SDN 開放源碼專案分析與操作

王順賢 [1] 林盈達 [1] 賴源正 [2]
[1] 交通大學資訊工程學系 [2] 台灣科技大學資訊管理系
Email: sswang.cs04g@nctu.edu.tw, ydlin@cs.nctu.edu.tw, laiyc@cs.ntust.edu.tw
September 18, 2015

摘要

在新型的網路架構 SDN 快速發展下，SDN 的開放源碼專案的數量也大量成長，要如何從數量繁雜的專案中找到所要的及如何藉 SDN 的開放源碼專案了解 SDN 網路架構是許多 SDN 初學者想知道的。本篇報告為基於 SDN 開放源碼社群針對 SDN 開放源碼之專案進行分類、分析，共六分項，並將專案實際進行實驗操作，實驗結果發現 Opendaylight Controller 搭配其內建的 Application: VTN 可以提供多控制器間協調，以擴大網路管理拓墣，減少單一控制器負載，提升網路管理操作上的彈性，若能加強使用者介面，將 SDN 應用在現實網路指日可待。

關鍵字: SDN、SDN 開放源碼專案、Controller、OpenDaylight、VTN

1. 簡介

面對現今網路架構的新舊革新，為了加速 SDN[1]網路架構發展，各界積極開發 SDN 的專案，也有許多研究團隊選擇將 SDN 專案開放給社群使用及開發，藉此希望能號召更多人共同研究，讓研發更能集思廣益並且加快 SDN 的建構，也可以提高專案的市占率，以在 SDN 領域占有一席之地。但因為各個專案的功能性及欲達到的目的性不盡相同，而且 SDN 網路架構極富彈性，使得許多專案難以分類、統整，導致初學者、開發者、使用者難以上手，無法搜尋到所要的專案，失去了開放源碼的目的。

SDN 介紹

Software-Defined Networking（SDN） 的概念是將網路設備中負責計算 封包路徑的控制層（control plane）與傳遞封包的資料層（data plane）分離，並且將控制層集中化、虛擬化至雲端伺服器由控制器（以下稱 controller）集中管理，而網路設備僅提供資料層的功能，只需傳遞封包，如圖 1 虛線所示。

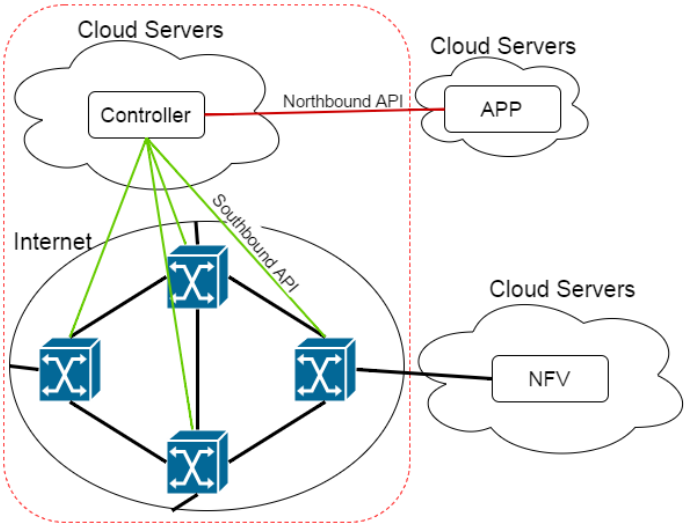

圖 1 SDN 網路架構圖

如圖一所示，SDN 架構對控制層的虛擬化使得網路管理可以在雲端上統一處理，不需要在由各個網路設備一一設定，另外計算封包路徑的功能也可以因應不同的需求而提供不同的服務，例如：服務品質（QoS），可以達到客製化的網路，因此新型的網路服務也孕育而生，稱應用程式（以下稱 Applications, App）。另一方面在資料層的功能簡化提升了網路設備的相容性，而在傳統網路的網路設備所提供的網路功能如防火牆（firewall）、網路位址轉譯（NAT）諸如此類，因應 SDN 的革新，所以也將網路功能虛擬雲端化，可以讓網路布局更加彈性，此項目稱為網路功能虛擬化（以下稱 Network Function Virtualization, NFV），其總體 SDN 架構如示意圖 1。

本篇將為基於 SDN 開放源碼社群[2][3]針對 SDN 開放源碼之專案進行六大分類的分析及比較，分別為:(1)Controllers、(2)Applications、(3)Network Function Virtualization、(4) Network Virtualization、(5) Virtual Switches、(6) Tools 介紹，最後將各專案整合並進行實驗操作，並針對實驗操作結果進行探討，以了解常用專案之成熟度。

2. SDN 開放源類別介紹

SDN 開放源碼社群上的專案種類繁多，但我們可以依其主要功能性可分為六大類:

- Controllers
- Applications
- Network Function Virtualization
- Network Virtualization

- Virtual Switches
- Tools

本章依上述分類於下列各小節中進行介紹與分析。

2.1 Controllers 介紹

Controller 的功能為計算封包路徑、決定封包的流向，其運作方式是當交換機（以下稱 Switch）接收到資料封包時，Switch 會先查詢自己的流程表（以下稱 flow table）確認是否有定義該封包流向規則，是則依照該規則定義傳遞，否則向 Controller 發出控制封包（packet in），Controller 接收到控制封包後，依照設定在背景（background）或前景（foreground）計算封包流向並回應給 Switch 更新其 flow table，而如同傳統網路的路由協定可以有許多不同的協定一樣，Controller 可以載入許多 Applications，變更其計算封包流向設定，來提供多樣的服務，表 1 為各種 Controller 開放源碼專案比較[4]-[14]。

表 1 各 Controller 專案比較

		NOX	**Beacon**	**Ryu**	**Pox**	**Trema**	**Floodlight**	**OpenMul**	**Opendaylight**	**ONOS**
Language		C++	Java	Python	Python	Ruby/C	Java	C	Java	Java
Support Southbound API	OF	1.0		1.0, 1.2, 1.3, 1.4	1.0	1.0	1.0	1.0, 1.3, 1.4	1.0, 1.3	1.0, 1.3
	Other							Netconf	OpFlex	Netconf
Learning Curve		Easy	Easy	Moderate	Moderate		Steep	Moderate	Steep	Steep
Performance		Fast	Slow	Slow	Slow		Fast	Fast	Fast	Fast

上表由左至右順序為時間先後，而我們可以從中發現較熱門或較新的 Controller 專案支援較多的 Southbound API，其中 Southbound API 為 Controller 與 Switch 交換控制封包的協定，其中最為主流的是 OpenFlow 1.3[15]，因為該協定為現今 SDN 網路架構定義最明確且成熟的，而這也可能是我們選擇開發專案的依據，另外兩項選擇的依據為專案使用的程式語言和 Controller 的模組化程度，這兩皆是以開發的難易度做評估，三項選擇依據可整理如下。

- Southbound API 支援的數量
- Controller 專案使用的程式語言
- Controller 的模組化程度

2.2 Applications 介紹

Applications 為 SDN 網路架構中一大賣點，如同現在當紅的手機 App 程式，我們可以在 Controller 載入不同的 Applications 服務配置到我們的網路中，例如:

我們可以使用機器學習來分析封包，分辨網路攻擊，並動態過濾可疑的封包，還有許多種不同的 App 在此無法一一介紹，另外如同手機 App 程式一樣，許多 App 都只能在 Android 或 Apple 系統上運作，現在幾乎所有的 SDN Applications 都只能在其對應之 Controller 上運作，是現在尚未突破的缺點。下表為各 Controller 專案所能使用的 Applications，但因 Applications 為數眾多，我們只能依其功能性做代表。

表 2 各 Controller 專案支援之 Applications

	NOX	Beacon	Ryu	Pox	Trema	Floodlight	OpenMul	Opendaylight	ONOS
Topology	✓	✓	✓		✓	✓	✓	✓	✓
Web UI	✓	✓	✓	✓		✓	✓	✓	✓
OpenStack Integation			✓			✓	✓	✓	✓
Network Analysis	✓							✓	✓
Security	✓				✓	✓		✓	✓
Service Chaining						✓		✓	

我們可以由表 2 分析出，較熱門及模組化程度較高的 Controller 擁有較多的 Applications 支援。其中值得探討的項目為 OpenStack[16] Integration，因為現在 SDN 大部份實作在 Data Center 的網路，而最通用的雲端管理系統為 OpenStack，其原因是 Data Center 的網路複雜且難以管理，使用 SDN 網路架構可使得建置、管理及應用較為容易。

因為本篇的實驗操作會使用 OpenDaylight 的 Controller 來做實驗，所以將 OpenDaylight 的 Applications 依其功能性做分類，如表 3。

表 3 OpenDaylight Controller 支援之 Applications

Management	(1) OVS- Management, (2) SNMP Plugin
Topology	(1) L2Switch, (2) BGP, (3) OpenFlow Flow Programming, (4) Topology Processing Framework
Service Chaining	(1)SFC over (L2, LISP, REST, VXLAN)
Network Secure	(1) Secure Networking Bootstrap, (2) Secure tag eXchange Protocol (SXP), (3) Unified Secure Channel (USC)
Network Analysis	(1) Time Series Data Repository (TSDR)
Network Function	(1) VPN Service, (2) VTN Service

2.3 Network Function Virtualization 介紹

Network Function Virtualization 如同第二章所介紹，是將原本網路設備提供的網路功能虛擬化至雲端中運作，不過這部份的專案只有兩項:（1）OPNFV[17]、

（2）CloudNFV[18]，兩者都只是提供應用 NFV 的平台，並無實體的服務，而兩者也正在開發中，尚不成熟，所以在此不多做介紹，但值得後續追蹤。

2.4 Virtual Switches 介紹

Virtual Switch 為虛擬的交換機，在進行 SDN 的實驗及測試時扮演 OpenFlow Switch 的角色，不需實際購買實體的 Switch，另外在建置雲端環境時，我們也常會使用 Virtual Switch 來提供 Virtual Machine 的網路建置，現在 SDN 開放源碼的 Virtual Switch 專案一共有六種如下：

- LINC Switch [19]
- Open vSwitch (OVS)[20]
- Indigo Virtual Switch (IVS)[21]
- Centec Lantern[22]
- CPqD OFSoftswitch 1.3[23]
- POFSwitch[24]

而 OVS 取得了壓倒性的市占率，其原因為該專案是所有 Virtual Switch 專案中，實作最多 OpenFlow 網路協定的 Virtual Switch，提供非常高的相容性，本篇末章的實驗操作亦使用該專案操作實驗。

2.5 Network Virtualization 介紹

Network Virtualization 的概念是在同一個實體網路下，各個用戶可以擁有不同的邏輯網路，營造出私有網路的假象，可以達到私有網路效果又能兼顧網路資源的善用，是現在雲端系統所追求的目標，而 SDN 即是達成 Network Virtualization 的實作方式之一。Network Virtualization 相關的專案有以下四項：

- OpenVirteX [25]
- FlowVisor [26]
- OpenVNet [27]
- OpenContrail [28]

其中我們選擇 OpenContrail 來做介紹，該專案之簡易架構圖如圖 3，圖中 Network Node 及 Compute Node 為 OpenStack 的元件，VRouter 及 Controller 為 OpenContrail 的元件。

如圖 2 中 Compute Node，我們想達到每個用戶的 VM 皆能擁有自己的 Virtual Network，OpenContrail 使用 VRouter 營造出區域網路的概念，在同一台 Server 區分出各個 VM，並且使用 GRE tunnal 或是 VXLAN 來達成 Network Virtualization，即便用戶的 VM 分散在不同 Server 或者在不同地理位置中，都可以實現。其中 VRouter 的功能如同 3.4 節的 Virtual Switch，不過為了強調其具有能處理網路層的服務的功能，所以強調為 VRouter。

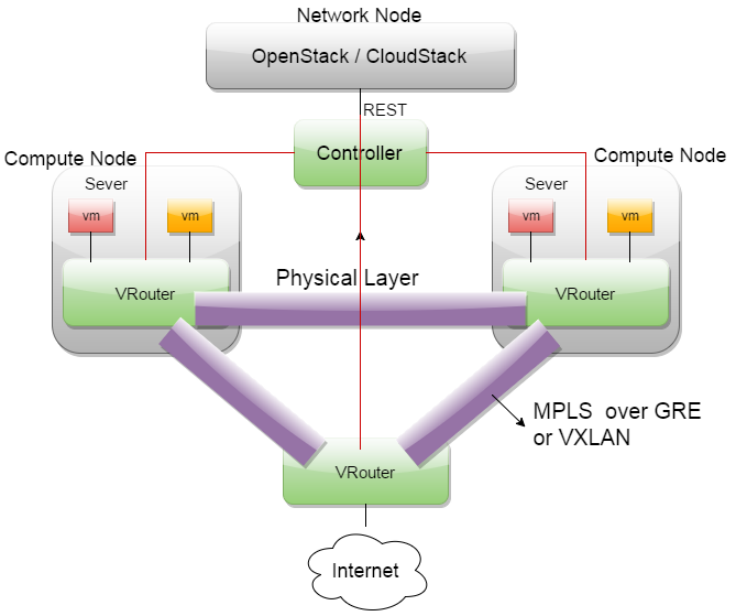

圖 2 OpenContrail 簡易架構圖

另外 OpenContrail 最大的特點是能藉由 SDN 來提供 Service Chaining 的服務，示意圖如圖 3。

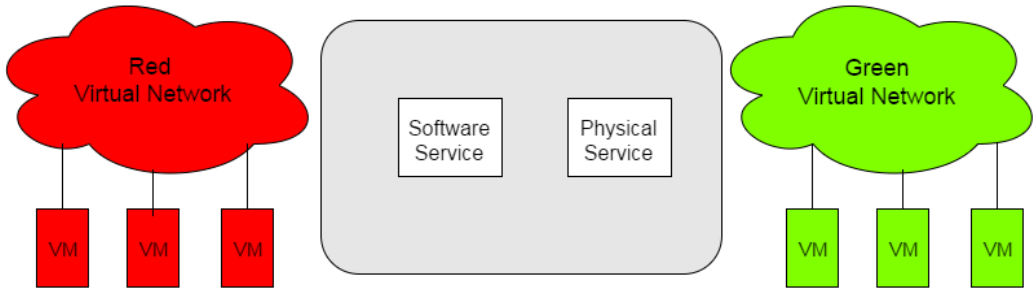

圖 3 OpenContrail Service Chaining 示意圖

假如我們想讓左方 VM 的用戶能與右方 VM 的用戶溝通但欲使其流量先經過許多網路服務例如:（1）firewall、（2）負載平衡、（3）NAT，不論是軟體服務或是硬體服務都能藉由 Controller 將流量引導至所在服務位址。

2.6 Tools 介紹

SDN 開放源碼專案也提供了許多建置 SDN 或操作 SDN 實驗的幫助工具，如下介紹。

- Mininet[29]
  提供 SDN 的模擬環境，使得實驗操作不需實際使用實體的 SDN 環境，其底層是使用 Open vSwitch 來實作，於本篇實驗操作亦會使用到。
- Flowsim[30]
  提供的功能與 Mininet 大致相同，但可以使用 Web UI 來操作，但主流

還是 Mininet 使用者較多。

- WireShark[31]
  提供監聽封包的功能，來了解實際封包內容。
- OFTest[32]
  用以 OpenFlow Switch 的評測，確認 Switch 是否符合 OpenFlow 的規格。

3.套件整合與操作

本章將利用前述之 SDN 開放源碼專案來建置 SDN 的環境，以了解實際 SDN 操作，本章分為兩節，第一小節介紹本實驗之環境設定，第二小節介紹實驗結果。

3.1 環境設定

本實驗所使用的 SDN 開放源碼專案如下：

- Controller: OpenDaylight
- Application: Virtual Tenant Network
  目的是提供在 SDN 網路架構下的各個用戶能擁有自己的私有網路，並能提供多 controller 間協調，使得不同 Domain 之網路得以互通及同步管理。
- Virtual Switch: Open vSwitch
- Tools: Mininet and WireShark

實驗目的欲在不同 Domain 間的 Hosts 能藉由 VTN 的服務來互相傳輸，實驗拓樸示意圖如圖 4。

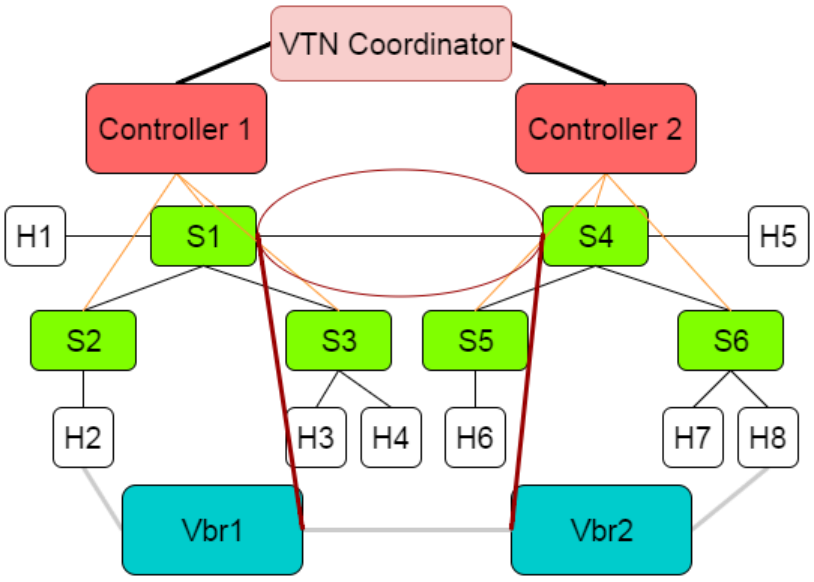

圖 4 VTN 實驗拓樸圖

圖 4 中 H1~H8 皆代表 Hosts，S1~S6 皆代表 virtual switch，Controller1 和 Controller2 分別跨兩個 Domain 的 OpenDaylight Controller 中間藉由 VTN Coordinator 做資訊協調，並虛擬出 Vbr1 及 Vbr2 使得不同 Domain 間的使用者能互相傳輸，並設定 H2 與 H5 為相同的虛擬網路，其他使用者則各自為獨立的網路。

實驗使用的作業系統為 Ubuntu 的 14.02.2 版，並使用 Mininet 來虛擬出網路拓撲，由 OpenDaylight Controller 分別控制兩個 Domain 的 Open vSwitch，在 Application 方面有主要兩個元件（1）VTN Coordinator、（2）VTN manager，VTN Coordinator 負責處理使用者的要求並藉由 REST API 轉達給 VTN manager；VTN manager 為 OpenDaylight Controller 裡的模組負責與 VTN Coordinator 對話並對 OpenDaylight Controller 下達指令。

3.2 實驗結果

本次實驗結果可以成功虛擬出一個區域網路提供兩個 hosts 使用，藉由 Mininet 上的指令 pingall 來觀察網路拓撲的連線狀況，可以發現只有 H2 與 H5 的連結是成功的；預設的網路拓撲連結在初始的狀態都為失敗的，但其中 Open vSwitch 的版本若為 2.3.0 以上的版本，需要先設定預設的 Flow entry，2.3.0 以上的版本預設的 Table Miss 的行為為 Drop 並不會 PacketIn，造成 controller 無法得知 switch 的狀況。

在實驗過程中使用 VTN 的服務的時候，OpenDaylight 並沒有提供較通用的使用者介面來控制網路，在使用 REST API 對 VTN Coordinator 設定時需使用 CLI 及 curl 的指令來設定，雖可使用 Google Chrome 的外掛程式 Postman 略為提升操作的容易度，但還是難以使用，另外已設定好的虛擬網路並無提供虛擬網路拓樸的顯示，只能使用指令查詢，且回報的資料不容易辨識，當使用者數量上升，網路管理的設定會更加困難。

4.結論

由本篇以上的介紹，可以知道目前 SDN 的開放源碼專案的發展與實用程度，以下作此兩點分析。

- SDN 開放源碼專案的發展情形
- SDN 開放源碼專案的實用情形

目前的 SDN 開放源碼專案的發展較熱門的種類為 Controllers 及 Applications，因為日後將 SDN 網路架構套用在現實網路的時候，兩者的商機將

不可限量，Controllers 競爭的情況可能會像現在的 Android 或 Apple 系統，而其效能及 Applications 的種類多寡及實用性都將是我們考慮的面向。

而目前 SDN 開放源碼專案雖具有一定的實用性但成熟度仍然不足，從實驗中可知，許多專案仍在開發階段，許多功能並不完善，目前還無法應用於現實網路，若能加強功能及穩定性，SDN 網路所提供的彈性必能創造出更多傳統網路所無法提供的服務，提升網路的效能及便利網路管理。

參考文獻

[1] K. Greene, Software defined networking, Technology review - the 10 emerging technologies of 2009, March 2009.

[2] sdxcentral, https://www.sdxcentral.com/

[3] OPEN SOURCE SDN, http://opensourcesdn.org/

[4] NOX, http://www.noxrepo.org/nox/about-nox/

[5] Beacon, https://openflow.stanford.edu/display/Beacon/Quick+Start

[6] Ryu, http://osrg.github.io/ryu/

[7] Pox, http://www.noxrepo.org/pox/about-pox/

[8] Trema, http://trema.github.io/trema/

[9] Floodlight, http://www.projectfloodlight.org/floodlight/

[10] OpenMUL, http://www.openmul.org/

[11] OpenDaylight, https://www.opendaylight.org/

[12] ONOS, http://onosproject.org/

[13] Coursera, https://class.coursera.org/sdn1-001

[14] Rahamatullah Khondoker, Adel Zaalouk, Ronald Marx, Kpatcha Bayarou, Feature-based Comparison and Selection of Software Defined Networking (SDN) Controllers, Jan 2014.

[15] Open Networking Foundation, OpenFlow Switch Specification, openflow.org, October 14, 2013

[16] OpenStack, https://www.openstack.org/

[17] OPNFV, https://www.opnfv.org/

[18] CloudNFV, http://www.cloudnfv.com/

[19] LINC Switch, http://flowforwarding.github.io/LINC-Switch/

[20] Open vSwitch, http://openvswitch.org/

[21] Indigo Virtual Switch, http://www.centecnetworks.com/en/OpenSourceList.asp?ID=260

[22] Centec Lantern, http://www.centecnetworks.com/en/Main.asp

[23] CPqD OFSoftswitch 1.3, http://www.cpqd.com.br/

[24] POFSwitch, http://www.poforwarding.org/
[25] OpenVirteX, http://ovx.onlab.us/
[26] FlowVisor, https://github.com/OPENNETWORKINGLAB/flowvisor/wiki
[27] OpenVNet, http://openvnet.com/
[28] OpenContrail, http://www.opencontrail.org/
[29] Mininet, http://mininet.org/
[30] Flowsim, https://flowsim.flowgrammable.org/
[31] WireShark , https://www.wireshark.org/
[32] OFTest, http://www.projectfloodlight.org/oftest/