在軟體定義的網路下實作具擴充性之服務佈建之設計與實現

[1]林俊宏 [2]蔡順昶
國立中山大學資訊工程學系
[1]lin@cse.nsysu.edu.tw , [2]s9772101@gm.pu.edu.tw

摘要

Open vSwitch 為 Nicira 所進行的開源項目，其旨在提供虛擬化平台上的虛擬交換機。而 Linux container 是有別於一般虛擬機器的輕量級虛擬機器。本論文主要以此兩項工具為主並配合 OpenFlow 協定去建構出虛擬網路資料中心。之後透過 OpenFlow controller 的設定，可以令指定之虛擬交換設備執行特定的規則，使其下封包路徑獲得有效控管。透過這樣的系統，使用者可以便於將自己的服務串連在一起，並且能夠隨時根據情況調整交換器的規則。

關鍵詞：Linux container、Open vSwitch、OpenFlow、GRE tunnel。

Abstract

Open vSwitch is an open source software developed by Nicira, its purpose is providing virtual switch on virtual platform. Linux container is a light virtual machine unlikely traditional virtual machine. This thesis uses both Open vSwitch and Linux container with OpenFlow to build virtual network data center. Then setting via OpenFlow controller to execute specific rule on assigned virtual switch to manage network path effectively. Users can easily customs their services through this system which will adjust the switch's rule dynamically on different conditions.
Keywords: Linux container、Open vSwitch、OpenFlow、POX、SDN

1. 前言

本論文目的將探討如何建構一個以虛擬機器為主，並支援 OpenFlow 協定的網路環境，使用者能夠輕易創造虛擬環境，並且自動讓虛擬機新增至虛擬交換器設備底下。再者我們將探討如何加入 OpenFlow 協定，讓管理者能夠輕易管理各個虛擬機器的封包流向。讓使用者不需要真正擁有實體的 OpenFlow switch 也能夠使用 OpenFlow 管理和佈署自己的網路環境。

2. 背景知識

2.1 LXC

Linux container 又可簡稱為 LXC 為 Linux 系統上獨有的虛擬機器，可以直接使用機器本身的硬體資源，免去了模擬這些硬體裝置所需要消耗資源。

2.2 SDN

根據 Open Networking Foundation 對 SDN 的定義，SDN 是一種動態，可管理，低成本，且擁有高適應性能夠理想的應用於目前高頻寬，多變化軟體現況之新興架構。SDN 網路將目前的網路分為控制層和資料層兩個階層。而 OpenFlow 則是目前較為人所知的建造起 SDN 架構的解決方法。

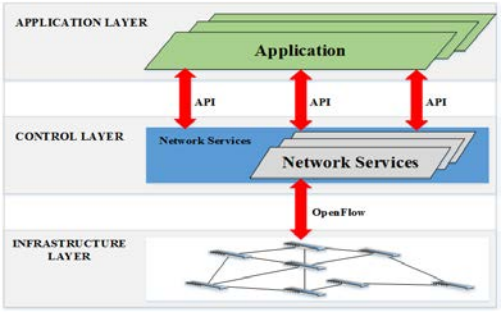

圖 1. SDN(軟體定義網路) 架構

2.3 OpenFlow

OpenFlow 起源於 Stanford University 的 Clean Slate 計劃，此計劃主要研究各種不同的方法去重新設計網路架構。OpenFlow 便是基於 SDN 網路的概念所制定出來的公開的標準，用於 SDN switch 和 SDN controller 之間的溝通。目前由 ONF 持續研究和更新 OpenFlow。

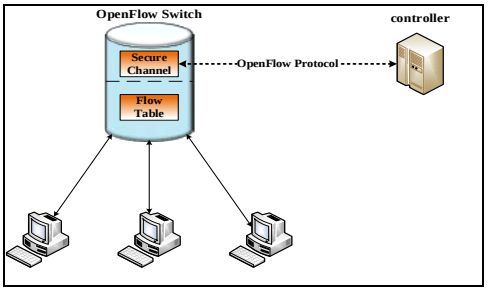

圖 2. OpenFlow 網路基本架構

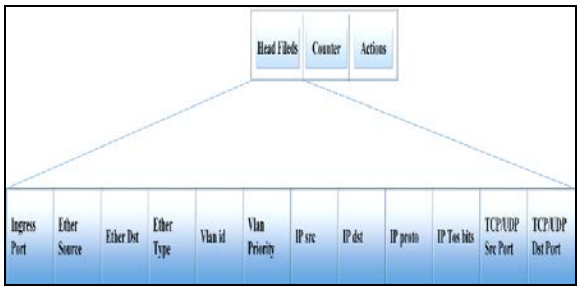

圖 3. Flow table

2.3.1 Matching

封包會依照優先權依序比較 flow table 中的 flow entry，當比對成功後，相對應的 counter 會隨之更新。假如無法找到相對應的 flow entry，則會將封包轉發給 controller。

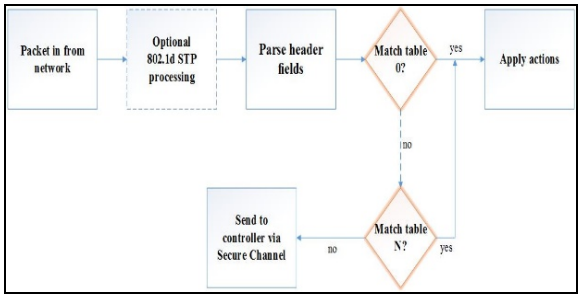

圖 4. Packet flow in an OpenFlow switch

2.4 Open vSwitch

Open vSwitch 是虛擬的交換器設備，它的目標鎖定在多服務的虛擬化平台佈署，在這樣的環境中通常存在可隨意更動的端點，且更動狀況頻繁，而 Open vSwitch 可以滿足這樣的需求。

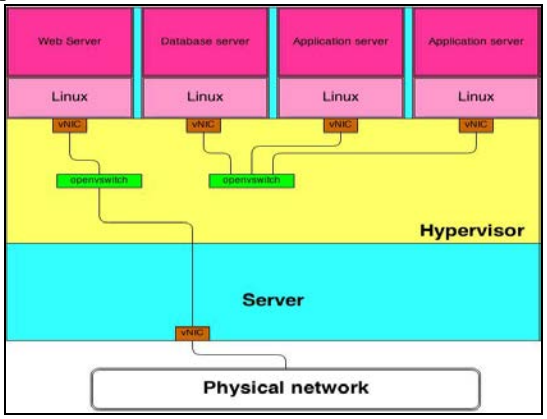

圖 5. Open vSwitch 在一般應用架構中扮演的角色

2.5 POX controller

POX 為一個用 python 語言開發，用於 SDN 網路中，作為控制器角色的開源軟體。POX 的前身為 NOX，後來開發團隊基於效能和開發便利性的原因，改以 python 為基底打造出 POX。

2.6 Django

Django 是由 python 所撰寫的一套高階的網頁框架，目的在於讓開發者能夠快速、乾淨、務實的完成網站。起源於 Kansas 洲 Lawrence 城中，由於當地一間新聞社的程式人員時常需要趕在截稿前完成網站的更新和變動，為了需要，它們便開發出這樣一個框架。

3. 系統架構與功能

本系統最基本需求為 SDN 網路之基本架構，因此以一台 controller 搭配複數台 Host 為主，而在兩者之間則以一般的第二層網路設備做連結。下圖中做為 controller 的設備，我們則用 Raspberry Pi 進行。

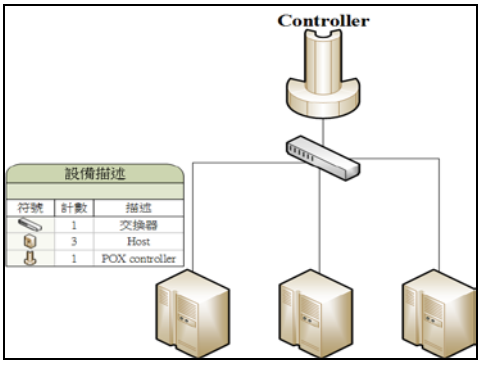

圖 6. 硬體系統架構圖

3.1.1 系統軟體架構

軟體部分主要分兩個層級，第一個部分是 Web Server 搭配 controller。因此這部份我們要考慮的是能夠讓 Web server 和 controller 結合。第二部分則以 Open vSwitch 搭配 LXC，做出虛擬的網路環境。而這兩個區塊之間則以實體網路做連結。

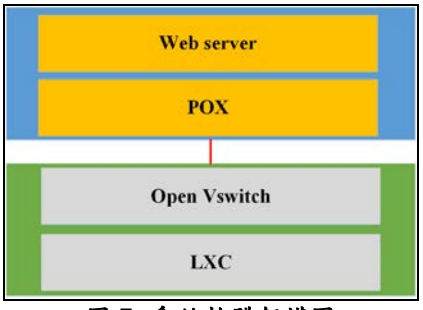

圖 7. 系統軟體架構圖

3.1.2 系統服務要求順序

當使用者發出一項基本的要求時，首先會抵達 nginx server 做處理。nginx 處理靜態要求效能極佳，而動態要求部分則會往後遞送。動態要求抵達 uWSGI server 後會依照 uwsgi protocol 做轉換，並往後遞送。 經過轉換的要求到達 Django server，此時便依照需求去執行 Django APP，假如有需要儲存的資料則透過 ORM 方式，存入 sqlite。最後 POX

controller 則會由 sqlite 取得所需的資料，並由 POX 使用的元件來進行相關應用。

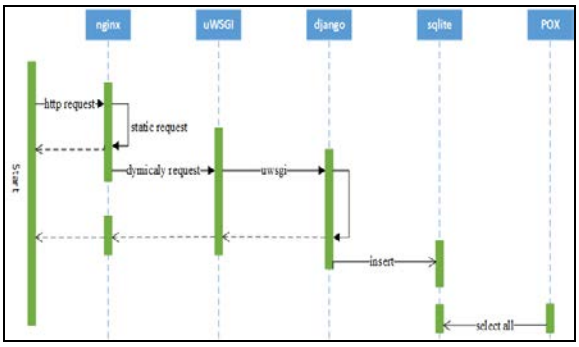

圖 8. http request 循序圖

3.1.3 系統功能

此頁面可選擇要進行的功能，如下圖所示，有兩項基本 Django 預設功能 Groups 和 User 的設定，以及我們自行新增的 Flows 功能。

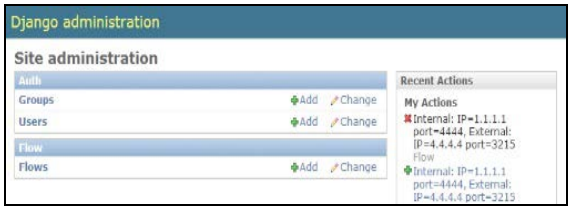

圖 9. 主要功能選擇頁面

此頁面選擇 flow 符合的規則，及封包傳送的方式。

圖 10. flow 製作頁面

此頁面提供使用者修改已經加入的 flow。

圖 11. flows 修訂頁面

使用者可以直接刪除已存在的 flow。

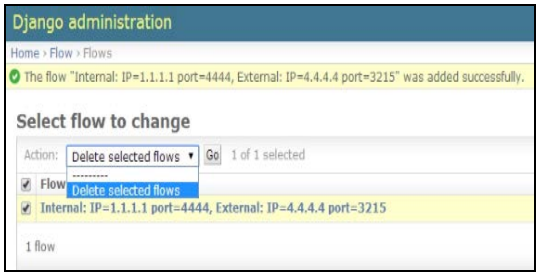

圖 12.刪除 flows 功能演示圖

4. 實作技術探討

4.1 LXC 和 Open vSwitch 的整合

我們透過修改 LXC 全域的設定來使得 LXC 在啟動之後能夠自動接上，我們利用 Open vSwitch 所建立的 Bridge。每個 LXC 除了自己內部的網路介面，在其外部，也就是 host 上也有相對應的虛擬網路介面。

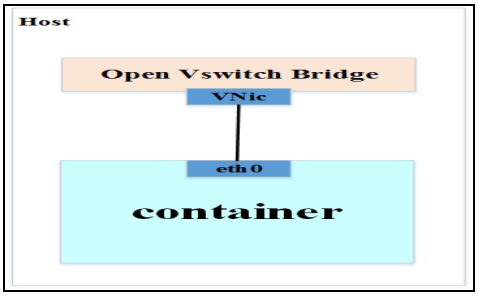

圖 13. LXC 對應 VNic 示意圖

LXC 全域設定檔案之預設路徑在 /etc/lxc/default.conf，我們在此檔案內新增兩行設定

```
1 lxc.network.script.up = /etc/lxc/ovsup
2 lxc.network.script.down = /etc/lxc/ovsdown
```

此處之 ovsup 以及 ovsdown 為我們自行撰寫之 script file，另外需注意的是 script.down 這項功能，在 LXC 0.75 以前的版本並不支援，因此可利用 PPA 安裝較新版本的 LXC。

```
1 #!/bin/bash
2
3 BRIDGE="ovsbr0"
4 ovs-vsctl -- --may-exist add-br $BRIDGE
5 ovs-vsctl -- --if-exists del-port $BRIDGE $5
6 ovs-vsctl -- --may-exist add-port $BRIDGE $5
```

圖 14. ovsup 內容

此處$5 變數即為 LXC 在 host 上所對應的虛擬網路介面名稱。透過此 script file，我們在每次啟動 LXC 時，檢查 Open Vswith 之 bridge 是否存在，並檢查所對應之虛擬網路介面是否有名稱重複的情況，假如沒重複才將其加入 bridge。同樣在 LXC 關閉時也做類似的處理。

```
3 BRIDGE="ovsbr0"
4 ovs-vsctl -- --if-exists del-port $BRIDGE $5
```

圖 15. ovsdown 內容

4.2 Django 和 POX 的整合

在這個部份我們需要修改 POX 的元件，以及自訂一個 Django 的 APP 來做配合。inux container 又可簡稱為 LXC 為 Linux 系統上獨有的虛擬機器，和一般常見的 VMware 或者 Virtual box 不同

4.2.1 Django APP

首先建立 Django APP。python manage.py startapp “APP name”。透過上述指令，會在 Django 的 project 內新增一個以我們新增的 APP 名稱為名的資料夾。接著增加我們要與資料庫互動的項目，修改 APP 資料夾底下的 models.py。範例:

```
1 class Flow(models.Model):
2 NwInternalIp=models.CharField(max_length=200)
3 NwExternalIp=models.CharField(max_length=200)
4 InternalPort = models.IntegerField()
5 ExternalPort = models.IntegerField()
6 idletime = models.IntegerField()
7 hardtime = models.IntegerField()
```

此處我們增加一些項目讓使用者在 web 介面可以填寫，並設定與之對應在資料庫內欄位的形態。之後我們必須讓我們自訂的 APP 加入 Django 預設的頁面，修改 APP 資料夾底下 admin.py 檔案。新增以下一行。

```
1 admin.site.register(Flow)
```

Flow 即範例修改 models.py 時所訂之 class 名稱。之後我們必須修改 Django project 的設定檔 settings.py，加入我們自訂的 APP。

```
1 INSTALLED_APPS = (,'flow', )
```

此時的 flow 是我們新增 APP 所使用的名稱，另外此欄位內原本有的 APP 並沒有刪除，此處沒有列出僅是突顯新增自己的 APP 之意。

4.2.2 POX component

我們直接修改 POX 所提供的元件來和 Django 做結合，如此一來不但可以保有此元件原有的功能，也能夠提供使用者容易自訂 flow 的彈性。我們選定 l2_learning 此一元件。修改 l2_learning.py 如以下範例，首先引入我們會用到的 class 以及加入 Django 環境設定:

```
1
os.environ.setdefault("DJANGO_SETTINGS_MOD
ULE","Django setting file")
2 from flow.models import Flow
3 from pox.lib.revent import *
4 from pox.lib.packet import ethernet
5 from pox.lib.packet import ipv4
```

範例第一行環境設定的方式根據使用之 Django 版本不同，有所不同。本論文使用為 1.6 版。標底線之參數即為 Django project 的設定檔位置。範例第二行則引入我們在前面自訂的 class。之後新增我們要比對的條件和相對應的動作如下:

```
1 def AddFlowFromModel(self, flow):
2    msg = of.ofp_flow_mod()
3    msg.match = of.ofp_match()
4    msg.match.dl_type = ethernet.IP_TYPE
5    msg.match.nw_src = str(flow.NwInternalIp)
6    msg.match.nw_dst = str(flow.NwExternalIp)
7    msg.match.in_port = flow.InternalPort
8    msg.actions.append(of.ofp_action_output(port =
flow.ExternalPort))
9    self.connection.send(msg)
```

我們自訂一個 function，會由資料庫中將訂好的 flow 之值取出，並配合 POX 定義之結構配置。在第 8 行的地方，我們決定符合規則的封包將被執行的動作。在第 9 行則將此 flow 寄給所有與 POX controller 連接的 OpenFlow switch，此處也可以針對 dpid 寄送。之後依照我們的需求，在適當的地方送出 flows 給 switch:

```
1 for flow in Flow.objects.all():
2    self.AddFlowFromModel(flow)
```

4.3 網路架構連接方式探討

此部分將探討不同網路架構的設定方式，以便配合不同情境使用。

4.3.1 直接將兩台電腦對接

一般電腦之間要溝通我們會透過一台實體的第二層設備來做為中介，透過第二層設備能夠便認出 MAC Address 的功能來達到資訊交換。然而此種方式將受限於中介設備本身的能力，假如電腦主機本身的網路介面傳輸能力大於中介設備，則無法盡全力發揮電腦主機本身設備的能力。

因此在需要大量傳輸的情況下，我們可以採用直接對接的方式。我們同樣在每台電腦上安裝 Open vSwitch，並將電腦上原先擁有的網路介面一一加入我們利用 Open vSwitch 所建造出來的 bridge。此時的情況就如同每台電腦自己本身都有一台第二層設備，之後互相串接。參考圖 5.4。

而電腦內則利用 LXC 的技術實現虛擬的叢集系統的架設。IP 配置部分，則可配合 Mask 的設置，達成不同網段的連接。有別於傳統網路，必須將實體網路重新架設，並主動進行設定，我們可規劃部分電腦原先即採用對接方式，在需要實現特殊功能的情況下，直接使用 POX controller，以 POX 提供之元件 l2_learning 可使 Open vSwitch 實現第二層設備的功能。

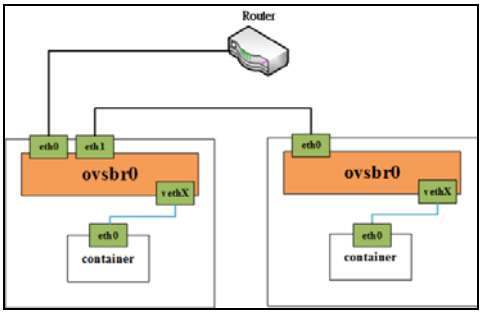

圖 16.直接對接表示圖

4.3.2 多台電腦對接

為了證明使用 OpenFlow 的便利性，此處以四台電腦互相對接的情況來做測試。如圖 5.5，每台

電腦均有 4 個網路介面，其中一個網路介面接至外部網路，其餘則和另外 3 台電腦做對接。

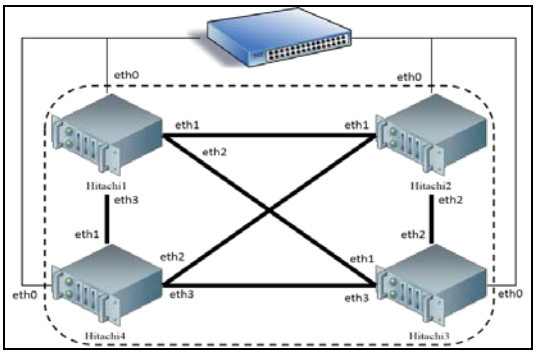

圖 17.多台電腦對接示意圖

除了電腦對接以外，每台電腦底下則建立 4 個 Linux container，用於做資料處理測試。

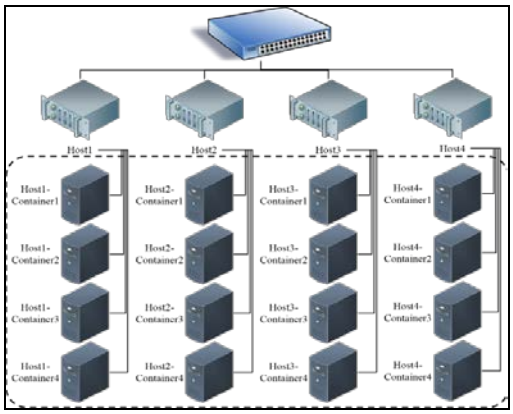

圖 18.多台電腦對接搭配 LXC 示意圖

4.3.2.1 設定方式比較

傳統方式:此處列出每台電腦配置一台 container，需要設置的 routing table 之指令。

表 1. 多台電腦對接設定表

網路界面	設置方式
Hitachi1-Container1	eth0 設置為對外網路，與 br0 連接 route add -host __Hitachi2-Container1_IP__ dev eth1 route add -host __Hitachi3-Container1_IP __ dev eth2 route add -host __Hitachi4-Container1_IP __ dev eth3
Hitachi2-Container1	eth0 設置為對外網路，與 br0 連接 route add -host __Hitachi1-Container1_IP__ dev eth1 route add -host __Hitachi3-Container1_IP__ dev eth2 route add -host __Hitachi4-Container1_IP__ dev eth3
Hitachi3-Container1	eth0 設置為對外網路，與 br0 連接 route add -host __Hitachi1-Container1_IP__ dev eth1 route add -host __Hitachi2-Container1_IP__ dev eth2 route add -host __Hitachi4-Container1_IP__ dev eth3
Hitachi4-Container1	eth0 設置為對外網路，與 br0 連接 route add -host __Hitachi1-Container1_IP__ dev eth1 route add -host __Hitachi2-Container1_IP__ dev eth2 route add -host __Hitachi3-Container1_IP__ dev eth3

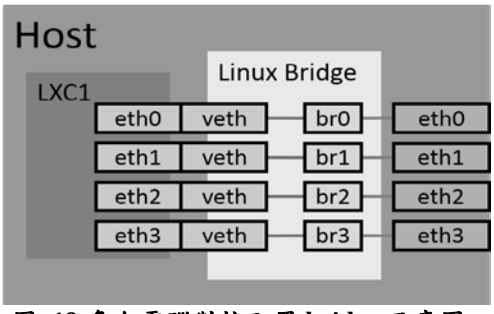

圖 19.多台電腦對接配置 bridge 示意圖

使用 POX 的方式:

./pox.py forwarding.l2_learning OpenFlow.discovry OpenFlow.spanning_tree

4.3.2.2 比較結果

在傳統設置方面，必須針對每網路介面之流向做相對應的設定，當電腦主機以及虛擬環境變多，所需要的配置便越多，會造成網管人員極大的困擾。尤其以傳統方式來做，通常需要每台電腦逐一設定。

若是透過 OpenFlow 的方式，僅須套用相對應的 App 或元件即可輕鬆掌控所有的交換設備。由於我們測試的方式在邏輯上是將所有的 switch 串接起來，因此會造成 looping 的情況，所以需另外套用 spanning_tree 此元件。不論如何，此處我們證明針對不同的網路架構所展生的情形，網管人員不再需要針對每台電腦或者交換設備做設定，而是透過 OpenFlow controller 做一個總管控即可。

4.3.2.3 實際測試結果

此處利用資料倉儲的軟體做測試，主要證明利用 OpenFlow 的方式所配置的網路架構，仍為穩定狀態。並且如果設計得當，效果是很好的。

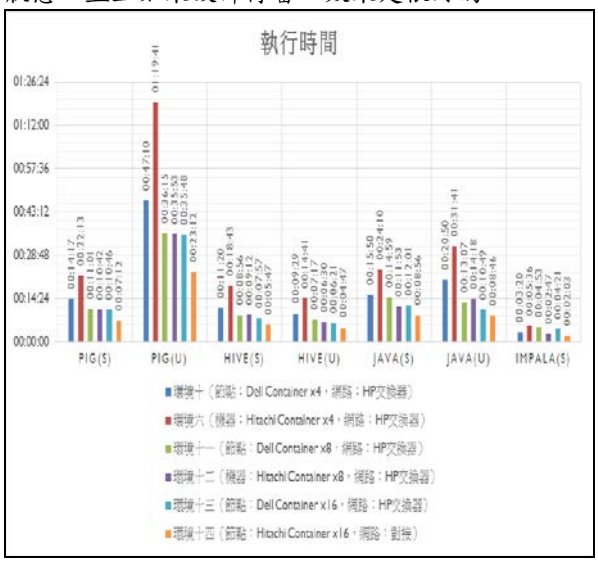

圖 20.多台電腦對接情況下實測 data warehouse 運作之結果數據

4.3.2.4 不同網段的網路架構探討

有時候我們會希望兩個不同的網段底下的機器可以互相溝通，但我們不希望依靠控制 Mask 的方式來達到這項目的，此時我們可以利用 POX controller 提供的 l3_learning 元件配合 fake way 參數來達到這項目的。參考圖 5.10。範例:

```
./pox.py forwarding.l3_learning
--fakeways=10.0.0.1,192.168.0.1
```

利用 fakeway 參數，POX controller 會自行回答 ARP Replay 至相應的 Open vSwitch bridge，使其知道該如何抵達目的地，如同兩者之間存在通道。另外範例中僅給予兩個網段為參數，而實際上可以多於兩個網段。

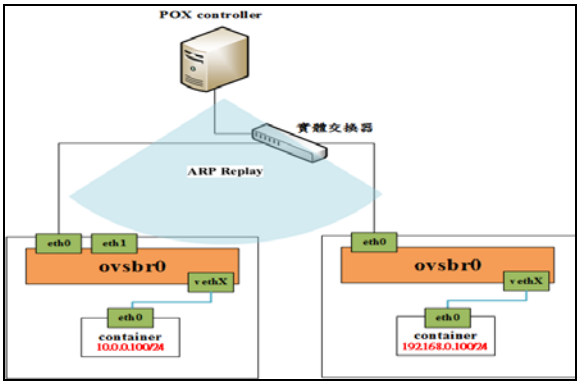

圖 21.fakeway 範例圖

5. 結論

5.1 結論

本論文貢獻在於將 Open vSwitch、LXC、POX controller 整合在一起，並且利用 Django 提供使用者能夠輕易使用 SDN 網路架構的介面，而後再利用各自的功能去架構出網路環境，期望能在這樣的基礎底下延伸出許多不同的變化。

在論文撰寫過程，可以發現 POX controller 以及 Open vSwitch 各自強大的功能，Open vSwitch 除了自身有多種模式，其本身也具備各式功能，可以搭配不同狀況使用，而其支援 OpenFlow 程度相當高，許多建議支援的功能，Open vSwitch 都已俱備，由於本論文主題主要相關為 OpenFlow 的應用，故沒有針對其他功能做介紹，然而單是 Open vSwitch 本身所具備功能，已足夠衍生許多有趣的題目。

5.2 未來展望

未來若要持續研究，首先不得不先替換 controller，因 POX controller 僅支援 OpenFlow 1.0，而目前 OpenFlow 已釋出 1.4 的版本。1.0 的版本僅擁有一張 Flow table 的做法，確實讓初上手者容易理解，但也侷限了發展性，就目前已知，較後的版本不僅有多張 Flow table，且也多出 group table 等新功能。另外一方面，在完成此次論文後，已對整體架構和開發方式有進一步的了解，因此會希望未來以開發最上層之 APP 為主，以實現 OpenFlow 最原先的目的。

參考文獻

[1] Open Networking Foundation,OpenFlow Switch Specification Version 1.1.0, 2011/02

[2] Open Networking Foundatio, https://www.opennetworking.org/

[3] Open vSwitch office website, http://openvswitch.org/

[4] Sam Russell Central Blog, http://pieknywidok.blogspot.tw/

[5] Django office website, https://www.djangoproject.com/

[6] Software-Defined Networking: The New Norm for Networks,2012/04

[7] Zdravko Bozakov and Panagiotis Papadimitriou. 2012. AutoSlice: automated and scalable slicing for software-defined networks. In Proceedings of the 2012 ACM conference on CoNEXT student workshop (CoNEXT Student '12). ACM, New York, NY, USA, pp.3-4.

[8] Ryota Kawashima. 2012. vNFC: A Virtual Networking Function Container for SDN-Enabled Virtual Networks. In Proceedings of the 2012 Second Symposium on Network Cloud Computing and Applications (NCCA '12). IEEE Computer Society, Washington, DC, USA, pp.124-129.

[9] Rami Rosen. 2014. Linux containers and the future cloud. Linux J. 2014, 240, pages.

[10] Miguel G. Xavier, Marcelo V. Neves, Fabio D. Rossi, Tiago C. Ferreto, Timoteo Lange, and Cesar A. F. De Rose. 2013. Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments. In Proceedings of the 2013 21st Euromicro International Conference on Parallel,Distributed, and Network-Based Processing (PDP '13). IEEE Computer Society, Washington, DC, USA, pp.233-240.

[11] Masayoshi Kobayashi, Srini Seetharaman, Guru Parulkar, Guido Appenzeller, Joseph Little, Johan Van Reijendam, Paul Weissmann, and Nick Mckeown. 2014. Maturing of OpenFlow and Software- defined Networking through deployments. Comput. Netw. 61 (March 2014), pp.151-175.

[12] Balazs Sonkoly, Andras Gulyas, Felician Nemeth, Janos Czentye, Krisztian Kurucz, Barnabas Novak, and Gabor Vaszkun. 2012. OpenFlow Virtualization Framework with Advanced Capabilities. In Proceedings of the 2012 European Workshop on Software Defined Networking (EWSDN '12). IEEE Computer Society, Washington, DC, USA, pp.18-23.

[13] Mukta Gupta, Joel Sommers, and Paul Barford. 2013. Fast, accurate simulation for SDN prototyping. In Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking (HotSDN '13). ACM, New York, NY, USA, pp.31-36.

[14] XAVIER, Miguel G.,et al. Performance evaluation of container-based virtualization for high performance computing environments. In: Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on. IEEE, 2013. p. 233-240.