Journal of Science and Engineering Technology, Vol. 5, No. 3, pp. 51-59 (2009)

MPLS 單點對多點傳輸之快速回復機制

黃翰隆[1]　翁永昌[2]　黃鈴玲[3]

[1]中央研究院資訊科學研究所

11529 台北市南港區研究院路二段 128 號

[2]靜宜大學資訊工程學系

43301 台中縣沙鹿鎮中棲路 200 號

[3]大葉大學資訊工程學系

51591 彰化縣大村鄉學府路 168 號

摘　要

多協定標籤交換（multiprotocol label switching, MPLS）是一個以標籤爲依據的封包繞送技術。在 MPLS 單點對多點傳輸（point-to-multipoint, P2MP）的回復機制方面，區域修復（local repair）與路徑保護（path protection）是兩個最常見的方法。區域修復能夠快速地從錯誤中回復，但需要花費相當多的備援頻寬；相反的，路徑保護需要較少的備援頻寬但是回復的速度較慢。我們提出一個新的備援機制「通報延遲妥協」（notification delay-compromised, NDC）。NDC 能夠以些微的路徑切換延遲，來換取有效地節省備援頻寬。模擬實驗結果顯示，當目的節點數目介於 2-5 時，NDC 能在多付出 1-2.6 跳躍（hop）延遲的情況下，比快速重新路由法（fast reroute）方法節省 30% 備援頻寬。

關鍵詞：多協標籤交換，單點對多點傳輸，備援樹，錯誤回復

Survivable Point-to-Multipoint Communications in MPLS Networks

HAN-LUNG HUANG[1], YUNG-CHANG WONG[2] and LINGLING HUANG[3]

[1]Institute of Information Science, Academia Sinica

No. 128, Academia Rd., Sec. 2, Nankang, Taipei, Taiwan 11529, R.O.C.

[2]Department of Computer Science and Information Engineering, Providence University

No. 200, Chung Chi Rd., Taichung, Taiwan 43301, R.O.C.

[3]Department of Computer Science and Information Engineering, Da-Yeh University

No. 168, University Rd., Dacun, Changhua, Taiwan 51591, R.O.C.

ABSTRACT

The Multiprotocol Label Switching (MPLS) backup mechanism is a comprehensive procedure to protect from the failure of routers or links in MPLS networks. The two most common backup and recovery mechanisms are Local Repair and Path Protection. Local Repair can rapidly recover from a crash but must expend more bandwidth; however, Path Protection has less bandwidth expenditure but

a longer notification delay. In this report, we have proposed a novel solution, namely the Notification Delay-Compromising Protection Strategy (NDC), which achieves a balance between bandwidth expenditure and notification delay. The empirical results indicated that the NDC can conserve more bandwidth expenditure by a slight delay in notification.

Key Words**:** multiprotocol label switching (MPLS), point-to-multipoint (P2MP) communications, backup and recovery mechanisms

一、簡介

多協定標籤交換（multiprotocol label switching, MPLS）[6] 是一個以標籤爲依據的封包繞送技術，可以改善封包繞送的效能並提供更有彈性的繞送服務。當封包從 IP 網路進入到MPLS網路時，邊界（edge）標記交換路由器（label switch router, LSR）負責將標籤貼在每個封包的第二層與第三層表頭之間，接著將這些貼有標籤的封包發送至 MPLS 網路中其他的 LSR。因爲封包已經貼上標籤，所以 LSR 就可以根據標籤來轉送封包，而不需要分析第三層的標頭資訊。當封包要離開 MPLS 網路時，edge LSR 須負責將封包上的標籤移除。上述過程中，負責貼上標籤的 LSR 稱爲入口（ingress）；負責移除標籤的 LSR 稱爲出口（egress）。

入口到出口之間建立的路徑稱作標籤交換路徑（label switching path, LSP）。在單點對單點傳輸的應用中，每條 LSP 都僅有一個入口與一個出口。隨著多媒體及視訊會議等需求的增加，單點對單點傳輸已經不敷使用，因此 MPLS 加入了單點對多點（point-to-multipoint, P2MP）傳輸的功能。P2MP 傳輸顧名思義就是每條 LSP 中，有一個入口和一個以上的出口，構成一棵 P2MP 樹。

在 P2MP 的快速回復機制方面，Aggarwal 等人 [1] 提出的方法如下：將一棵 P2MP 樹視爲是由多條標籤交換子路徑（sub-LSP）所組成，每條 sub-LSP 代表由來源節點 S 到某個目的節點 d_i 的最短路徑。每條 sub-LSP 須分別建立備援路徑予以保護。此法的優點是在發生錯誤時，節點 S 可以立即偵測到錯誤，從而把封包導向備援路徑；缺點是節點 S 需將同一個封包沿著每條 sub-LSP 各送一份。爲了減少上述冗贅，Kodialam 和 Lakshman [3] 直接對 P2MP 樹建立備援路徑。Li 等人 [5] 進一步提出備援路徑共享的概念，以降低備援頻寬。在本篇論文中，我們提出折衷的方法稱爲通報延遲妥協（notification delay-compromised, NDC）保護機制，期能犧牲些微的路徑切換時間，以節省可觀的備援頻寬。

本文架構安排如下：第二節將介紹 MPLS 快速回復機制的相關研究，並比較這些方法的優劣。第三節將說明我們提出的 NDC 機制。第四節將說明模擬的環境並分析實驗的結果。最後是結論與未來研究方向。

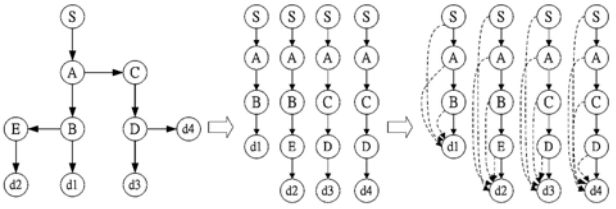
二、相關文獻

本節將介紹 P2MP 快速回復機制：Sub-LSP 快速重新路由、區域修復、以及快速重新路由。

（一）Sub-LSP 快速重新路由（Sub-LSP Fast Reroute）

Aggarwal 等人 [1] 提出 Sub-LSP 快速重新路由（fast reroute）法，將一棵 P2MP 樹視爲是由多條 sub-LSPs 所組成，其中每條 sub-LSP 代表由來源節點 S 到目的節點 d_i 的最短路徑。以圖 1 爲例，圖 1(a) 的 P2MP 樹包含來源節點 S 與目的節點 d_1、d_2、d_3 與 d_4。圖 1(b) 爲對應的 4 條 sub-LSP：$S\rightarrow A\rightarrow B\rightarrow d_1$、$S\rightarrow A\rightarrow B\rightarrow E\rightarrow d_2$、$S\rightarrow A\rightarrow C\rightarrow D\rightarrow d_3$ 和 $S\rightarrow A\rightarrow C\rightarrow D\rightarrow d_4$。

在受保護的 LSP 中，每個節點 X 須建立從自己到 d_i 的備援路徑，以保護 X 的下一個節點 Y 以及鏈結 X-Y。以 sub-LSP $S\rightarrow A\rightarrow B\rightarrow d_1$ 爲例，須建立 $S\rightarrow d_1$、$A\rightarrow d_1$、以及 $B\rightarrow d_1$ 共三條備援路徑（圖 1(c)）。當節點 A 發生錯誤時，上游節點 S 立即可以偵測到錯誤，從而把封包導向備援路徑 $S\rightarrow d_1$、$S\rightarrow d_2$、$S\rightarrow d_3$、以及 $S\rightarrow d_4$；但如此一來，節點 S 必須爲將每一個封包複製 4 份分別送出，這無疑是一種冗贅。



(a) P2MP 樹　(b) sub-LSPs　(c) 備援路徑

圖 1. P2MP sub-LSP 快速重新路由

（二）區域修復（Local Repair）

Kodialam和Lakshman [3] 提出了區域修復(local repair, LR）法以直接對 P2MP 樹建立備援樹。備援樹建立的規則爲：1. 每個節點必須保護與自己最接近的下游節點（downstream node），也就是說必須建立一條備援路徑，『繞過』與自己最接近的下游節點；2. 每個樹葉節點的上游節點，由於其下游節點已是目的地，因此僅須保護自己與樹葉節點間的鏈結。

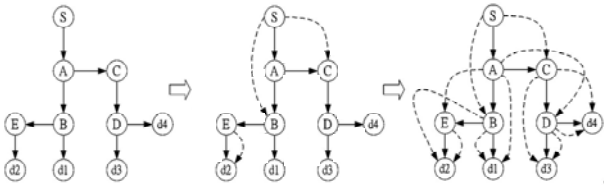
以圖 2(a) 的 P2MP 樹爲例，節點 *S* 最近的下游節點爲 *A*。爲了保護節點 *A*，節點 *S* 利用規則（1）建立備援路徑 *S*→*B* 與 *S*→*C*。至於節點 *E*，因需保護的下游節點爲目的節點 d_2，故利用規則（2）建立備援路徑 *E*→d_2。依照上述的模式，最後完成的備援樹如圖 2(c) 所示。假設節點 *A* 發生錯誤，LR 區域修復方法只需將封包透過 2 條備援路徑 *S*→*B* 及 *S*→*C* 來繞過節點 *A*；相較於 Sub-LSP 方法 [1] 需使用 4 條備援路徑，LR 區域修復方法能減少較多的備援頻寬。

（三）快速重新路由（Fast Reroute）

Li 等人 [5] 在區域修復法中加入頻寬共享的概念，以進一步降低所需的備援頻寬，稱爲快速重新路由法（fast reroute, FRR）。FRR 備援路徑的建立方式如下：

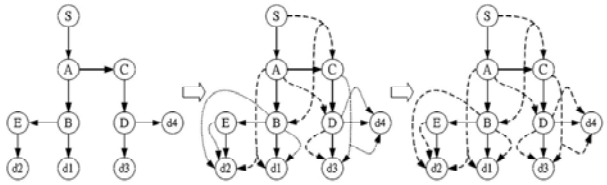
1. 若最接近自己（節點 *X*）的下游節點（節點 *Y*）並非連接樹葉節點的分支節點，則繞過節點 *Y*，與 *Y* 的每一個下游節點（節點 *Z*）都建立備援路徑（*X*→*Z*）。備援路徑若不止一條，彼此之間應盡量共享頻寬；
2. 若最接近自己（節點 *X*）的下游節點（節點 *Y*）爲連接樹葉節點（節點 d_i）的分支節點，則建立備援路徑 *X*→d_i。備援路徑若不止一條，彼此之間應盡量共享頻寬；
3. 若自己（節點 *X*）的子節點爲樹葉節點（節點 d_i），直接建立備援路徑 *X*→d_i。

圖 3 爲建立 FRR 備援路徑的範例。首先說明根節點 *S*。由於 *S* 最接近的下游節點 *A* 並非連接樹葉節點的分支節



(a) P2MP 樹　　(b) local repair　　(c) 最後的備援樹

圖 2. 區域修復（實線表示工作樹，虛線爲備援路徑）



(a) P2MP 樹　　(b) fast re-route　　(c) 最後的備援樹

圖 3. 快速重新路由（實線表示工作樹，虛線爲備援路徑）

點，因此採用規則 1 來建立備援路徑 *S*→*B* 與 *S*→*C*，兩條路徑間盡可能共享頻寬（見圖 3(b)）。接下來說明節點 *A*。最接近節點 *A* 的下游節點有兩個，分別是節點 *B* 與節點 *C*。因爲節點 *B* 爲連接樹葉節點 d_1 與 d_2 的分支節點，因此採用規則 2 來建立 *A*→d_1 與 *A*→d_2 的備援路徑，兩條路徑間盡可能共享頻寬；因爲節點 *C* 並非連接樹葉節點的分支節點，因此採用規則 1 來建立備援路徑 *A*→*D*。最後說明節點 *D*，因爲節點 *D* 的子節點爲樹葉節點 d_3，故採用規則 3 來建立備援路徑 *D*→d_3。圖 3(c) 爲完成後的備援樹。

三、NDC 保護機制

本節將介紹我們提出的通報延遲妥協（NDC）保護機制。NDC 保護機制包含路徑備援與路徑切換兩個部分，將分別在下面兩個小節加以說明。

（一）路徑備援

圖 4 爲採用 NDC 所建立的備援樹，主要是透過來源（source）節點 *S* 以及樹葉節點（leaves）d_i，*i*=1, 2, 3, 4 來建立備援路徑。在圖 4 中，節點 *B* 是距離 *S* 最近的分支節點，我們稱它爲第一層分支節點（tier 1 branching node）。節點 *B* 將整棵 P2MP 樹切割成以節點 *C* 爲根節點的左子樹（T_C）及以節點 *D* 爲根節點的右子樹（T_D），其中樹葉節點 d_1、d_2 屬於 T_C，而節點 d_3、d_4 則屬於 T_D。

爲了確保能夠找到備援路徑，我們假設要探討的 MPLS 網路 *G* 爲 2-connected：當任意移除一個節點，剩下的子網路仍保持連通性（connected）；換句話說，網路 *G* 中任兩節點之間至少存在兩條不共點的路徑 [2]。

NDC 建立備援樹的過程包含下面兩個階段：

1. Phase A-I：建立來源節點 *S* 到第一層分支節點 *B* 的子節點 *C* 和 *D* 之間的備援路徑（圖 4 虛線所示）。以工作路徑 P_1: *S-A-B-D* 爲例。因爲我們假設 MPLS 網路爲 2-connected，這意謂了在節點 *S* 與節點 *D* 之間，除了路

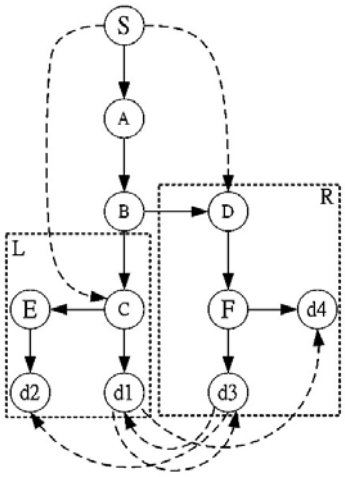


圖 4. NDC 備援樹（虛線部分）

徑 P_1 尚存在另外一條（備援）路徑 P_2: $S \rightarrow D$，其中 P_1 和 P_2 不共點。當路徑 P_1 中的節點 B（或節點 A）發生錯誤時，節點 S 與節點 D 仍可透過備援路徑 P_2 連通。

2. Phase A-II：針對節點 d_1，挑選另一棵子樹 T_D 中與 d_1 最接近的節點 d_3，建立 $d_3 \rightarrow d_1$ 之單向備援路徑（圖 4）。

依此類推，其他的備援路徑尚有 $d_3 \rightarrow d_2$、$d_1 \rightarrow d_3$、以及 $d_1 \rightarrow d_4$。

以下將分別說明 Phase A-I 的演算法 NDC_A-I（圖 5）與 Phase A-II 的演算法 NDC_A-II（圖 6），表 1 和表 2 爲演算法對應的符號說明。

表 1. NDC_A-I 演算法之符號說明

s	入口（ingress）s，爲 P2MP 樹的起點。
T	現行的 P2MP 樹 T。
G	MPLS 網路拓樸所成之圖。
P_b	NDC_A-I 演算法欲建立之備援路徑集合。
v_{t1}	T 中的第一層分支節點。
children(v_{t1})	v_{t1} 所有子節點形成的集合。
Path(s, v, T)	P2MP 樹 T 上從 s 到節點 v 的路徑。
G'	圖 G 減去 *Path*(s, v, T) 所成的圖。
Shortest-path(s, v, G')	圖 G' 中從節點 s 到節點 v 的一條最短路徑。

```
NDC_A-I (s, G, T)          // 找尋 s 至 tier 1 branching node 子節點的備援路徑
// s: the source, G: the underlying graph, T: the P2MP tree
01    P_b ← ∅;             // Initialization
02    for each v ∈ children(v_t1)
      begin
03          G' ← G – Path(s, v, T);
04          P_b ← P_b ∪ Shortest-path(s, v, G');
      end
```

圖 5. NDC_A-I 演算法

```
NDC_A-II (D, G, T)         // 建立樹葉節點間的備援路徑
// D: the set of destination, G: the underlying graph, T: the P2MP tree
01    P_b ← ∅;       // Initialization
02    For each leaf_i ∈ D do
      begin
03          Find leaf_j ∈ D – {leaf_i} such that
04                d(leaf_i, leaf_j) is minimum and Minimum-common-ancestor(leaf_i, leaf_j) = v_t1;
05          G' ← G – Path(leaf_i, leaf_j, T);
06          P_b ← P_b ∪ Shortest-path(leaf_j, leaf_i, G');      // a directed path from leaf_j to leaf_i
      end
```

圖 6. NDC_A-II 演算法

表 2. NDC_A-II 演算法符號說明

D	目的出口(target egresses)的集合。
T	現行的 P2MP 樹 T。
G	MPLS 網路拓樸所成之圖。
P_b	NDC_A-II 演算法欲建立之備援路徑集合。
v_{t1}	T 中的第一層分支節點。
$d(leaf_i, leaf_j)$	點 $leaf_i$ 與節點 $leaf_j$ 的距離。
$Path(leaf_i, leaf_j, T)$	T 上從節點 $leaf_i$ 到節點 $leaf_j$ 的路徑。
G'	圖 G 減去 $Path(leaf_i, leaf_j, T)$所成的圖。
$Minimum\text{-}common\text{-}ancestor(leafi, leaf_j)$	節點 $leaf_i$ 與節點 $leaf_j$ 的最小共同父節點。
$Shortest\text{-}path(leaf_j, leaf_i, G')$	圖 G'中從節點 $leaf_j$ 到節點 $leaf_i$ 的一條最短路徑。

NDC_A-I 演算法採用逐步探索的方式，個別建立從來源節點 s 到每個第一層分支節點之子節點的備援路徑。圖 5 中，以 $path(s, v, T)$ 表示 P2MP 樹 T 上從節點 s 到節點 v 的路徑，而節點 v 爲第一層分支節點的子節點。NDC_A-I 演算法在建立備援路徑時，由於符合 2-connected 環境，因此從節點 s 到節點 v 之間，會有兩條不相交的路徑連通，我們使用 G' 表示 G 減去 $Path(s, v, T)$ 後剩下的子圖，G' 必定是連通的（圖 5 第 3 行），從圖 G' 中個別探索從來源節點 s 到第一層分支節點之子節點 v 之間的最短路徑，並將該最短路徑納入備援路徑 P_b 中（圖 5 第 4 行），直到所有第一層分支節點之子節點 v 都拜訪完畢，備援路徑 P_b 即爲 NDC_A-I 演算法的最終結果。

NDC_A-II 演算法，同樣於 2-connected 環境中，此一階段的目標是將不同子樹（如圖 4 T_C、T_D）間最接近的葉節點做配對，並建立同一對之兩節點間的備援路徑。我們以 D 表示所有目的出口（target egresses）的集合，從 D 中依序選取樹葉節點 $leaf_i$，尋找一個不同子樹且最接近的樹葉節點 $leaf_j$ 配對，並建立 $leaf_j$ 到 $leaf_i$ 的備援路徑（圖 6 第 3-4 行）。完成配對的動作後，其餘建立備援路徑的過程可參考 NDC_A-I 演算法之說明。當所有樹葉節點都完成配對且建立備援路徑後，便完成了 Phase A-II 之步驟，備援路徑 P_b 即爲 NDC_A-II 演算法的最終結果。

（二）路徑切換

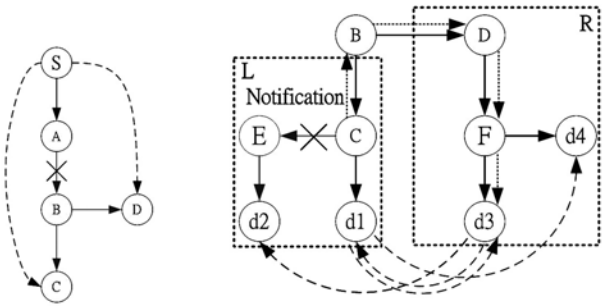
備援路徑建立完成後，即可得到如圖 4 的 NDC 備援樹。當有故障發生時，由發現錯誤的節點將錯誤通報（notification）傳送給上游的路徑切換路由器（path switch LSR, PSL）；PSL 接收通報後，將運作中的路徑切換至備援路徑，以繞過故障的節點或鏈結。在 PSL 的挑選上，我們以圖 4 爲例，分成底下兩個部份來討論。

Case B-I 故障發生在來源節點 S 到第一層分支節點 B 之間（見圖 7(a)），可能是節點 A、B、link SA 或 AB 任何一處發生錯誤。這一類的錯誤稱爲第一型錯誤（Type I fault）。

Case B-II 故障發生在 node B 的下游（見圖 7(b)）。這一類的錯誤稱爲第二型錯誤（Type II fault）。

在 Case B-I 中，PSL 無庸置疑的由節點 S 擔任。以圖 7(a) 爲例，當 link AB 發生錯誤時，節點 A 發現該錯誤後，將發送錯誤通報給身爲 PSL 的節點 S，節點 S 收到錯誤通報後，立即將運作中的路徑切換至備援路徑 $S\rightarrow C$ 及 $S\rightarrow D$。

在 Case B-II 中，PSL 將由樹葉節點擔任。當有故障發生時，對應的錯誤通報都被匯集到節點 B。節點 B 維護一份備援路徑對應表（backup path mapping table, BPMT）。當節點 B 收到來自下游節點的錯誤通報時，將從 BPMT 中查找受影響之目的以及該負責的 PSL。



(a) Type I fault　　(b) Type II fault

圖 7. NDC 路徑切換

表 3 爲節點 *B* 所維護的 BPMT，最左邊的欄位代表無法連通到的節點（source of breaking），之右的每一欄位代表受影響的目的節點以及對應的 PSL（括號中）。舉例來說，假設 link *CE* 發生錯誤（如圖 7(b)），節點 *C* 偵測到錯誤後，會立即將「source of breaking 爲節點 *E*」的訊息附加至錯誤通報中，並向節點 *B* 通報該錯誤。當 *B* 收到該錯誤通報後，可以從 BPMT 中查到受影響的目的節點有 d_2（表 3 第 3 列），可以透過 PSL d_3 來切換備援路徑。

四、系統模擬與效能評估

本節將介紹我們採用的模擬環境、效能評估項目、以及實驗結果。我們將比較 NDC 以及 FRR [5]。

（一）模擬環境

我們採用圖 8 的網路拓樸來模擬 NDC 的效能。該網路包含 15 個節點及 28 條鏈結，並且爲 2-connected。模擬程式是使用 C 語言來撰寫。入口與目的出口的選擇以均勻分佈的亂數來決定。在效能評估上，我們將量測下列項目：（a）現行 P2MP 樹使用的總頻寬 α；（b）採用 NDC 與 FRR 所需的備援頻寬 β_{NDC} 和 β_{FRR}；以及（c）錯誤通報延遲。利用（a）和（b）可以計算備援/工作路徑頻寬比 Π。

表 3. 節點 *B* 之 Backup Path Mapping Table

受影響節點 / source of breaking	d_1 (d_3)	d_2 (d_3)	d_3 (d_1)	d_4 (d_1)
C	√	√		
D			√	√
E		√		
F			√	√
d_1			√	
d_2			√	
d_3	√			
d_4	√			

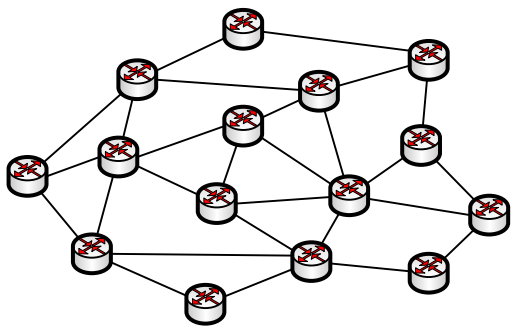


圖 8. 模擬環境之網路架構 [4, 5]

（二）實驗結果分析

實驗結果的部分我們將分成兩種情況說明：第一種是當入口節點同時亦爲第一層分支節點的情況。在這種情況下，省略了 Phase A-I 的備援步驟；第二種是當入口節點與第一層分支節點爲不同節點的情況。

1. Scenario 1：當入口節點同時亦爲第一層分支節點

圖 9 爲目的出口之數目介於 2-5 的情況下，所需要的工作頻寬與備援頻寬。曲線 P2MP 樹代表建立 P2MP 樹所需要的頻寬；曲線 NDC 與曲線 FRR 則分別代表使用 NDC 與 FRR 所需的備援頻寬。圖中每個節點代表 20 次實驗結果的平均值。

在 Scenario 1 中，NDC 省略建立從入口到第一層分支節點下游節點間的備援路徑，只需建立出口間的備援路徑，所以影響 NDC 使用頻寬多寡的主因即爲出口之間的距離。當目的出口數目介於 2-5 時，FRR 約需 12.6-40.95 個單位頻寬；而 NDC 卻只需 5.4-14.8 個單位頻寬，足足比 FRR 省下約 57-64% 備援頻寬。原因是 FRR 需要個別保護每一個節點與鏈結，因此當 P2MP 樹越龐大時，FRR 所需付出的頻寬就越多。

圖 10 爲目的出口數目介於 2-5 的情況下，NDC 與 FRR 所需的通報延遲。我們計算通報延遲的方式爲：從發送錯誤通報的節點到目的 PSL 沿途所經過的節點數。因 FRR 爲區域修復，所以通報延遲固定爲 1 hop delay（與出口的數目無關）。一般來說，NDC 省下頻寬的同時，也必須付出較多的通報延遲。當目的出口個數介於 2-5 時，NDC 通報延遲介於 2.9-5.4 跳躍傳輸延遲（hop delay），比 FRR 多出 1.9-4.4 跳躍數（hop）。

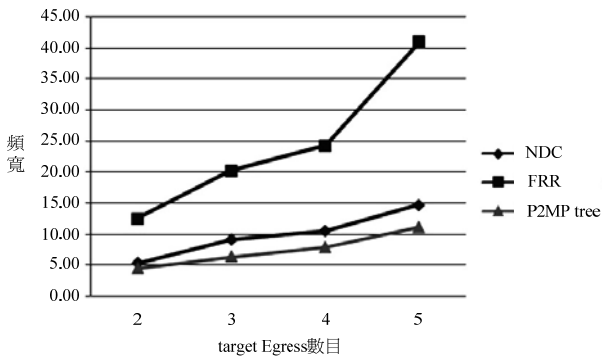


圖 9. 目的出口數與頻寬（工作頻寬及備援頻寬）之比較

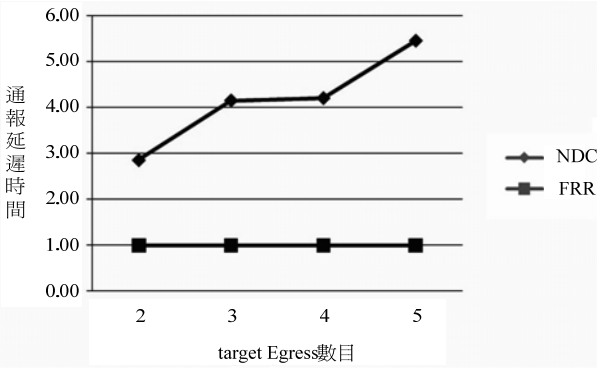


圖 10. 目的出口數與通報延遲時間之比較，以跳躍數（hop count）計

2. Scenario 2：當入口節點與第一層分支節點爲不同節點

圖 11 爲目的出口數目介於 2-5 時，所需要的工作頻寬以及備援頻寬。Scenario 2 必須遵循 NDC 的 Phases A-1/A-2 來建立備援路徑，因此在節省頻寬的效益上，不如 Scenario 1 來的顯著。但整體而言，NDC 仍舊優於 FRR：NDC 使用了 9.1-25.1 單位頻寬；相較於 FRR 使用的 13-36.6 單位頻寬，省下約 30-31.4% 的備援頻寬。

與 Scenario 1 相比（見圖 9），在 Scenario 2 中 NDC 使用的備援頻寬提高近 2 倍左右（從 5.5-14.8 單位頻寬增加爲 9.7-25.7 單位頻寬）。我們以圖 12 爲例，將原因說明如下：圖 12 是目的出口數目等於 2 的樣本（sample）。因爲入口 *G* 到第一層分支節點 *M* 之間的路徑較偏向網路邊緣，節點 *G*→*N* 的備援路徑需要繞一大圈遠路，造成備援頻寬的提升。

圖 13 爲目的出口數目介於 2-5 時，NDC 與 FRR 所需的通報延遲。因爲入口與第一層分支節點爲相異節點，使得原先距離最遠的出口節點之間的通報延遲得以分爲 Case B-I 與 Case B-II 兩部份來計算，讓平均錯誤通報的延遲從原

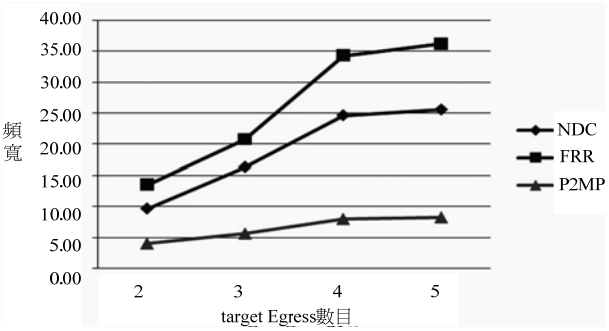


圖 11. 目的出口數與頻寬（工作頻寬及備援頻寬）之比較

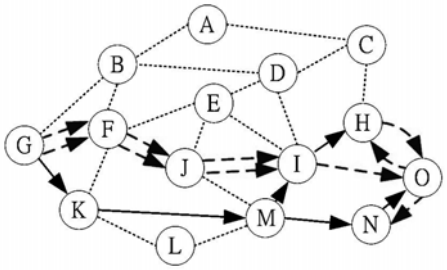


圖 12. Scenario 2 範例

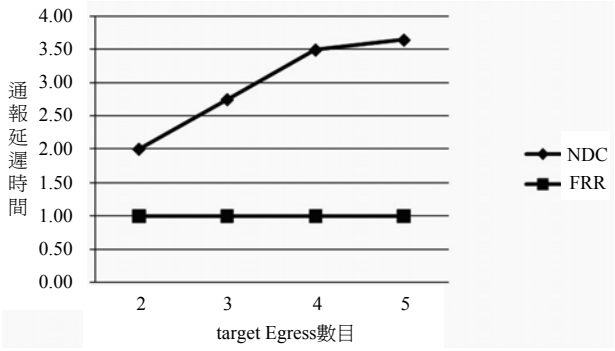


圖 13. 目的出口數與通報延遲時間之比較，以跳躍數計

本的 2.9-5.4 跳躍數（hop）下降至 2-3.6 跳躍數。

五、結論

本文提出一個 MPLS P2MP 的快速回復機制 NDC。NDC 以第一層分支節點爲中心，將 P2MP 樹切割成兩部分，然後分別建立所需的備援路徑。NDC 的主要貢獻在於能以些許的錯誤通報延遲，換取相當可觀的備援頻寬。

在一般情況下，降低備援頻寬與降低通報延遲這兩種需求無法兼顧。因此，改善 NDC 機制使其能在給定的通報延遲上限之內找到合適的備援方式，是我們未來的研究議題。

參考文獻

1. Aggarwal, R., D. Papadimitriou and S. Yasukawa (2007) *Extensions to Resource Reservation Protocol- Traffic Engineering* (*RSVP-TE*) *for Point-to-Multipoint TE Label Switched Paths* (*LSPs*). RFC 4875, Internet Engineering Task Force, Fremont, CA.
2. Chartrand, G. and O. R. Oellermann (1993) *Applied and Algorithmic Graph Theory*, McGraw-Hill, New York, NY.
3. Kodialam, M. and T. V. Lakshman (2002) Dynamic routing of bandwidth guaranteed multicasts with failure

backup. The 10th IEEE International Conference on Network Protocols, Paris, France.

4. Kodialam, M., T. V. Lakshman and S. Senqupta (2004) A simple traffic independent scheme for enabling restoration oblivious routing of resilient connections. The 23rd IEEE International Conference on Computer Communications, Hong Kong, China.
5. Li, G., D. Wang and R. Doverspike (2006) Efficient distributed MPLS P2MP fast reroute. The 25th IEEE International Conference on Computer Communications, Barcelona, Spain.
6. Rosen, E., A. Viswanathan and R. Callon (2001) *Multiprotocol Label Switching Architecture*. RFC 3031, Internet Engineering Task Force, Fremont, CA.

收件：98.01.09　修正：98.03.02　接受：98.04.30

附錄 A　建構 P2MP 樹

最近鄰居優先（nearest neighbor first, NNF）是建立 P2MP 樹常用的方法 [3, 5]。NNF 演算法如圖 A-1，表 A-1 爲演算法的符號說明。

NNF 演算法之目的在於找到一棵從入口 *s* 到所有目的出口 *D* 之間的最小樹（minimum tree），亦即從入口到出口所經過最少跳躍數（hop count）的樹。透過 NNF 演算法計算出來的最小樹，即是我們所要建立的 P2MP 樹。

NNF 演算法採取一次又一次探索的方式，透過每回合計算所有節點與 P2MP 樹 *T* 之間的距離，挑選出與 tree *T* 距離最短的節點及該鏈結（表 A-1 第 4 行），並將該節點與鏈結加入 P2MP 樹 *T* 中（圖 A-1 第 5 行）。NNF 演算法將不斷地重複上述步驟，直到 *X* 集合中所有目的地節點都已經被納入 P2MP 樹 *T* 中（圖 A-1 第 3 行）。

```
NEAREST_NEIGHBOR_FIRST (s, D, G=(V, E))
// s: the source, D: the set of destination, G: the underlying graph
01    X ← D;          // Initialization
02    T ← {s};
03    while (X≠∅)
      begin
04          Find a ∈ X, such that d(a, T) = min_{x ∈ X} d(x, T);
05          T ← T ∪ Shortest-path(a, T, G);
06          X ← X̃{a};
      end
07    Return T;
```

圖 A-1. NNF 演算法

表 A-1. NNF 演算法符號說明

s	入口 *s*，爲 P2MP 樹的起點。
D	目的出口的集合。
G	MPLS 網路拓樸所成之圖。
X	待探訪之目的出口的集合。
T	NNF 演算法欲建立之 P2MP 樹。
d(*a*, *T*)	節點 *a* 到樹 *T* 之距離。
Shortest-path(*a*, *T*, *G*)	圖 *G* 中從節點 *a* 到樹 *T* 的一條最短路徑。