoto e di poter automatizzare il processo di configurazione e modifica della rete aprono la strada a soluzioni che ben si sposano con le necessità moderne. Va però detto che la difficoltà nel comporre i messaggi JSON-RPC è relativamente alta e sicuramente risulta più facile usare i comandi nativi di Open vSwitch che invece già implementano OVSDB.

Inoltre, poichè OVSDB si basa sull'architettura OVS, ogni dispositivo deve essenzialmente avere le stesse caratteristiche o interfacce interne di uno switch virtuale e ciò comporta una limitazione sulla classe di dispositivi su cui è possibile lavorare.

Appendix A

Codice dei messaggi JSON

A.1 Creazione di un bridge

```
{
  "id": 2,
  "method": "transact",
  "params": [
    "Open_vSwitch",
    {
      "rows": [
        {
          "bridges": [
            "uuid",
            "c4395e6b-47cf-457c-8be3-786848d81052"
          ]
        }
      ],
      "until": "==",
      "where": [
        [
          "_uuid",
          "==",
          [
            "uuid",
            "92f648e0-77c4-4351-a3a8-2686ca2f9bc2"
          ]
        ]
      ],
      "columns": [
        "bridges"
      ],
      "timeout": 0,
      "op": "wait",
      "table": "Open_vSwitch"
```

```
},
{
  "rows": [
    {
      "interfaces": [
        "uuid",
        "49bcd537-09e6-43f0-9ffa-163385fc9e12"
      ]
    }
  ],
  "until": "==",
  "where": [
    [
      "_uuid",
      "==",
      [
        "uuid",
        "516fc265-43fb-466a-98a0-d588872a3d99"
      ]
    ]
  ],
  "columns": [
    "interfaces"
  ],
  "timeout": 0,
  "op": "wait",
  "table": "Port"
},
{
  "rows": [
    {
      "ports": [
        "uuid",
        "516fc265-43fb-466a-98a0-d588872a3d99"
      ]
    }
  ],
  "until": "==",
  "where": [
    [
      "_uuid",
      "==",
      [
        "uuid",
        "c4395e6b-47cf-457c-8be3-786848d81052"
      ]
    ]
  ],
  "columns": [
```

```
      "ports"
    ],
    "timeout": 0,
    "op": "wait",
    "table": "Bridge"
  },
  {
    "uuid-name": "row42ef0a14_7f8a_4bba_a3e6_5493e06906d2",
    "row": {
      "name": "br89",
      "type": "internal"
    },
    "op": "insert",
    "table": "Interface"
  },
  {
    "where": [
      [
        "_uuid",
        "==",
        [
          "uuid",
          "92f648e0-77c4-4351-a3a8-2686ca2f9bc2"
        ]
      ]
    ],
    "row": {
      "bridges": [
        "set",
        [
          [
            "named-uuid",
            "row36dda593_2dd8_4f23_b2b1_cd6ec80b35b4"
          ],
          [
            "uuid",
            "c4395e6b-47cf-457c-8be3-786848d81052"
          ]
        ]
      ]
    },
    "op": "update",
    "table": "Open_vSwitch"
  },
  {
    "uuid-name": "rowd81dfdba_0d53_4b96_92f5_f006123b2942",
    "row": {
      "name": "br89",
      "interfaces": [
```

```
        "named-uuid",
        "row42ef0a14_7f8a_4bba_a3e6_5493e06906d2"
      ]
    },
    "op": "insert",
    "table": "Port"
  },
  {
    "uuid-name": "row36dda593_2dd8_4f23_b2b1_cd6ec80b35b4",
    "row": {
      "name": "br89",
      "ports": [
        "named-uuid",
        "rowd81dfdba_0d53_4b96_92f5_f006123b2942"
      ]
    },
    "op": "insert",
    "table": "Bridge"
  },
  {
    "mutations": [
      [
        "next_cfg",
        "+=",
        1
      ]
    ],
    "where": [
      [
        "_uuid",
        "==",
        [
          "uuid",
          "92f648e0-77c4-4351-a3a8-2686ca2f9bc2"
        ]
      ]
    ],
    "op": "mutate",
    "table": "Open_vSwitch"
  },
  {
    "where": [
      [
        "_uuid",
        "==",
        [
          "uuid",
          "92f648e0-77c4-4351-a3a8-2686ca2f9bc2"
        ]
```

```
        ]
      ],
      "columns": [
        "next_cfg"
      ],
      "op": "select",
      "table": "Open_vSwitch"
    },
    {
      "comment": "ovs-vsctl (invoked by sudo):
            ovs-vsctl --db=tcp:192.168.8.16:6640 add-br br89",
      "op": "comment"
    }
  ]
}
```

A.2 Creazione di una porta su un bridge


```
{
  "id": 2,
  "method": "transact",
  "params": [
    "Open_vSwitch",
    {
      "rows": [
        {
          "bridges": [
            "set",
            [
              [
                "uuid",
                "c4395e6b-47cf-457c-8be3-786848d81052"
              ],
              [
                "uuid",
                "f1ea29d9-3938-461b-be0b-7c781cb20365"
              ]
            ]
          ]
        }
      ],
      "until": "==",
      "where": [
        [
          "_uuid",
          "==",
          [
            "uuid",
            "92f648e0-77c4-4351-a3a8-2686ca2f9bc2"
          ]
        ]
      ],
      "columns": [
        "bridges"
      ],
      "timeout": 0,
      "op": "wait",
      "table": "Open_vSwitch"
    },
    {
      "rows": [
        {
          "interfaces": [
            "uuid",
            "37214c64-8655-42f5-8760-d589fb292fd7"
```

```
        ]
      }
    ],
    "until": "==",
    "where": [
      [
        "_uuid",
        "==",
        [
          "uuid",
          "c2b29779-ad40-4f07-bd8a-dfc4ba2d96b9"
        ]
      ]
    ],
    "columns": [
      "interfaces"
    ],
    "timeout": 0,
    "op": "wait",
    "table": "Port"
  },
  {
    "rows": [
      {
        "interfaces": [
          "uuid",
          "49bcd537-09e6-43f0-9ffa-163385fc9e12"
        ]
      }
    ],
    "until": "==",
    "where": [
      [
        "_uuid",
        "==",
        [
          "uuid",
          "516fc265-43fb-466a-98a0-d588872a3d99"
        ]
      ]
    ],
    "columns": [
      "interfaces"
    ],
    "timeout": 0,
    "op": "wait",
    "table": "Port"
  },
  {
```

```
    "rows": [
      {
        "ports": [
          "uuid",
          "c2b29779-ad40-4f07-bd8a-dfc4ba2d96b9"
        ]
      }
    ],
    "until": "==",
    "where": [
      [
        "_uuid",
        "==",
        [
          "uuid",
          "f1ea29d9-3938-461b-be0b-7c781cb20365"
        ]
      ]
    ],
    "columns": [
      "ports"
    ],
    "timeout": 0,
    "op": "wait",
    "table": "Bridge"
  },
  {
    "rows": [
      {
        "ports": [
          "uuid",
          "516fc265-43fb-466a-98a0-d588872a3d99"
        ]
      }
    ],
    "until": "==",
    "where": [
      [
        "_uuid",
        "==",
        [
          "uuid",
          "c4395e6b-47cf-457c-8be3-786848d81052"
        ]
      ]
    ],
    "columns": [
      "ports"
    ],
```

```
    "timeout": 0,
    "op": "wait",
    "table": "Bridge"
  },
  {
    "where": [
      [
        "_uuid",
        "==",
        [
          "uuid",
          "f1ea29d9-3938-461b-be0b-7c781cb20365"
        ]
      ]
    ],
    "row": {
      "ports": [
        "set",
        [
          [
            "uuid",
            "c2b29779-ad40-4f07-bd8a-dfc4ba2d96b9"
          ],
          [
            "named-uuid",
            "rowf6036497_cfae_4a67_9163_774cb2912aed"
          ]
        ]
      ]
    },
    "op": "update",
    "table": "Bridge"
  },
  {
    "uuid-name": "row45673b5a_5c83_4079_a91c_5888ed4d9943",
    "row": {
      "name": "eth0"
    },
    "op": "insert",
    "table": "Interface"
  },
  {
    "uuid-name": "rowf6036497_cfae_4a67_9163_774cb2912aed",
    "row": {
      "name": "eth0",
      "interfaces": [
        "named-uuid",
        "row45673b5a_5c83_4079_a91c_5888ed4d9943"
      ]
```

```
      },
      "op": "insert",
      "table": "Port"
    },
    {
      "mutations": [
        [
          "next_cfg",
          "+=",
          1
        ]
      ],
      "where": [
        [
          "_uuid",
          "==",
          [
            "uuid",
            "92f648e0-77c4-4351-a3a8-2686ca2f9bc2"
          ]
        ]
      ],
      "op": "mutate",
      "table": "Open_vSwitch"
    },
    {
      "where": [
        [
          "_uuid",
          "==",
          [
            "uuid",
            "92f648e0-77c4-4351-a3a8-2686ca2f9bc2"
          ]
        ]
      ],
      "columns": [
        "next_cfg"
      ],
      "op": "select",
      "table": "Open_vSwitch"
    },
    {
      "comment": "ovs-vsctl (invoked by sudo):
            ovs-vsctl --db=tcp:192.168.8.16:6640 add-port br89 eth0",
      "op": "comment"
    }
  ]
}
```

Ringraziamenti

Ringrazio la mia famiglia per essermi stato vicino in questi anni e avermi continuato ad appoggiare anche nei momenti in qui pensavo che non ce l'avrei fatta.

Voglio anche ringraziare i miei amici di corso per l'incommensurabile aiuto che mi hanno fornito.

Infine ringrazio il mio relatore Walter Cerroni per la fiducia concessami.

Bibliography

[1] OVSDB Protocol
https://tools.ietf.org/html/rfc7047

[2] Utility for querying and configuring ovs-vswitchd
http://www.openvswitch.org/support/dist-docs/ovs-vsctl.8.html

[3] Open vSwitch database server
http://www.openvswitch.org/support/dist-docs/ovsdb-server.1.html

[4] Open vSwitch database schema
http://www.openvswitch.org/support/dist-docs/ovs-vswitchd.conf.db.5.html

[5] Open vSwitch Documentation
https://media.readthedocs.org/pdf/openvswitch/latest/openvswitch.pdf

[6] OpenFlow Switch Specifications
https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

[7] Command-line interface to ovsdb-server
http://www.openvswitch.org/support/dist-docs/ovsdb-client.1.html

[8] Global Digital 2018
https://www.slideshare.net/wearesocial/digital-in-italia-2018