延伸 Dijkstra 演算法達成軟體定義網路最佳路由

Achieving Optimal Routing for Software Defined Networking by Extending Dijkstra's Algorithm

江振瑞(Jehn-Ruey Jiang) 黃信文(Hsin-Wen Huang) 廖基豪(Ji-Hau Liao) 陳思源(Szu-Yuan Chen)

國立中央大學
資訊工程研究所

摘要

軟體定義網路(Software defined networking, SDN)是近年來很熱門的研究領域，其概念為將網路裝置的控制層面(control plane)與資料層面(data plane)分離，並由控制器(controller)集中管理所有裝置的控制層面。本論文延伸著名的最短路徑演算法 — Dijkstra 演算法，除了考慮網路拓樸相對應圖(graph)的邊權重(edge weight)之外，也一併考慮節點權重(node weight)來求取最短路徑，以達成軟體定義網路最佳路由。本篇論文使用 Pyretic 語言來實作延伸的 Dijkstra 演算法(extended Dijkstra algorithm)，並使用 Mininet 網路模擬器，在 Abilene 網路拓樸中，利用 Iperf 工具模擬發送最大量的 TCP 封包進行測試，以比較原始的 Dijkstra 演算法、全單位權重的 Dijkstra 演算法(unit-weighted Dijkstra algorithm)和延伸的 Dijkstra 演算法在端對端延遲(end-to-end latency)的效能比較。模擬結果顯示，延伸的 Dijkstra 演算法在效能上勝過其他的演算法。

關鍵字:軟體定義網路、最短路徑、Dijkstra 演算法、網路拓樸

一、簡介

軟體定義網路(Software defined networking, SDN)是一個將網路裝置的控制層面(control plane)與資料層面(data plane)分離[8][15]的概念。Mckeown 等學者提出了 OpenFlow 協定來實現 SDN 的概念，讓研究者可以實驗新的網路協定[6]。在軟體定義網路中，一個邏輯上為集中式(centralized)的控制器(controller)可以配置多台交換器(switch)上的轉送表(forwarding table)或流表(flow table)，以轉送通訊流(flow)的封包。如此一來，SDN 使用者可以編寫可以在控制器上執行的應用程式，以即時及集中的方式來監控和管理整個網路。

SDN 技術的出現，帶出許多可以在 SDN 控制器上執行的網路應用程式(network application)。典型的網路應用程式包括負載平衡(load balancing)、多媒體群播(multimedia multicast)、入侵偵測(intrusion detection)及網路虛擬化(network virtualization)[2]等。有些研究學者開發出方便設計 SDN 網路應用程式的程式語言，例如 Frenetic [3]和 Pyretic [11]。Frenetic 是一種宣告式查詢語言(declarative query language)，可以將網路流量分類，並提供回應式的函式庫來描述高階的封包轉送策略[3]。基於 Python 語言，Pyretic 是由 Frenetic 延伸出來的語言。Pyretic 提高了網路抽象化的層級，讓程式設計師可以建構出 SDN 的模組化軟體[3]。

在本篇文章中，我們延伸了眾所熟知的 Dijkstra 最短路徑演算法[1]，不僅考慮 SDN 網路拓樸相對應圖(graph)的邊權重(edge weight)，也一併考慮節點權重(node weight)來求取最短路徑，以達成軟體定義網路最佳路由。我們使用 Pyretic 來實作延伸的 Dijkstra 演算法，並使用 Mininet 網路模擬器，在 Abilene 網路拓樸下，與原始的 Dijkstra 演算法和每邊都是單位權重的 Dijkstra 演算法(unit-weighted Dijkstra algorithm)在端點對端點延遲

(end-to-end latency)的效能比較。比較的結果顯示，延伸的 Dijkstra 演算法在效能上勝過其他的演算法。

我們注意到論文[4]討論使用 OpenFlow 實作修改的 Dijkstra 演算法(modified Dijkstra algorithm)[10]和修改的 Floyd-Warshall 最短路徑演算法的問題。然而，上述修改的 Dijkstra 演算法不同於本篇所提出的延伸的 Dijkstra 演算法。因為前者是修改原始演算法來解決多重源點單一終點(multi-source single-destination)的最短路徑問題，而後者是延伸原始演算法，一併考慮節點和邊的權重來解決單一源點多重終點(single-source multi-destination)的最短路徑問題，因此在本論文中我們並不與修改的 Dijkstra 演算法進行效能比較。值得一提的是，本論文所提出的延伸的概念也可以應用在修改的 Dijkstra 演算法上。

本論文其他部份的內容如下所述。在第二節中，我們介紹了一些初步的知識，包括 SDN 的概念、Pyretic 和 Mininet。第三節描述延伸的 Dijkstra 演算法及其實作；第四節則展示效能模擬與比較結果。最後第五節則為全篇論文下結論。

二、相關研究

2.1 軟體定義網路

SDN 提出將控制層面和資料層面分開，讓底層稱為交換器(switch)的硬體交換裝置，交由在稱為控制器(controller)的控制裝置上執行的軟體(稱為應用程式)控制[15]。圖 1 描述 SDN 邏輯上的架構。SDN 能讓網路管理者撰寫應用程式來管理網路服務，包括路由(routing)、存取控制(access control)、群播和其他傳輸工程(traffic engineering)的工作。OpenFlow 是在 SDN 架構中，率先定義了控制層裝置(控制器)與資料層裝置(交換器)間的開放協定之一[15]。一個 OpenFlow 交換器包含一或多個流表和群流表(group table)，如圖 2 所示。一個 OpenFlow 控制器可以主動或被動地更新、增加和刪除流表中的流項(flow entry)。每台交換器中的流表包含一條以上的流項，每個流項資料包含了比對欄位(match fields)、優先等級(priority)、計數器(counters)和指令(instructions)等，如圖 3 所示。

當交換器接收到封包時，首先使用流表中每一流項的比對欄位與封包對應欄位進行比對，比對的程序依優先順序從第一張流表開始，針對所有流項比對，然後再比對下一張流表的所有流項，依此相同方式直到最後一張流表被比對完為止。依此方式，第一個成功比對吻合流項中的指令將被取出執行，可能的指令包括透過某些特定的端口(port)轉送封包到其他的交換器、修改封包表頭、到另一個特定流表上比對封包、丟棄封包等。若任何流表都無法成功比對，則會執行流表的預設指令，一般的預設指令為將封包轉送到控制器處理[9]。而控制器的處理一般為針對封包進行路由計算，然後送出指令在適當的交換的流表上寫入特定的流項，以便讓後續的封包可以依新寫入的流項在交換器上直接處理。

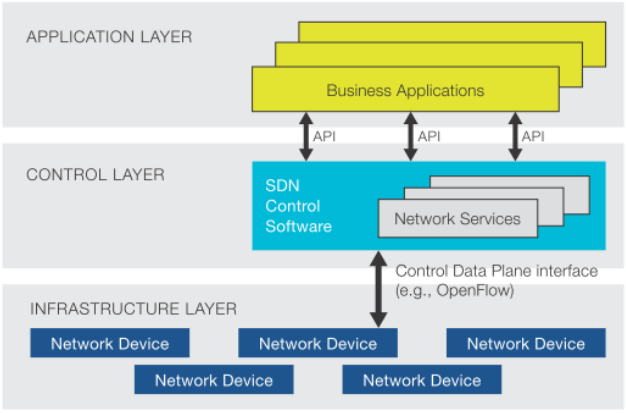

圖 1、SDN 架構示意圖 [15]

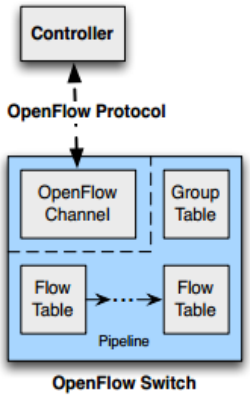

圖 2、OpenFlow 控制器和交換器 [9]

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

圖 3、OpenFlow 交換器中的流項 [9]

2.2 Pyretic

Pyretic 由 Frenetic 延伸而來。Frenetic 是一個 SDN 網路中用來撰寫控制器應用程式來管理交換器的高階語言[3]。Frenetic 提供程式設計師一個宣告示查詢的介面，來分類和彙整網路流量，以及一個回應示的函式庫來描述高階的封包轉送策略。

基於 Frenetic 和 Python，Pyretic 被定位為一個可以提高抽象化程度的 SDN 程式語言或平台[11][7]，可以讓程式設計者專注於設計較高層次的抽象化網路策略。例如，Pyretic 隱藏了低階的細節，藉由讓程式設計者使用簡潔的、抽象的函式，用封包當做輸入，並回傳一組新封包。Pyretic 也簡化模組化設計，藉由使用兩種策略結構運算子，也就是平行構成(parallel composition)與順序構成(sequential composition)，讓程式設計者可結合多種不同的策略，而不用擔心產生矛盾。Pyretic 程式設計者亦可建立動態策略，可以隨著時間而改變策略。總之，Pyretic 可以讓 SDN 應用程式設計者使用高階抽象的方式，設計簡潔且模組化的網路應用程式。

2.3 Mininet

Mininet[5][14]是開放源碼的網路模擬器，支援 SDN 架構的 OpenFlow 協定，它是 SDN 研究領域中著名的工具之一。它使用虛擬化的方式來建造網路中虛擬的主機、交換器、控制器和鏈結。如同作業系統將運算資源虛擬化成執行程序(process)一樣，Mininet 在單一作業系統核心上，以程序為基礎執行實際的程式碼，以模擬各個實體，包括標準的網路應用程式、實際的作業系統核心和網路堆疊。因此，在 Mininet 中運作順暢的設計，通常可以直接移轉到實際網路上執行。

Mininet 可以讓使用者很容易取得不同的網路拓樸下的 SDN 網路行為和效能，可以執行複雜的拓樸測試。Mininet 不僅支援基本的參數化拓樸設定，也支援客製化的任意拓樸。Mininet 提供可擴充的 Python API，因此我們甚至能透過這個 API，撰寫 Python 腳本(script)來建立客制化的拓樸。

三、 延伸的 Dijkstra 演算法與其實作

3.1 延伸的 Dijkstra 演算法的描述

給定一有權重的有向圖 $G=(V, E)$和單一源點 s，原始的 Dijkstra 演算法可以回傳從源點 s 到任何其他節點的最短路徑，其中 V 為節點的集合，E 為邊的集合，而每個邊皆有非負的權重。在原始的 Dijkstra 演算法中，節點並沒有設定權重。底下我們將展示延伸的 Dijkstra 演算法如何考慮到邊的權重和點的權重。

圖 4 為延伸的 Dijkstra 演算法，其輸入為給定的有向圖 $G=(V, E)$和單一源點 s，邊的權重設定 ew，及點的權重設定 nw。延伸的 Dijkstra 演算法使用 $d[u]$來儲存目前從源點 s 到終點 u 的最短路徑的距離，使用 $p[u]$來儲存目前最短路徑中到達節點 u 的前一個節點。在初始狀態下，$d[s]=0$, $d[u]=\infty$ (針對所有的節點 $u\in V, u\neq s$)而且 $p[u]=$null(針對所有的節點 $u\in V$)。

延伸的 Dijkstra 演算法與原始的 Dijkstra 演算法類似。不同的是我們在演算法中的第六行與第七行增加了節點的權重項目。依循原始的 Dijkstra 演算法的證明，我們可以證明延伸的 Dijkstra 演算法可以確實的算出從單一源點到其他節點上，包含邊與節點權重的最短路徑。為了節省論文篇幅，我們省略了延伸的 Dijkstra 演算法的正確性證明。

Extended Dijkstra's Algorithm

Input: $G=(V, E)$, ew, nw, s

Output: $d[|V|]$, $p[|V|]$

```
1: d[s]←0; d[u]←∞, for each u≠s, u∈V
2: insert u with key d[u] into the priority queue Q, for each u∈V
3: while (Q≠∅)
4:     u←Extract-Min(Q)
5:     for each v adjacent to u
6:         if d[v] > d[u]+ew[u,v]+nw[u] then
7:             d[v]←d[u]+ew[u,v]+nw[u]
8:             p[v]←d[u]
```

圖 4、延伸的 Dijkstra 演算法

3.2 延伸的 Dijkstra 演算法的應用

延伸的 Dijkstra 演算法在計算從一個特定源點發送封包到其他節點(即終點)的最佳路由路徑(routing path)是非常有用的，尤其是在節點及/或邊上延遲顯著的 SDN 網路環境中。下面我們將展示如何定義邊的權重以及點的權重，使我們所提出的延伸 Dijkstra 演算法可以應用這些權重，為一些特定的 SDN 環境計算出適當的路由路徑。

假設我們可以從 SDN 拓樸得出一個圖 G=(V,E)，此圖必須為有權重的有向圖及連接圖。對於一個點$v \in V$及一個邊$e \in E$，令$Flow(v)$與$Flow(e)$表示為所有經過 v 與 e 的流的集合，分別令$capacity(v)$表示為 v 的容量(意即節點 v 每秒可以處理多少位元)，並令$Bandwidth(e)$表示 e 的頻寬(意即邊 e 每秒可以傳輸多少位元)。節點 v 的權重(延遲)公式定義於等式(1)，邊 e 的權重(延遲)公式定義於等式(2)中。

$$nw[v] = \frac{\sum_{f \in Flow(v)} Bits(f)}{Capacity(v)}, \quad (1)$$

其中$Bits(f)$表示節點 v 目前每秒處理流 f 的位元數。

$$ew[e] = \frac{\sum_{f \in Flow(e)} Bits(f)}{Bandwidth(e)}, \quad (2)$$

其中$Bits(f)$表示邊 e 每秒通過流 f 的位元數。

我們可以很容易地藉由 OpenFlow 交換器流表中的計數區(counter fields)，來幫助我們計算出特定流 f 在節點或者邊上所通過的位元數。

另外需要注意等式(1)和等式(2)的分子單位是“位元數”，分母是“每秒處理的位元數”。因此，節點權重$nw(v)$與邊的權重$nw(e)$的單位都是“秒”。當我們沿著一條路由路徑累加所有節點及邊的權重，便可得到從路徑源點到路徑終點的延遲。

四、實驗模擬

4.1 模擬環境

我們使用 Mininet [5][14]模擬 Abilene 網路[13]的拓樸以進行效能評估。Abilene 網路是由 Internet2 計畫所提出的高性能骨幹網路。圖 5 展示出 Abilene 網路的核心拓樸[16]，一共連接美國各地 11 個區域站點或節點。Abilene 網路在每個相鄰節點間均以 10Gbps 的頻寬連線，以及在每個節點對骨幹外主機均有 100Mbps 的連線。

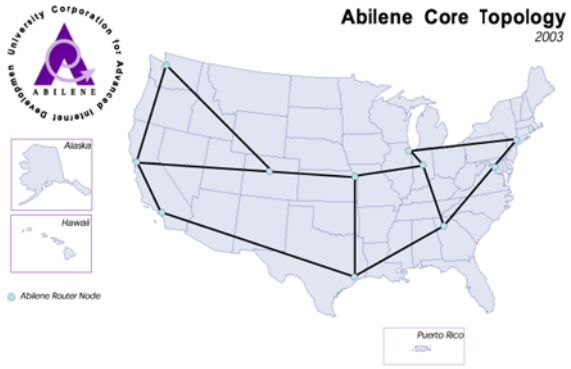

圖 5、Abilene 網路核心拓樸 [16]

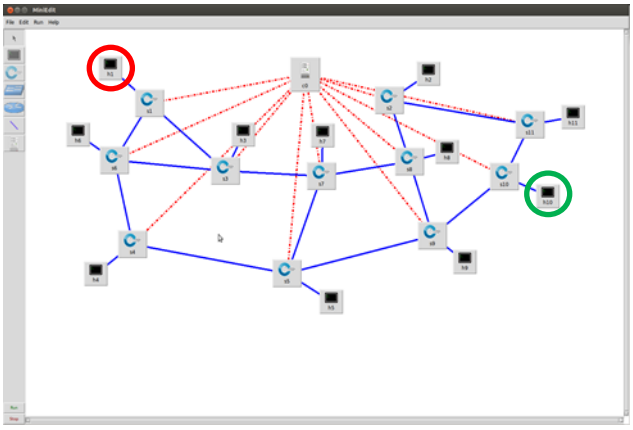

圖 6、Mininet 上建立 Abilene 拓樸

基於 Abilene 的核心拓樸，我們在 Mininet 中建立了一個 SDN/OpenFlow 網路，其中使用 11 台交換器當作其節點並連接著一台控制器，每個交換器也同時連接到一個主機(host)。如圖 6 所示，而模擬參數設置則於表 1 中顯示。

在模擬中，我們使用 Iperf 做為測試工具來產生 53 位元組大小的 TCP 封包，在網路中由 Iperf 客戶端向 Iperf 伺服端傳輸，而每個測試案例的測試時間為 1000 秒。如圖 6 所示，當 Abilene 拓樸在 Mininet 上生成時，我們設定主機 1(紅色圈內)為 Iperf 的伺服端，其餘 10 個主機則為 Iperf 的客戶端。當模擬開始時，所有的客戶端同時使用 Iperf

的命令反覆發送 TCP 封包到主機 1 的 Iperf 伺服端，同時主機 1 使用 Iperf 伺服端的命令接收並回覆所有客戶端。Iperf 測試工具會回報在客戶端與伺服端之間傳輸 TCP 封包的平均延遲時間。在我們的模擬實驗中有 10 個 Iperf 客戶端，我們以其中一個主機，也就圖 6 綠色圈中的主機 10 作為代表，以顯示不同演算法對於延遲時間的影響。

表 1. 模擬環境參數設定

Parameter	Setting
Bandwidth on edges	100Mbps ~ 1Gbps
Capacity of nodes	3Gbps ~ 7Gbps
Number of hosts	10
Number of nodes (switches)	11
Number of edges	25
Controller	POX 2.0 supporting Pyretic
Testing tool	Iperf
Testing time per case	1000 sec

4.2 模擬結果

當模擬進行時，三個最短路徑演算法：原始的 Dijkstra 演算法(original Dijkstra's algorithm)、延伸的 Dijkstra 演算法(extended Dijkstra's algorithm)以及全單位權重的 Dijkstra 演算法(unit-weighted Dijkstra's algorithm)，分別用來產生所有客戶端到伺服端的路由路徑以及伺服端返回所有客戶端的路由路徑。所謂全單位權重的 Dijkstra 演算法，指的是將原始的 Dijkstra 演算法中所有的邊權重都設為 1。因此全單位權重的 Dijkstra 演算法將會產生從源點到終點的最小跳數(hop count)路由路徑。

我們進行兩個模擬案例的模擬，在每個案例中，均針對邊的頻寬以及節點的容量以隨機方式依據表 1 中給定的範圍而設定。模擬結果於圖 7 中呈現，藉由模擬結果，我們可以發現延伸的 Dijkstra 演算法有最短的端對端延遲，而全單位權重的的 Dijkstra 演算法則有最長的端對端延遲。這是因為延伸的 Dijkstra 演算法同時考量了邊與節點的權重，有別於原始的 Dijkstra 演算法只考慮邊的權重及全單位權重的 Dijkstra 演算法只考慮跳數，延伸的 Dijkstra 演算法自然有較短的端對端延遲。

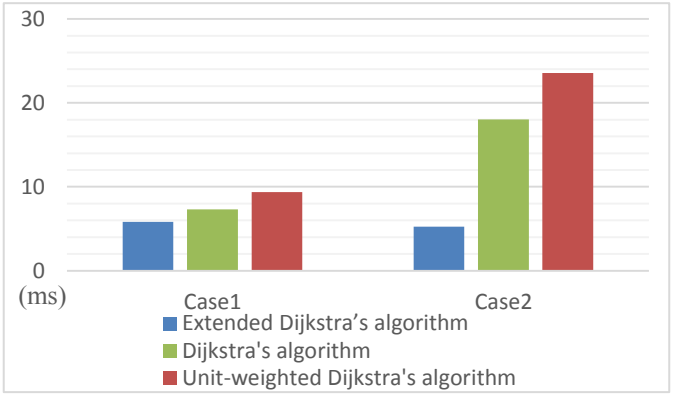

圖 7、各演算法的端對端延遲比較

五、結論

在本篇論文中，我們延伸著名的 Dijkstra 演算法，同時考慮一個網路拓樸圖中邊與節點的權重，以求得單一源點到其他節點的最佳(最短)路由路徑取。我們也使用 Pyretic 語言實作延伸的 Dijkstra 演算法，並透過 Mininet 以及 Iperf 等工具，在 Abilene 網路拓樸中，針對延伸的 Dijkstra 演算法、原始的 Dijkstra 演算法以及全單位權重的 Dijkstra 演算法進行端對端傳輸延遲的模擬與比較。如模擬結果所示，延伸的 Dijkstra 演算法具有最佳的效能。

在未來，我們預計針對更多 SDN 網路拓樸與情境案例，透過更多類型的測量及模擬工具，進行各種不同的網路效能模擬，以顯示延伸的 Dijkstra 演算法的優點。我們也預計在實體的 SDN 網路上實作延伸的 Dijkstra 演算法，並實際測試其網路效能。

參考文獻

[1] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no.1, 1959, pp. 269-271.

[2] D. Drutskoy, E. Keller, and J. Rexford, "Scalable network virtualization in software-defined networks," *IEEE Internet Computing*, vol. 17, no. 2, 2013, pp. 20-27.

[3] N. Foster, R. Harrison, M. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A Network Programming Language", in *Proc. of the 16th ACM*

SIGPLAN International Conference on Functional Programming, 2011, pp 279-291.

[4] A. Furculita, M. Ulinic, A. Rus, and V. Dobrota, "Implementation issues for Modified Dijkstra's and Floyd-Warshall algorithms in OpenFlow," in *Proc. of 2013 RoEduNet International Conference 12th Edition: Networking in Education and Research*, 2013, pp. 141-146

[5] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks," in *Proc. of ACM Hotnets'10*, 2010.

[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication*, 2008.

[7] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing Software-Defined Networks," in *Proc. of NSDI*, 2013.

[8] B. Nunes, M. Mendonça, X. Nguyen, K. Obraczk, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials*, vol. 16, issue 3, pp. 1617-1634, 2014.

[9] Open Networking Foundation, "OpenFlow Switch Specification version 1.4.0," October 14, 2013.

[10] A. Rus, V. Dobrota, A. Vedinas, G. Boanea, and M. Barabas, "Modified Dijkstra's algorithm with cross-layer QoS," *ACTA TECHNICA NAPOCENSIS, Electronics and Telecommunications*, vol. 51, no. 3, 2010, pp. 75-80.

[11] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, "Modular SDN Programming with Pyretic", *Technical Reprot of USENIX*, available at http://*www.usenix.org*, 2013.

[12] Abilene Network, http://en.wikipedia.org/wiki/Abilene_Network-#cite_note-line-1, last accessed on March 4, 2014.

[13] Mininet Website, http://mininet.org/, last accessed on May 2014.

[14] Open Network Foundation (ONF) Website (SDN white paper), https://www.opennetworking.org/sdn-resources/sdn-definition, last accessed on January 2014.

[15] Historical Abilene Connection Traffic Statistics, http://stryper.uits.iu.edu/abilene/, last accessed in March 2014.

[16] Iperf, http://iperf.fr/, last accessed in May 2014.