建構多重 Steiner 樹用於軟體定義網路群播

江振瑞
國立中央大學
資訊工程學系

陳思源
國立中央大學
資訊工程學系

摘要

本論文提出一個啟發式演算法以建構一個 Steiner 樹集合，用在軟體定義網路(software-defined networking, SDN)架構之下的多重源端群播(multiple-source multicast)，其中每個源端均關連於一群接收者。所提的啟發式演算法是基於延伸 Dijkstra 最短路徑演算法與修正的選擇最近終端優先 Steiner 樹演算法。前者不僅考慮邊的權重，也考慮節點的權重，以形成具有最短延遲的資料路由路徑。後者則選取離某一個 Steiner 樹節點具有“最短的最短路徑”的不在樹上的節點來擴展 Steiner 樹，如此可以減少 Steiner 樹邊的數量，降低頻寬的消耗，但同時也不會造成太大的延遲提昇。我們利用 EstiNet 仿真器，搭配 Ryu 控制器在不同的群播場景模擬所提的演算法，並與其他相關的演算法在源端至接收者延遲和頻寬消耗進行比較，以顯示所提出演算法的優點。

關鍵詞：軟體定義網路、群播、Dijkstra 最短路徑演算法、Steiner 樹

1. 緒論

軟體定義網路(Software-Defined Networking, SDN)由開放式網路基金會(Open Network Foundation, ONF)[1]提出，是一個將網路設備控制面(control plane)和資料面(data plane)分離的新型網路架構。SDN 交換器負責資料面的封包轉發，而一個 SDN 控制器或一組 SDN 控制器則負責從交換器收集網路狀態資與配置交換器轉發表(forwarding table)。轉發表也稱為流表(flow table)，所有的交換器都是根據流表作為處理資料封包的基礎。透過控制器修改交換器流表的方式，SDN 使用者可以有設計出在控制器上執行的應用程式，以集中且及時的方式監控與管理整個網路。

Google 在它的私有廣域網路 B4[2]上應用了 SDN 技術，以改善 B4 網路效能。在 SDN 架構下，藉由集中式流量工程(traffic engineering, TE)，讓網路鏈路利用率(utilization)從 30-40%提升到接近 100%。除了在 Google B4 網路的應用外，還存在許多以 SDN 為基礎的應用[3][4]，諸如負載平衡(load balancing)、存取控制(access control)及群播(multicast)。

群播是一個基本通信功能，使用群播功能可以讓一個源端(source)發送資料封包，經過中間設備複製以轉發到多個輸出鏈路，並最終送達群播群組(multicast group)中的所有接收者。它可應用於如 IPTV、視訊會議及串流直播(live streaming)等。群播路由(multicast routing)可視為是建構一棵以源端為根節點，涵蓋所有接收者的群播樹(multicast tree)。PIM-SM 協定[5]針對每一個接收者找出一條從源端到接收者的最短路徑，並將所有路徑聯合形成一棵最短路徑樹(shortest path tree)以作為群播樹。然而，這個最短路徑樹可能包含許多消耗大量頻寬的鏈路。反之，最小 Steiner 樹[6]包含最小數量的鏈路以涵蓋所有的接收者；因此它可作為建立消耗較少頻寬群播樹的良好選擇之一。

本論文提出一個啟發式演算法，藉由結合最短路徑樹及最小 Steiner 樹，以建構一組群播樹。這些被建構的樹在 SDN 架構下被應用於擁有多重源端的群播情境。該演算法採用延伸 Dijkstra 演算法[7]與選擇最近終端優先演算法 Selective Closest Terminal First, SCTF)[8]兩者的優點。延伸 Dijkstra 演算法將網路建模成一個有向圖，它不僅考慮邊的權重，也考慮節點的權重，以獲得從單一節點至其它每個節點的最短路徑。SCTF 演算法使用一個啟發(heuristic)建構一個近乎最小總成本(權重)的 Steiner 樹。本論文修改這個啟發以適合 SDN 的群播場景。我們利用 EstiNet 仿真器，搭配 Ryu 控制器[10]在不同的群播場景模擬所提的演算法，並與其他相關的演算法在源端至接收者延遲和頻寬消耗進行比較，以顯示所提出演算法的優點。

本論文的其餘部分安排如下：第二節介紹了一些相關研究；所提出的演算法在第三節詳述；第四節展示該演算法的模擬結果；最後，在第五節總結全文。

2. 相關研究

2-1 SDN

SDN 提倡將控制面與資料面分離的概念，以使底層稱為交換器(switch)的資料面交換裝置被一個上層稱為控制器(controller)的集中式設備控制，而在控制器之上，可以運行各式各樣的應用程式(applications)。圖 1 描繪了 SDN 的架構邏輯圖。SDN 允許網路管理者編寫應用程式透過北向介面(northbound interface)與控制器相互合作，而控制器則透過南向介面(southbound interface)與交換器交互作用以提供網路服務，包含路由、存取控制、負載平衡、群播及其它流量工程(traffic engineering)工作。

OpenFlow[12]是使用在控制器與交換器之間最知名的南向介面協定。如圖 2 所示，一個交換器具有一或多個流表及/或群組表，而一個流表項目包含匹配欄位、計數器及指令等。一個控制器可以在流表中被動地及主動地編輯、新增和刪除流表項目。

當接收到一個封包時，交換器首先將封包表頭與在流表中每個項目的匹配欄位進行匹配。匹配過程開始於第一個表，並隨後繼續進行到其他的表。匹配過程是具有優先次序的，也就是說，只要找到匹配項目，即會回傳這個項目，並且終止匹配過程。如果找到了匹配項目，與這個項目相關的計數器便會更新以作為稍後的統計之用，而相關指令將會執行，以完成特定的動作，諸如透過一個輸出端口轉發封包至其它交換器，或丟棄這個封包等。如果在任何流表中均未找到匹配，則丟棄該封包或轉發至控制器。

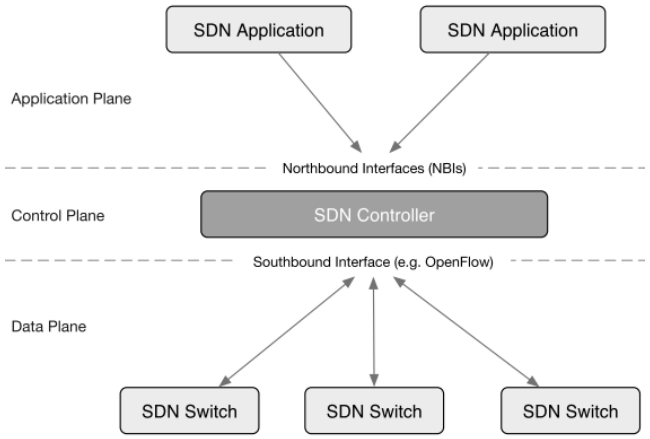
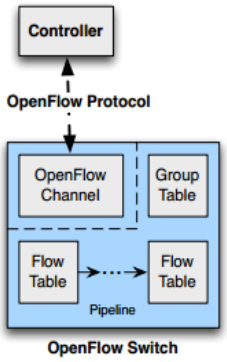


圖 1. SDN 架構示意圖[11]



Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flags

圖 2. OpenFlow 控制器、交換器及流表項目[12]

2-2 延伸 Dijkstra 演算法

考慮一個加權連通有向圖$G=(V,E)$及單一源端節點s，其中V是節點集合，E是邊集合，而且每一個邊均賦予一個非負權重(成本)。原始的 Dijkstra 演算法[13]可以針對每一個節點，傳回從源端節點 s 至每一個節點的最短路徑。在傳統的 Dijkstra 演算法中，節點並未賦予權重。文獻[7]提出延伸的 Dijkstra 演算法(Extended Dijkstra Algorithm, EDA)，同時考慮邊的權重與節點的權重來建立最短路徑。延伸的 Dijkstra 演算法也被應用在文獻[14]，以實現負載平衡與群播。

圖 3 顯示延伸 Dijkstra 演算法(EDA)，它的輸入是一個加權有向圖 G=(V, E)、邊的權重設定 ew、節點的權重設定 nw 以及單一源端節點 s。EDA 使用 $dist[u]$儲存目前從源端節點 s 至目的節點 u(即接收者)的最短路徑距離，並且使用 $pred[u]$儲存節點 u 在目前最短路徑上的前一個節點。最初，對每一個 $u\in V$，$u\neq s$ 而言，$dist[s]=0$，$dist[u]=\infty$；對每一個 $u\in V$ 而言，$pred[u]=null$。這個演算法最終根據 $dist[u]$與 $pred[u]$計算並回傳一個從源端節點 s 到所有其他節點最短路徑的集合 SP。

在 SDN 環境中，若其中封包通過中間節點與邊(也就是網路鏈路)會發生顯著不同延遲，則 EDS 非常適合用於此種環境以推導從特定源端節點發送一個封包至特定目的節點的最佳路由路徑。假設 SDN 網路被建模成一個連通有向圖 G=(V, E)，文獻[7]定義了邊與節點權重，描述如下。對一節點 $v\in V$ 及一邊 $e\in E$ 而言，令 $Flow(v)$與 $Flow(e)$表示分別通過 v 與 e 所有流量的集合，令 $Capacity(v)$表示 v 的容量（即節點 v 每秒可處理的位元數），$Bandwith(e)$表示 e 的頻寬(即邊 e 每秒可傳送的位元數)。節點 v 的權重 $nw[v]$ 被定義如式(1)，且邊 e 的權重 $ew[e]$被定義如式(2)。

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

其中 $Bits(f)$是節點 v 每秒處理的流量 f 的位元數。

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

其中 $Bits(f)$是邊 e 每秒通過的流量 f 的位元數。

請注意，我們可以透過 OpenFlow 交換器流表計數器的協助，很容易地獲得一個流通過一個邊或通過一個節點的位元數。另外請注意，在式(1)與式(2)分子的單位是位元數，而分母的單位是每秒位元數。因此，節點權重 $nw[v]$ 與邊權重 $ew[e]$的單位是“秒”。當我們延著一條路徑累加所有節點與所有邊的權重，我們可以求得從路徑的一個端點(如源端節點)至其它端點(如接收者）的延遲。

Algorithm: EDA (Extended Dijkstra's Algorithm)

Input: G=(V, E), ew, nw, s //G =(V, E) is a digraph with edge and node weights stored in ew and nw, and s is the source

Output: SP //SP is the set of shortest paths from s to all other nodes

```
1:  dist[s]←0; dist[u]←∞, for each u≠s, u∈V
2:  insert u with key dist[u] into priority queue Q, for each u∈V
3:  while (Q≠∅)
4:      u←Extract-Min(Q)
5:      for each v adjacent to u
6:          if dist[v] > dist[u]+ew[u,v]+nw[u] then
7:              dist[v]←dist[u]+ew[u,v]+nw[u]
8:              pred[v]←u   //v's predecessor in the shortest path is u
9:  calculate the shortest path from s to u to add into set SP according to
    dist[u] and pred[u], for each u∈V, u≠s
10: return SP
```

圖 3. 延伸 Dijkstra 演算法

2-3 最小 Steiner 樹演算法

考慮一個無向加權圖 $G=(V, E)$，與一個包含所有終端節點(*terminal*)的集合 R，其中 $R \subseteq V$。最小 Steiner 樹問題意指找出稱為最小 Steiner 樹(minimum Steiner tree)的 G 的子圖，該子圖形成一棵樹且具有最小權重，並涵蓋所有在 R 中的終端節點。當 $R=V$，最小 Steiner 樹其實就是最小生成樹。此外，當每個邊的成本為 1，最小 Steiner 樹即為具有最小數目的邊而且涵蓋所有終端節點的樹。最小 Steiner 樹問題已經被證明是 NP-hard 問題。因此，不太可能存在多項式時間複雜度演算法來解決這樣的問題。然而，已有許多多項式時間複雜度啟發式演算法被提出來，可以求出這個問題的近似最佳解。

SCTF 演算法[8](如圖 4 所示)是解決最小 Steiner 樹問題的啟發式演算法之一。本論文因為下述理由而專注於 SCTF 演算法。第一，此演算法視網路為有向圖(而不是原始問題中的無向圖)來解決問題。因此，它可以應用於兩個節點之間的邊可以是非對稱的(比方說，邊可以具有不同頻寬)更實際的網路環境。第二，該演算法被參數化，用以在快速演算法執行時間與低權重 Steiner 樹之間進行權衡。

Algorithm: SCTF (Selective Closest Terminal First) algorithm

Input: $G=(V, E)$, ew, s, $R=\{r_1,\ldots,r_n\}$ and κ //$G=(V, E)$ is a graph with edge weights stored in ew, s is the source, R is the group (set) of receivers associated with s, and κ is a control knob for the priority queue
Output: $T=(V_T, E_T)$ //T=(V_T, E_T), where $V_T \subseteq V$ and $E_T \subseteq E$, is a Steiner tree rooted at s and spanning all nodes in R

```
1:  Q←{s}, V_T←{s}, E_T← ∅    //Q: priority queue
2:  while (R≠∅) do
3:      B ← the set of the first Min(κ, |Q|) nodes in Q
4:      P* ← ShortestPath(x, y), where x∈B and y∈R are arbitrary
5:      for each x in B do
6:        for each y in R do
7:            if   w(P ← ShortestPath(x, y)) < w(P*)
8:                 P* ← P    //P* is the "shortest" shortest path
9:      z ← the terminal at which P* terminates
10:     Branch ← subpath(u, z) such that only u is in V_T
11:     insert nodes in Branch into Q
12:     V_T ← V_T ∪ {nodes in Branch}
13:     E_T ← E_T ∪ {edges in Branch}
14:     R_T ← R_T − {terminals in Branch}
15: return T
```

圖 4. SCTF 演算法

SCTF 演算法的基本概念描述如下：SCTF 演算法首先找出從源端節點到每個終端節點的最短路徑，然後找出“最短的最短路徑”P^*，並將在 P^*中的所有節點都加入到 Steiner 樹 T。緊接著，SCTF 演算法根據下列優先順序：源端節點>終端節點>非終端節點，將在 P^*中的所有節點放入優先佇列 Q 中。然後針對 Q 中最前面 κ(kappa)個節點的每一個節點找出到所有尚未包含在 Steiner 樹中的終端節點的最短路徑。如此，可以再次找出所有最短路徑中具有最小總加權的路徑，也就是“最短的最短路徑”P^*。令 z 是與 P^*相關聯的終端節點，且令 *Branch* 是一條由節點 u 到達節點 z 的 P^*的子路徑，而其中 u 是 Branch 中唯一屬於 T 的節點。請注意，“最短的最短路徑”P^*計算僅在 Q 中最前面的κ個節點進行，所以P^*可能會通過T中的某些節點，因此需要被修剪成 *Branch*。在每次迭代中，SCTF 演算法將 *Branch* 中的節點與邊加入 T 中，直到 T 包含所有終端節點為止。值得一提的是 κ 被用於限制 SCTF 演算法的計算量。較大的 κ 會導致較大的計算量，但是較大的 κ 值也通常會產出更好的 Steiner 樹結果。

3. 所提演算法

本論文所提出的演算法稱為 M-SCTF/EDA，代表它是一個修改的 SCTF 演算法，而且也結合了 EDA 演算法。它是針對多重源端的場景，其中的每一個源端與一個具有許多群播接收者的群組相關聯。它構成一個多重 Steiner 樹的集合，其中每棵樹以一個源端為根，並涵蓋與該源端相關聯群組的所有接收者。此演算法的目的是找到多重 Steiner 樹，以儘量降低平均源端至接收者延遲與平均頻寬消耗。

以下描述兩個用在所提演算法的重要量測。第一個量測是源端至接收者延遲，定義於式(3)。

$$\frac{\sum_{r\in R} Delay(r)}{|R|} \tag{3}$$

在式(3)中，$Delay(r)$是接收者 r 與它相關聯的源端之間的延遲，且 R 是所有接收者的集合。

第二個量測是頻寬消耗，定義於式(4)。

$$\frac{\sum_{e\in E}\sum_{f\in Flow(e)} Bits(f,e)}{\sum_{e\in E} Bandwidth(e)} \tag{4}$$

在式(4)中，E 是所有邊的集合，$Flow(e)$是所有通過邊 e 的流的集合，且 $Bits(f, e)$則表示流 f 每秒通過邊 e 的位元數。

請注意，頻寬消耗是介於 0 和 1 之間的比率，其代表每秒傳送位元數與總頻寬的比率。另外也請注意，頻寬消耗並不等於所有邊的平均邊(鏈路)利用率。

圖 5 中顯示 M-SCTF/EDA 演算法，其具有 SCTF 演算法與延伸 Dijkstra 演算法兩者的優點，可以使用在多重源端群播的場景，並儘量降低源端至接收者延遲與頻寬消耗。集合 M 啟始的時候包含所有的接收者群組，而 M-SCTF/EDA 該演算法針對每個 M 中的接收者群組建構一棵 Steiner 樹。參照圖 5 的第 3 行，一個屬於集合 M 的接收者群組 R_a 會首先被選擇來建構 Steiner 樹，假如 R_a 是 M 中具有最短的由某源端至某接收者的“最短路徑”的接收者群組。然後，該演算法從 M 中移除 R_a，並且遵循 SCTF 演算法的相同概念來構成一個以 s_a 為根，且涵蓋所有 R_a 中所有節點的 Steiner 樹(參照圖 5 的第 5 至第 17 行)。假如偵測到 M 是空的，演算法將終止，並且回傳一個 Steiner 樹的集合 T。

請注意，從源端至接收者的最短路徑將藉由延伸 Dijkstra 演算法進行計算，也就是是會同時考慮節點權重與邊權重，以獲得更適合實際情況下的最短路徑。在

M-SCTF/EDA 演算法中，延伸 Dijkstra 演算法的呼叫被簡略為 EDA(x, y)，代表這個呼叫會直接回傳從節點 x 至節點 y 的最短路徑。

在 M-SCTF/EDA 演算法中，優先佇列中節點的優先順序被修正為：源端>非終端節點(離源端跳數較小者優先)>終端節點。由於優先權的修改，源端至接收者延遲可能會因此而減少，但是頻寬消耗卻不一定會增加。這可幫助 M-SCTF/EDA 演算法構成較低源端至接收者延遲與較低頻寬消耗的 Steiner 樹。

Algorithm: M-SCTF/EDA

Input: $G=(V, E)$, ew, nw, n, $S=\{s_1,\ldots,s_n\}$, $M=\{R_1,\ldots,R_n\}$, κ
//$G=(V, E)$ is a graph with edge and node weights stored in ew and nw, n is the number of sources, R_i is the i^{th} group (set) of receivers associated with s_i, $1\le i\le n$, and κ is a control knob for the priority queue
Output: $T=\{T_1,\ldots,T_n\}$ //T is a set of Steiner trees, where $T_i=(V_i, E_i)$, $V_i\subseteq V$ and $E_i\subseteq E$, is a tree rooted at s_i and spanning all nodes in R_i, $1\le i\le n$

```
1:  Q_i ← {s_i}, V_i ← {s_i}, E_i ← ∅, for every i, 1≤ i ≤ n;
    //Q_i: i^th priority queue
2:  while (M≠∅) do
3:      r_a ← Arg Min_{r ∈ (∪_{R_j∈M} R_j)} w(EDA(r's source, r)), where r_a∈R_a
4:      M ← M − R_a
5:      while (R_a≠∅) do
6:          B ← the set of the first Min(κ, |Q|) nodes in Q
7:          P* ← EDA(x, y), where x∈B and y∈R_a are arbitrary
8:          for each x in B do
9:              for each y in R_a do
10:                 if  w(P ← EDA(x, y)) < w(P*)
11:                     P* ← P   // P* is the "shortest" shortest path
12:         z ← the terminal at which P* terminates
13:         Branch ← subpath(u, z) such that only u is in V_a
14:         insert nodes in Branch into Q_a
15:         V_a ← V_a ∪ {nodes in Branch}
16:         E_a ← E_a ∪ {edges in Branch}
17:         R_a ← R_a − {terminals in Branch}
18: return T
```

圖 5. 本論文提出的 M-SCTF/EDA 演算法

4. 效能評估

我們利用 EstiNet 仿真器，搭配 Ryu 控制器在兩個不同的群播場景模擬 M-SCTF/EDA 演算法，並與其他相關的演算法在源端至接收者延遲和頻寬消耗進行比較。兩個模擬場景的參數設置如表 1 所示，其中一些參數是根據現成產品的規格而設定的，這些產品包括 NEC Programmable-Flow PF5248 交換器、Xinguard Pica8 3290 交換器及 HP 3500 系列交換器。這兩個場景的拓樸圖分別描述於圖 6 與圖 7，其中節點 21 是控制器。第一個場景只有 1 個源端(節點 40)及 18 個接收者，而第二個場景有 2 個源端(節點 40 與節點 23)，其各有 8 個接收者。

模擬的其他參數描述如下，群播資料是 UDP 封包，以固定位元速率(constant bit rate) 2500 kbps (20 MB/minute) 進行傳送，也就是 H.264 high profile HD 720p 影片資料的傳輸率。而對所有模擬情況而言，優先佇列的控制旋鈕參數 κ 均設置為 4。

表 1. 場景 1 及 2 的參數設定

Parameter	Setting
Bandwidth on edges	100Mbps ~ 1Gbps
Capacity of each node (switch-	10Gbps ~ 179Gbps
Number of sources	1 or 2
Number of receivers	18 or 16
Number of switches	20
Number of edges	63
Controller	Ryu ver 1.7.90
Simulation time per case	1000 sec

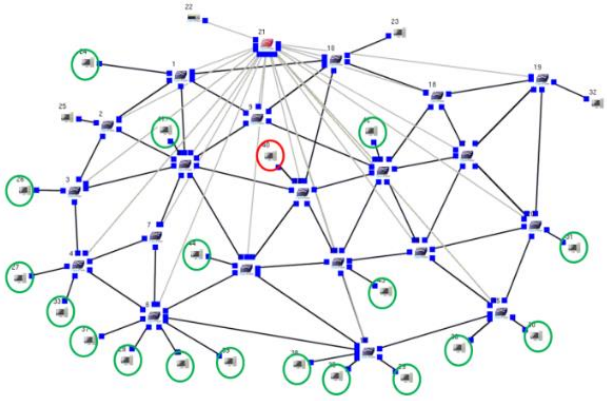


圖 6. 模擬場景 1

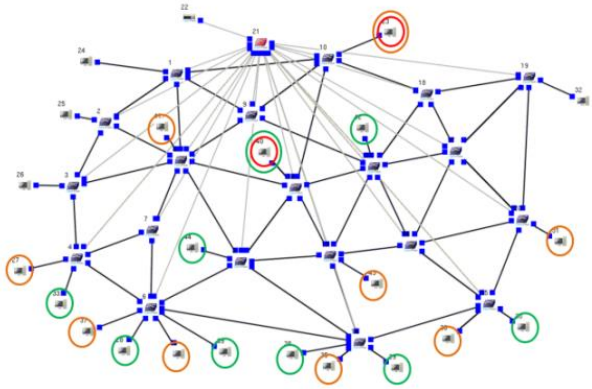


圖 7. 模擬場景 2

我們不僅模擬所提出的 M-SCTF/EDA 演算法，而且也模擬 SCTF/EDA、SCTF、 M-SCTF、DA(即原始的 Dijkstra 演算法)、EDA 以及 MST 演算法(也就是使用窮舉方式找出使用最少數量的邊去涵蓋所有接收者的最小 Steiner 樹演算法）以進行比較。如圖 8 所示，所提出的 M-SCTF/EDA 演算法擁有非常好的效能，它具有第二低的源端至接收者延遲與頻寬消耗。在源端至接收者延遲方面，M-SCTF/EDA 演算法僅劣於完全專注在最短延遲的 EDA 演算法。然而，EDA 演算法具有非常高的頻寬消耗。在頻寬消耗方面，M-SCTF/EDA 演算法僅劣於完全專注在最低頻寬消耗的 MST 演算法。然而，MST 演算法是一種窮舉演算法，耗

費大量的計算，而且具有非常高的源端至接收者延遲。我們可以說，當源端至接收者延遲與頻寬消耗兩者同時被考慮，所提出的 M-SCTF/EDA 演算法是適用於 SDN 架構以實現多重源端群播最好的演算法。

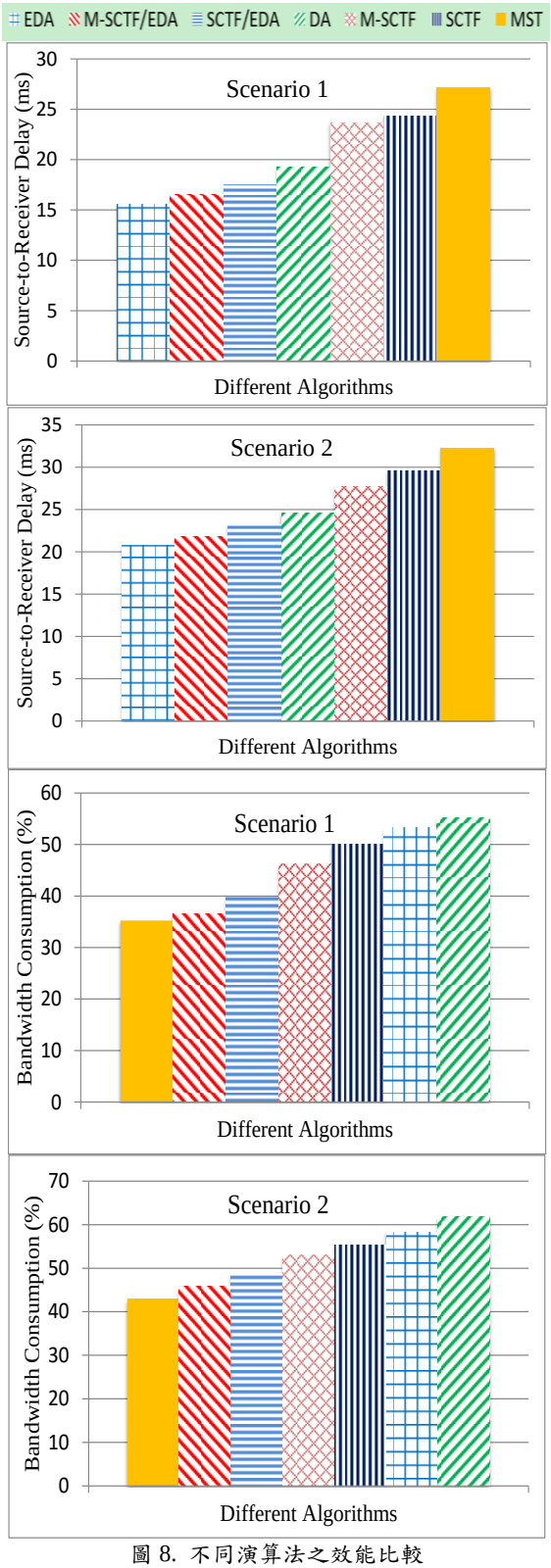


圖 8. 不同演算法之效能比較

5. 結語

本論文所提出的 M-SCTF/EDA 演算法利用延伸 Dijkstra 最短路徑演算法，並修改選擇最近終端優先(SCTF)Steiner 樹演算法來建構一個適合用於多重源端群播的 Steiner 樹集合，以達成儘量降低源端至接收者延遲與頻寬消耗的目標。我們利用 EstiNet 仿真器，搭配 Ryu 控制器在兩個不同的群播場景模擬所提的 M-SCTF/EDA 演算法並與其他相關演算法在源端至接收者延遲與頻寬消耗兩方面進行比較。藉由比較結果，我們可以觀察到所提出的 M-SCTF/EDA 演算法是唯一在所有比較的演算法之中在各方面均表現得很好的演算法。

我們已經觀察到，如同 Dijkstra 演算法一樣，Floyd-Warshall 演算法[15]也可以輕易地擴展為同時考慮邊權重與節點權重的演算法。因此，如果我們改寫 M-SCTF/EDA 演算法中兩個內層 for 迴圈，並且以“擴展 Floyd-Warshall 的演算法”取代 EDA 演算法，則所提出的 M-SCTF/EDA 演算法可以在計算負載方面得到改善。這是因為 Dijkstra 演算法是一個一對全部最短路徑演算法，Floyd-Warshall 演算法是一種全配對最短路徑演算法，而且兩個內部 for 迴圈實際上是用於在所有配對的最短路徑中尋找“最短的最短路徑”，我們計畫在未來實現這個改善。

近來，許多關於 SDN 群播的研究被提出，它們專注於群播的不同面向，諸如容錯[16]、可擴展性[17][18]及負載平衡[19][20]等。本論文所提出的演算法並未與那些演算法比較，因為他們強調不同於源端至接收者延遲與頻寬消耗的面向。在未來，我們計畫考慮更多的面向來探究所提出的演算。我們也計畫應用所提出的演算法到更實用的群播場景，如串流直播與視訊會議等。

6. 參考文獻

[1] Open Network Foundation (ONF). https://www.opennetworking.org/sdn-resources/sdn-definition (last accessed on March 2016).

[2] Jain, S., et al. 2013. B4: Experience with a globally-deployed software defined WAN. *ACM SIGCOMM Computer Communication Review*. 43, 4, 3-14.

[3]Nunes, B. A., Mendonca, M., Nguyen, X. N., Obraczka, K., and Turletti, T. 2014. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials*. 16. 3. 1617-1634.

[4] Farhady, H., Lee, H., and Nakao, A. 2015. Software-defined networking: A survey. *Computer Networks*. 81. 79-95.

[5] Fenner, B., et al. 2006. Protocol independent multicast - sparse mode (pim-sm): protocol specification (revised). *IETF RFC 4601*.

[6] Hwang, F. K., Richards, D. S., and Winter, P. 1992. *The Steiner tree problem*. Elsevier.

[7] Jiang, J. R., Huang, H. W., Liao, J. H. and Chen, S. Y. 2014. Extending Dijkstra's shortest path algorithm for software defined networking. In *Proc. of 16th IEEE Asia-Pacific Network Operations and Management Symposium (APNOMS)*. 1-4.

[8] Ramanathan, S. 1996. Multicast tree generation in networks with asymmetric links. *IEEE/ACM Transactions on Networking (TON)*. 4. 4. 558-568.

[9] Wang, S. Y., Chou, C. L., and Yang, C. M. 2013. EstiNet OpenFlow network simulator and emulator. *IEEE Communications Magazine*. 51. 9. 110-117.
[10] Ryu OpenFlow Controller. URL: http://osrg.github.io/ryu/
[11] Banse, C, and Rangarajan, S. 2015. A secure northbound interface for SDN applications. In *Proceedings of the* 2015 *IEEE Conference on* Trustcom/BigDataSE/ISPA.
[12] Open Networking Foundation. 2015. *OpenFlow Switch Specification version 1.5.1*.
[13] Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische mathematik*, 1. 1. 269-271.
[14] Jiang, J.-R., et al. 2014. Load balancing and multicasting using the extended Dijkstra's algorithm in software defined networking. In *Proc. of the International Computer Symposium 2014 (ICS 2014)*.
[15] Floyd, R. W. 1962. Algorithm 97: Shortest path. Communications of the ACM. 5. 6. 345.
[16] Pfeiffenberger, T., et al. 2015. Reliable and flexible communications for power systems: Fault-tolerant multicast with SDN/OpenFlow. In *Proc. of the 7th IEEE International Conference on New Technologies, Mobility and Security (NTMS)*. 1-6.
[17] Cui, W., and Qian, C. 2015. Scalable and load-balanced data center multicast. In *Proc. of 2015 IEEE Global Communications Conference (GLOBECOM)*. 1-6.
[18] Zhou, S., Wang, H., Yi, S., and Zhu, F. 2015. Cost-efficient and scalable multicast tree in software defined networking. *Algorithms and Architectures for Parallel Processing*. 592-605.
[19] Iyer, A., Kumar, P., and Mann, V. 2014. Avalanche: Data center multicast using software defined networking. In *Proc. of the 6th IEEE International Conference on Communication Systems and Networks (COMSNETS)*. 1-8.
[20] Craig, A., Nandy, B., Lambadaris, I., and Ashwood-Smith, P. 2015. Load balancing for multicast traffic in SDN using real-time link cost modification. In *Proc. of IEEE International Conference on Communications (ICC)*. 5789-5795.