# 第三章 動態負載平衡演算法和其缺失修正

因爲本論文主要參考由 Keping Long,Zhongshan Zhang,Shiduan Cheng 三人所提出的動態負載平衡演算法 (DLB(Dynamic Load Balance Algorithm))，我們發現此演算法有些缺失，因而加以修正，然後提出一種演算法稱爲具有優先權的動態負載平衡演算法 (DLBP(Dynamic Load-Balance Algorithm with Priority))，此演算法乃是將修正後之 DLB 演算法加以延伸，加入優先權的觀念。因此要說明我們所提出的 DLBP 演算法之前，必需先對 DLB 演算法加以描述，等對 DLB 演算法有所了解之後，我們才能對 DLBP 演算法加以講解。本章將對 DLB 演算法和它的缺失修正詳加介紹。

## 3.1 介紹

在流量工程裡，負載平衡扮演著很重要的角色。負載平衡的任務是使用一些機制，精準的控制資料流經過網路的路徑，以避免流量的擁塞，並且提昇整體的網路輸出量 (network throughput)和網路資源利用率 (network resource utilization)以及維持該有的品質保證(Quality of Service)。

在 Keping Long 等三人共同提出的這篇論文中，提出了三種負載平衡演算法：TSLB 演算法(Topology-Based Static Load-Balance Algorithm)、RSLB 演算法 (Resource-Based Static Load-Balance)、DLB 演算法(Dynamic Load-Balance Algorithm)。此三種演算法只適用於無優先權的情況，而根據模擬的結果，証明了在這三種演算法中，以 DLB 演算法的效能最佳。我們提出的 DLBP 演算法便是將修正後之 DLB 演算法加以延伸，加入優先權的觀念，使其能適用於具有優先權資料流的情況。其中 DLB 演算法乃是應用了 TSLB 演算法，因此在我們介紹 DLB 演算法之前，我們必需先說明 TSLB 演算法。

## 3.2 TSLB 演算法

### 3.2.1 TSLB 演算法的描述

在介紹 TSLB 演算法之前，我們必需先介紹一些相關的專有名詞：

UB (Used Bandwidth)： 被一 CR-LSP 所消耗掉的資源。

FC (Free Capacity)： 我們假設CR-LSP R 經由鏈接(links) $L_{i,j}$ ，其中i，j 分別是$L_{i,j}$的上游節點(upstream node)和下游節點(downstream node)。如果$L_{i,j}$未被使用的容量是$F_{i,j}$，那麼FC=MIN($F_{i,j}$)。

CB (Contributed Bandwidth)： 一 CR-LSP 放棄它的資源之後，此路徑的 FC 值，即 CB = UB + FC。

SRC (Selectable-Route-Collection)： 是由 Ingress LSR 及 Egress LSR 之間的所有平行路徑所組成。

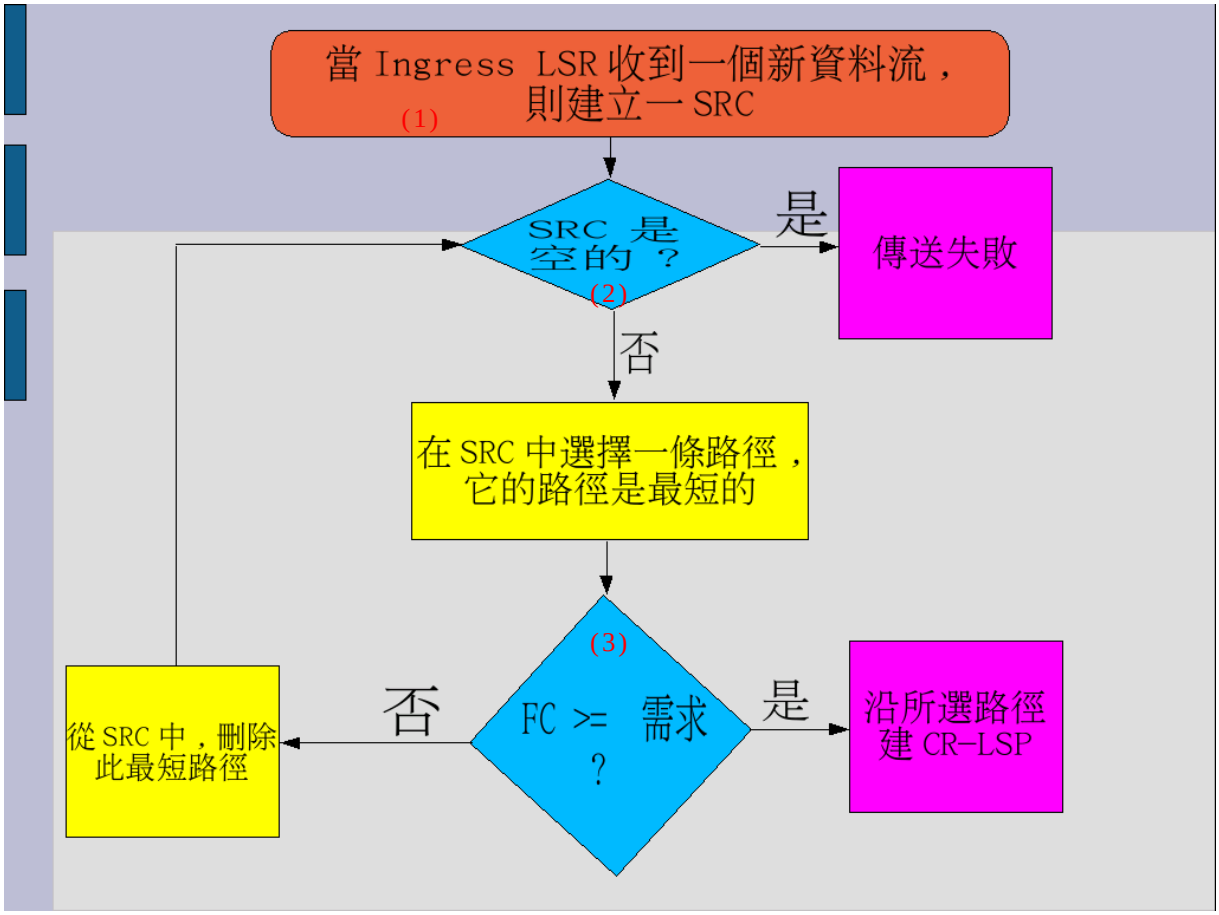


圖 11 TSLB 演算法之流程圖

圖 11 爲 TSLB 演算法之流程圖，我們對照流程圖及其編號描述如下：

（1） 當 Ingress LSR 收到一個新資料流(traffic flow)，先建立 SRC。

（2） 如果 SRC 是空的，則表示網路容量無法滿足新資料流的頻寬需求，傳送失敗，演算法結束；否則在 SRC 中選擇一條路徑，它的長度是最短的

（3） 此最短的路徑，如果它的 FC 值是大於新資料流的頻寬需求，則沿著所選的路徑，建立一條新的 CR-LSP，並將此新資料流導向此 CR-LSP，然後將此路徑的 FC 值更新，此演算法成功地完成。；否則，從 SRC 中刪除此最短路徑，重新尋找其它路徑。

在 MNS 模擬器已有與 TSLB 演算法功能一樣的模組，但其演算法，與 TSLB 演算法有些不同，有點類似 Dijkstra 的最短路徑演算法，而其所求到的路徑 FC 值滿足新資料流的頻寬需求。因爲本篇論文是以 MNS 模擬器爲基礎的，因此在本篇所有的模擬，只要需要用到 TSLB 演算法時，皆使用在 MNS 模擬器已經具有，而功能和 TSLB 演算法相同的模組，而不另外撰寫 TSLB 模組。

3.2.2 TSLB 演算法之模擬和分析

我們利用MNS模擬器本身具有和TSLB功能相同的模組，發展一個OTcl劇本（OTcl scipt），來模擬如圖12之情境。圖12 爲用來模擬TSLB演算法之網路拓撲、各資料流的傳送順序及各資料流的需求頻寬。此網路拓撲中N3爲MPLS之網域入口節點(Ingress Node)，N8爲MPLS之網域出口節點(Egress Node)，而N0到N2爲產生資料流電腦設備，而N9到N11爲接收資料流的電腦設備，而N3到N8爲MPLS網域的核心路由器(Core Router)，核心路由器之間的每一個網路鏈接（link）頻寬如圖所示。 其中有三個資料流：從N0流向N9，從N1流向N10，從N2流向N11。

我們將TSLB演算法的模擬環境參數設定如下：

資料流傳送資料的型態與方式：Poisson process/UDP。

資料流的傳送順序：依序爲N0，t=0sec,其所建之CR-LSP之LSPid爲1100

N1，t=0.5sec,其所建之CR-LSP之LSPid爲1200

N2，t=1sec,其所建之CR-LSP之LSPid爲1300

模擬過程中所產生的資料流總數：3個。
每個資料流產生的間隔時間：0.5秒。
每個資料流傳送資料的時間：3秒。
總共觀察的時間：3秒。
每個資料流所需求頻寬大小：如圖12所示。

上面所提到，資料流傳送資料的型態爲Poisson process，表示資料流封包送出的過程爲卜松過程（Poisson process）。也就是說每個時間點送出封包的機率相等，即均勻分佈（uniform distribution），且無叢發（burst）產生。因爲是卜松過程，所以二個封包被送出的時間間距是指數分佈（exponential distribution）。我們設定資料流的平均送出率（mean sending rate），爲其頻寬需求的90%。之所以設定爲資料流頻寬需求的90%，是爲了避免當封包太過集中送出時，超過CR-LSP保留的頻寬，而封包被丟棄。我們對網路的設計爲：當Ingress LSR收到的資料流資料傳送速率（sending rate）超過爲此資料流而建的CR-LSP保留的頻寬時，會先將多出來的封包暫存在Ingress LSR的緩衝區（buffer），若超過緩衝區之封包，則給予丟棄。

由圖 13 ，我們可以看出每一個資料流都順利的被導向適當的路徑。其中各資料流要求 Ingress LSR 建立 CR-LSP 的時間和 CR-LSP 被建立的時間差，即是建立 CR-LSP 所需要的時間。例如：N1 送出的資料流要求 Ingress LSR 建立 CR-LSP 是在時間爲 0.5 秒的時候，而其 CR-LSP（LSPid=1200）被建立的時候，時間爲 0.54 秒，故建立此 CR-LSP 所花費的時間爲 0.04 秒。

圖 14 是將圖 12 之資料流的傳送順序改成依序爲 N2.N1.N0，其餘不變。由圖 15 得知 N0 送出的資料流，因沒有找到足夠的可用頻寬，因此傳送失敗。所以我們可以看出在一些特別的情況，TSLB 演算法會使得網路資源不合理的分配，導致低的輸出量和低的網路資源利用率。

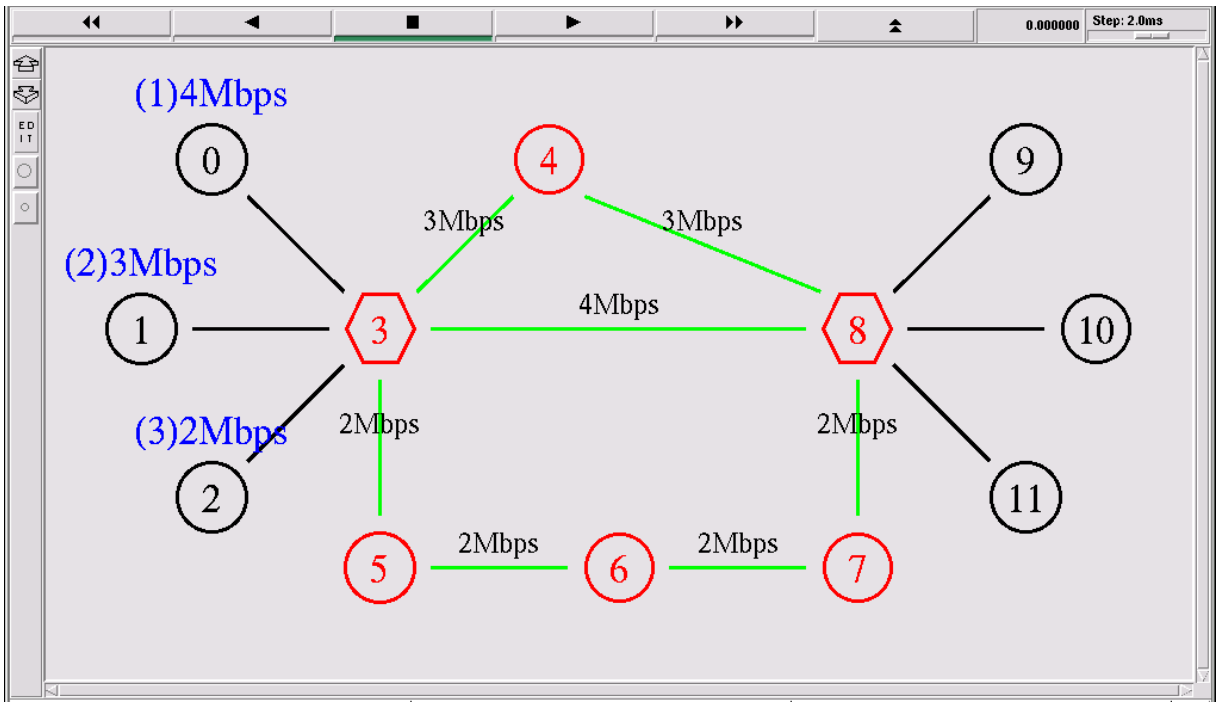


圖 12 用來模擬 TSLB 演算法之網路拓撲、各資料流的傳送順序、各鏈接的頻寬及各資料流的需求頻寬。

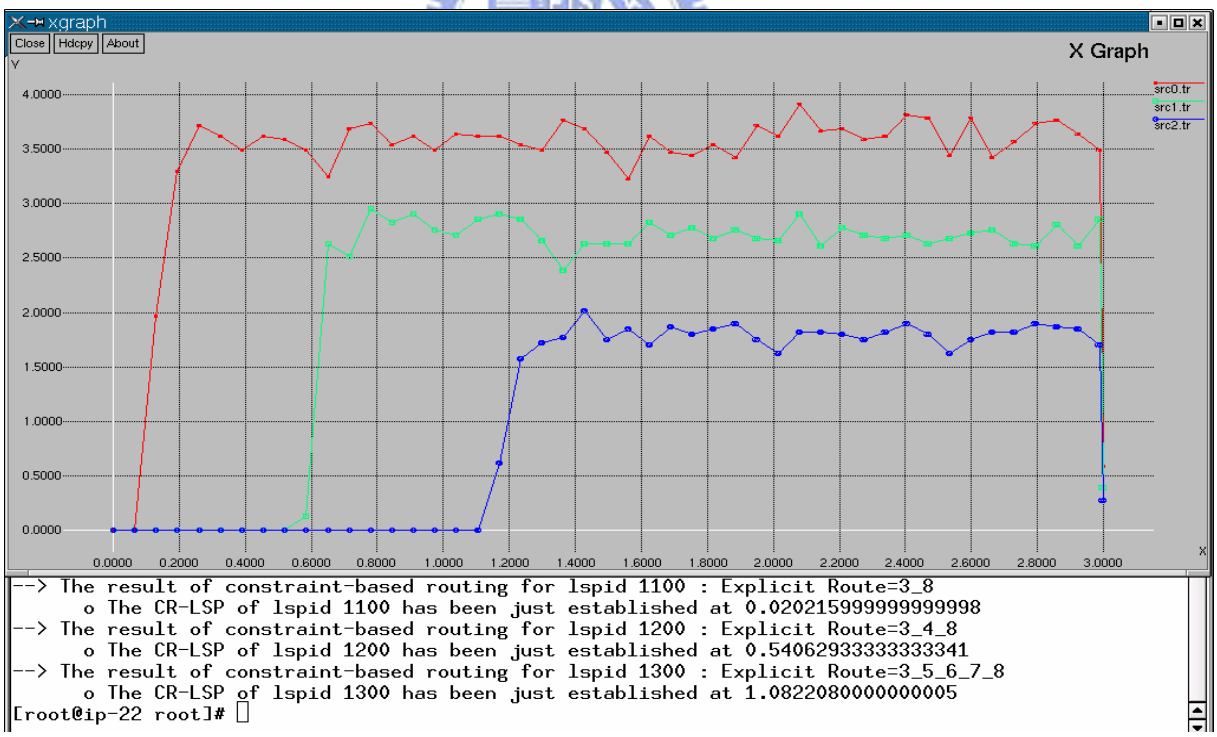


圖 13 使用 TSLB 演算法模擬圖 12 得到之每一個資料流的輸出量（throughput），以及每一條 CR-LSP 之路徑和建立時間

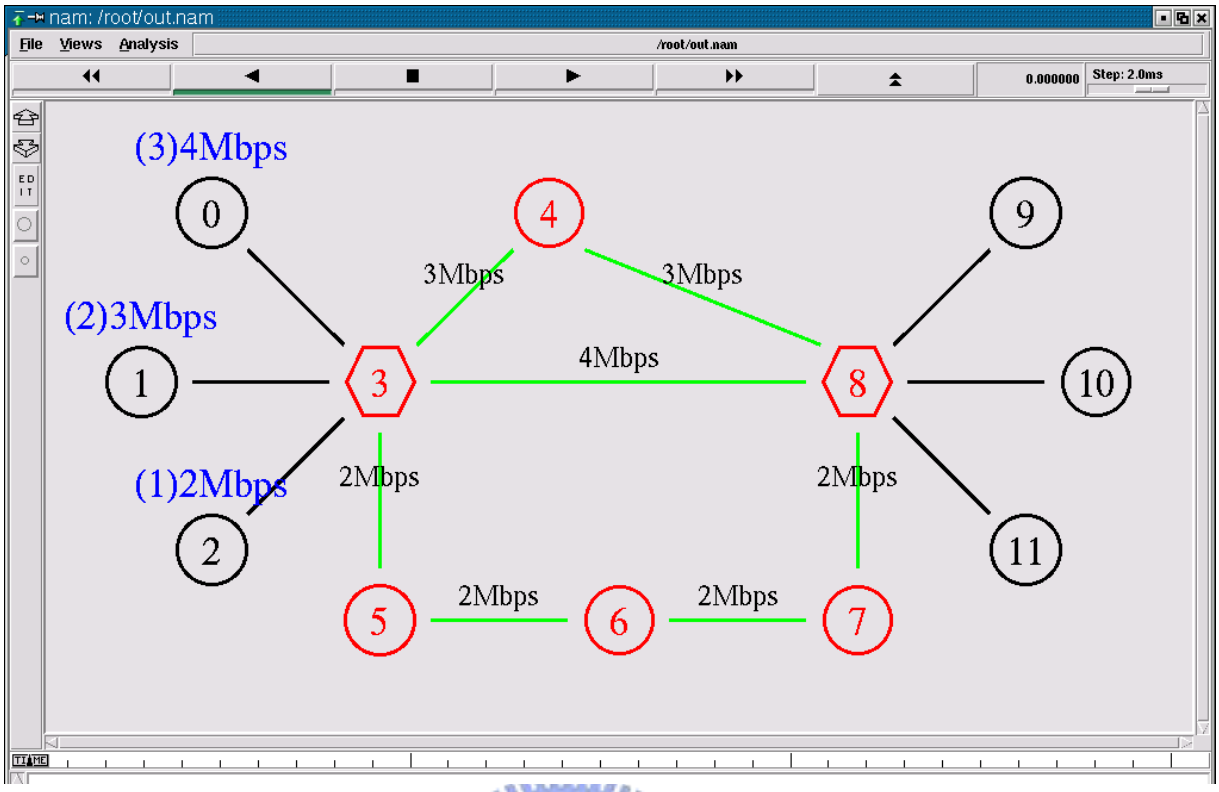


圖 14 用來模擬 TSLB 演算法之網路拓撲、各資料流的傳送順序、各鏈接的頻寬及各資料流的需求頻寬。

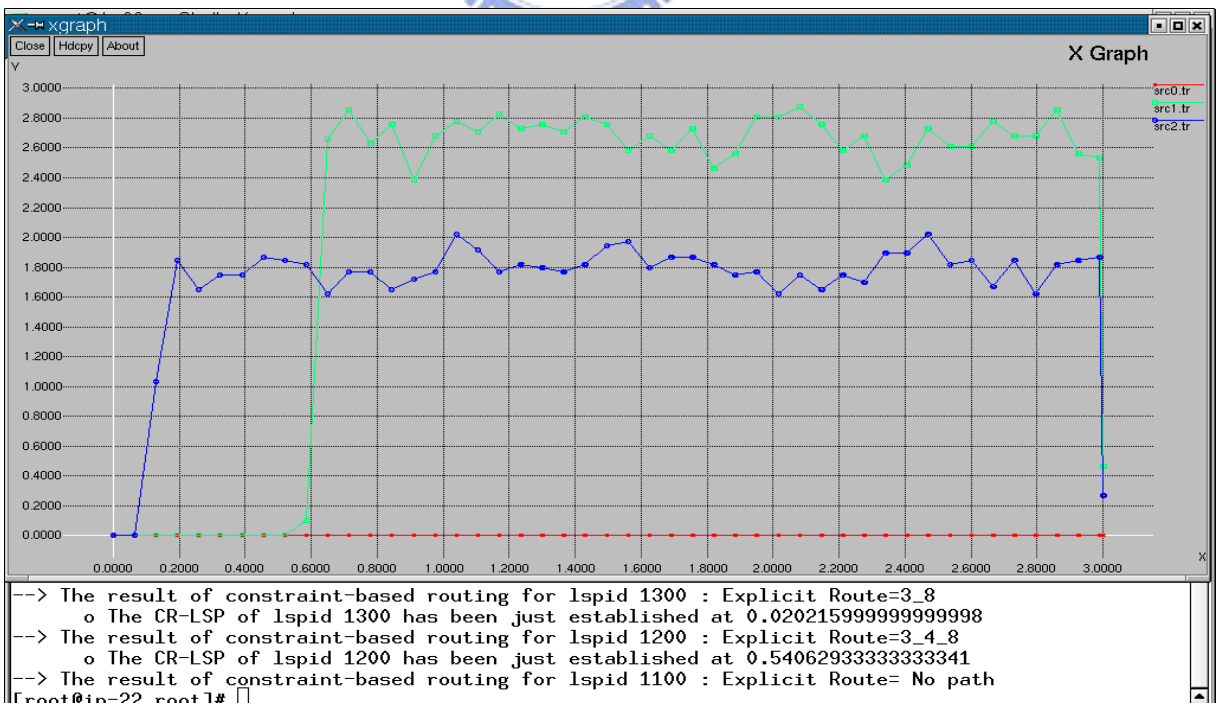


圖 15 使用 TSLB 演算法模擬圖 14 得到之每一個資料流的輸出量（throughput），以及每一條 CR-LSP 之路徑和建立時間

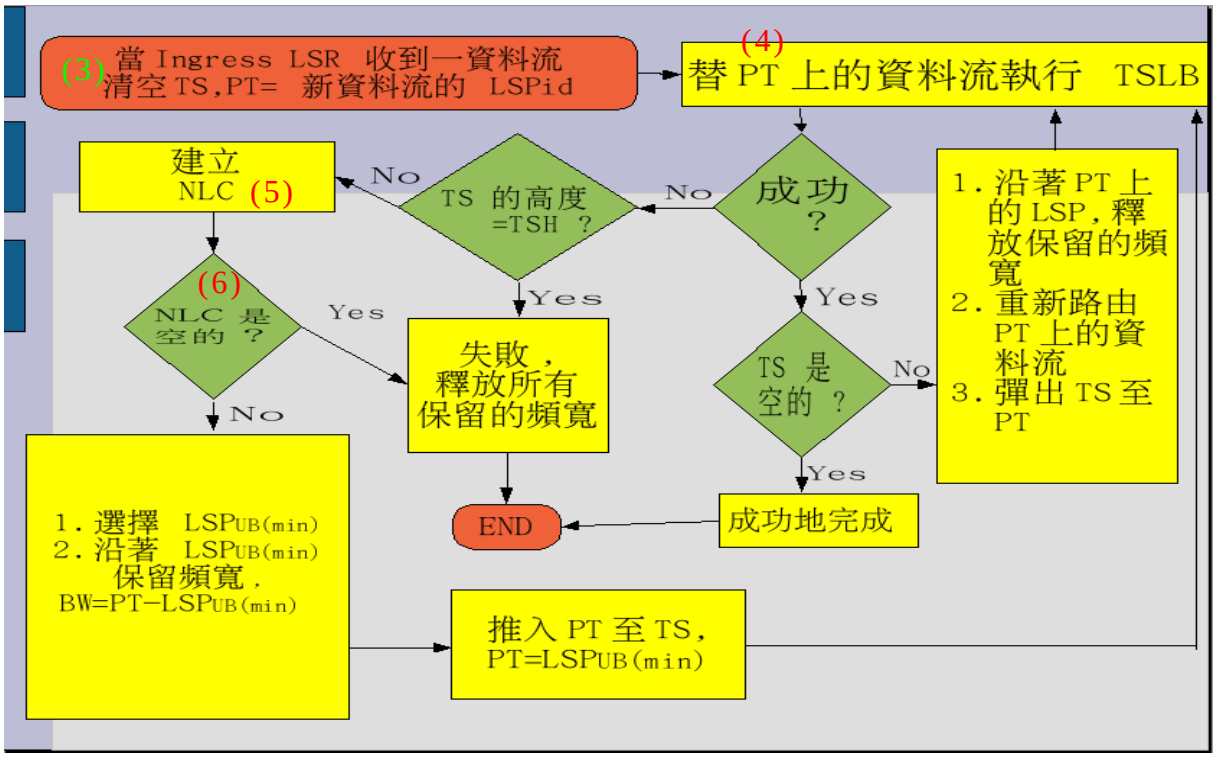


圖 16 經修正之 DLB 演算法之流程圖

## 3.3 經修正之 DLB 演算法

### 3.3.1 經修正之 DLB 演算法的描述

因 TSLB 演算法會使得網路資源不合理的分配，導致低的輸出量和低的網路資源利用率。DLB 演算法乃是針對 TSLB 演算法的缺失，加以改良。圖 16 爲經修正之 DLB 演算法之流程圖，我們對照流程圖及其編號描述如下：

（1） 使用一個變數 pending-Traffic 來存放即將選擇路徑或重新路由的 LSPid，並且建一個堆疊(stack)稱爲 Traffic-Stack(TS)。此堆疊用來存放等待選擇路徑或等待被重新路由的 LSPid。

（2） 我們設定允許再路由的最高次數 TSH。

✧ 說明：設定 TSH 的目的爲避免因重新路由次數太高，而影響傳輸品質，將於 3.3.3 節中詳加敘述。

（3） 當 Ingress LSR 收到一個新的資料流，我們清空 TS，然後將此資料流的 LSPid 放入 pending-Traffic。

（4） 使用 TSLB 演算法替 pending-Traffic 上的那一個資料流找尋一個路徑：

- 如果成功，則我們進一步判斷 TS 是否爲空的：
  - — 如果 TS 是空的：表示 pending-Traffic 上的那一個資料流爲新資料流，因此我們沿著所選的路徑建立一條 CR-LSP，然後將新資料流導向此新建的 CR-LSP,此演算法成功地完成。
  - — 如果 TS 不是空的：表示 pending-Traffic 上的那一個資料流不是新資料流。我們首先釋出在 pending-Traffic 的 CR-LSP 路徑上之額外保留之頻寬，然後重新路由 pending-Traffic 上的那一個資料流至剛找到的路徑，之後彈出（pop）TS 頂端的值給 pending-Traffic，最後再使用 TSLB 演算法替目前 pending-Traffic 上的那一個資料流找尋路徑。
- 如果失敗，則我們進一步判斷 TS 之高度是否等於 TSH：
  - — 如果 TS 之高度等於 TSH，則此 MPLS 網路容量無法滿足新資料流的頻寬需求，傳送失敗，釋出之前額外保留之所有頻寬，演算法結束。
  - — 如果 TS 之高度小於 TSH，那麼我們建立一個 NLC 集合。
    - ✧ 說明：建立一個 NLC 集合的目的爲爲了找尋一個已存在之 CR-LSP，此 CR-LSP 可重新路由至其它路徑，而騰出較高容量的路徑給 pending-Traffic 上的資料流使用。

（5） 建立 NLC：在 MPLS 網路裡，收集所有已存在之 CR-LSP，這些 CR-LSP 必需符合：

＞ 它的 Ingress 及 Egress LSR 和 pending-Traffic 上的資料流相同。

＞ 它的 UB 值必須小於 pending-Traffic 的資料流的頻寬需求。

＞ 它的 CB 值必須大於或等於 pending-Traffic 的資料的頻寬需求。

之後，我們建立一個集合稱爲 NLC(Negotiate CR-LSP Collection)，此 NLC 是由這些收集到的 CR-LSP 所組成。

（6） 判斷 NLC 是否爲空集合：

- 如果 NLC 是空的，此 MPLS 網路容量無法滿足新資料流的頻寬需求，傳送失敗，釋出之前額外保留之所有頻寬，演算法結束。
- 如果 NLC 不是空的，首先選一 CR-LSP，它的 UB 值是在 NLC

裡面最小的，然後沿著此一選到之 CR-LSP 路徑，額外保留一些頻寬，此頻寬大小爲（目前 pending-Traffic 上的資料流需求頻寬減去此一 CR-LSP 的資料流保留頻寬）。之後，我們推入(push) pending-Traffic 值至 TS 頂端，將剛剛所選到的 CR-LSP 的 LSPid 分配給 pending-Traffic。

✧ 說明：1. 爲何要額外保留一些頻寬，將於 3.3.3 節中詳加敘述。

2. 之所以要將 pending-Traffic 的值推入至 TS 頂端，因爲 TS 是用來存放等待選擇路徑或等待被重新路由的 LSPid。而之所以要將剛剛所選到的 CR-LSP 的 LSPid 分配給 pending-Traffic，因爲 pending-Traffic 是用來存放即將選擇路徑或重新路由的 LSPid。而剛剛所選到的 CR-LSP 必須重新路由到其它路徑，以便騰出較高容量的路徑。

### 3.3.2 經修正之 DLB 演算法之模擬和分析

對於 DLB 演算法，在輕載的情況，資料流可以被導向較短及容量較高的路徑；當負載變重時，頻寬需求較小的資料流會被重新路由(reroute)至其它適合的路徑，使得能保留高容量的路徑給予頻寬需求大的資料流。因此 DLB 演算法可以明顯的增加網路輸出量和資源利用率。

我們根據修正後之 DLB 演算法流程圖，撰寫了一個 DLB 演算法的模組，加入在 MNS 模擬器上，使此 MNS 模擬器能夠用來模擬 DLB 演算法。然後發展一個 OTcl 劇本（OTcl scipt），來模擬如圖 17 之情境。圖 17 爲用來模擬 DLB 演算法之網路拓撲、各資料流的傳送順序、各鏈接的頻寬及各資料流的需求頻寬。

我們將DLB演算法的模擬環境參數設定如下：

資料流傳送資料的型態與方式：Poisson process/UDP。

資料流的傳送順序：依序爲N1，t=0秒,其所建之CR-LSP之LSPid爲1200

N2，t=0.5秒,其所建之CR-LSP之LSPid爲1300

N2，t=1秒,其所建之CR-LSP之LSPid爲1400

N0，t=1.5秒,其所建之CR-LSP之LSPid爲1100

模擬過程中所產生的資料流總數：4個。

每個資料流產生的間隔時間：0.5秒。

每個資料流傳送資料的時間：3秒。

總共觀察的時間：3秒。

每個資料流所需求頻寬大小：如圖17所示。

在圖19中，LSPid=1201,1301,1401分別是爲了重新路由LSPid=1200,1300,1400上的資料流，所新建立的CR-LSP。等資料流重新導向新建的CR-LSP後，便可將原CR-LSP釋放。

我們根據圖18和圖19做分析，前面三個資料流（頻寬需求依順序分別爲3Mbps,2Mbps,1.2Mbps）都根據TSLB演算法順利的建立CR-LSP（LSPid分別爲1200,1300,1400），並且順利地將資料流傳送。但Ingress LSR在第四個資料流（需求頻寬爲4Mbps）要求建立CR-LSP時，執行TSLB無法替此資料流找到路徑，因此根據DLB演算法，開始尋找是否可將其它已存在之CR-LSP重新路由至其它路徑，而騰出之路徑具有足夠的可用頻寬（即CB值），給予新資料流建立CR-LSP。從圖19可知，若重新路由LSPid=1200之CR-LSP，則重新路由後，其原路由即有足夠的頻寬給新資料流使用，但爲能成功地重新路由LSPid=1200之CR-LSP，則需要先重新路由LSPid=1300之CR-LSP，爲要能成功地重新路由LSPid=1300之CR-LSP，則需要先重新路由LSPid=1400之CR-LSP，LSPid=1400之CR-LSP很順利地找到路徑，重新路由成功(重新路由後之CR-LSP之LSPid=1401)。之後依序LSPid=1300,LSPid=1200亦順利重新路由成功(重新路由後之CR-LSP之LSPid分別爲1301,1201)，最後新資料流便找到路徑，將CR-LSP（LSPid=1100）建立起來。

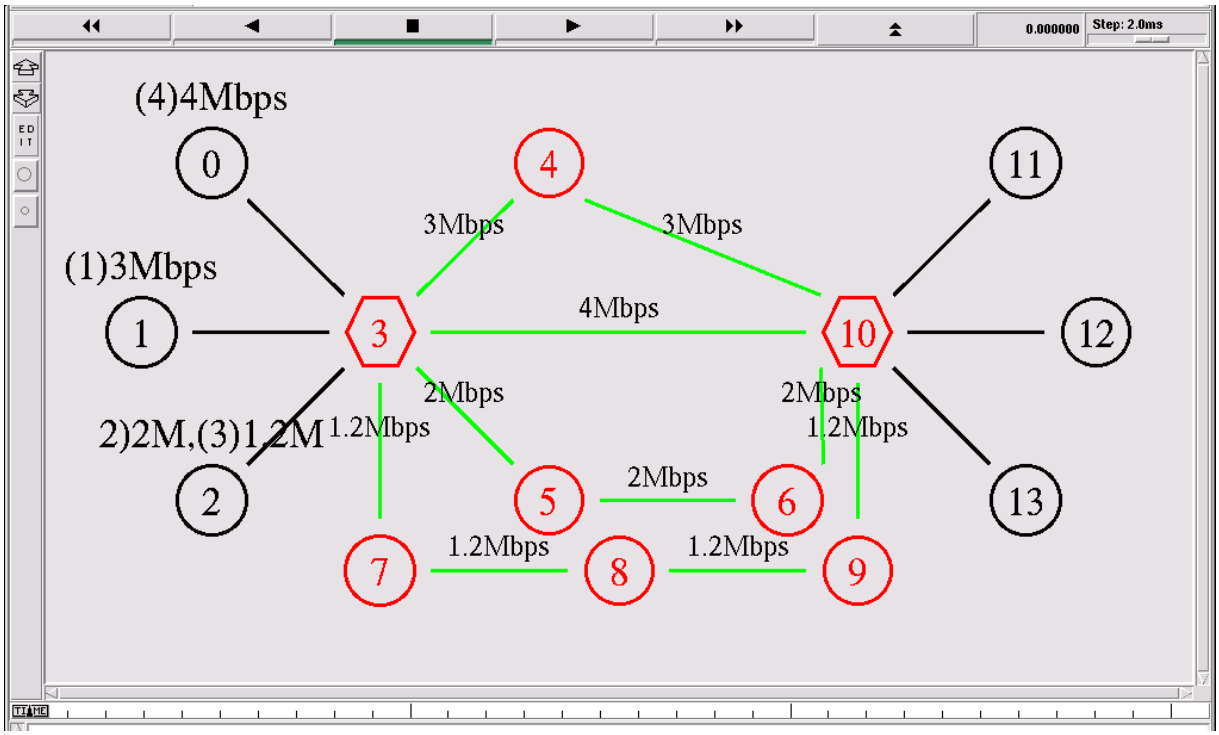


圖 17 用來模擬 DLB 演算法之網路拓撲、各資料流的傳送順序、各鏈接的頻寬及各資料流的需求頻寬

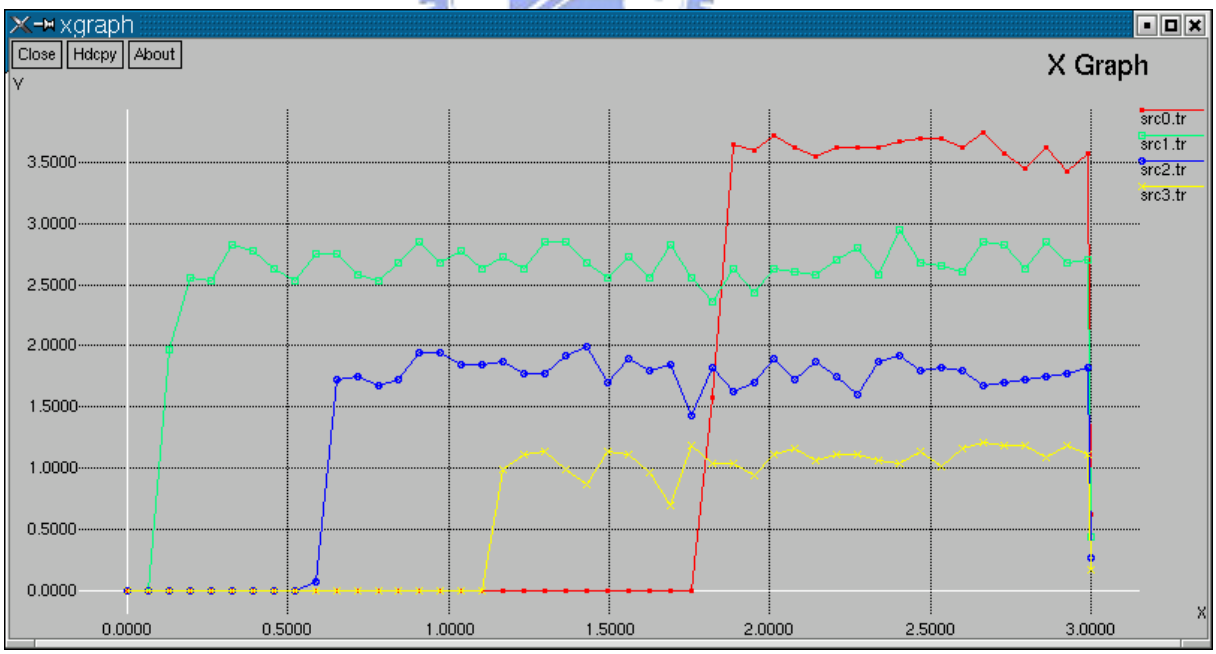


圖 18 使用 DLB 演算法模擬圖 17 得到之每一個資料流的輸出量（throughput）

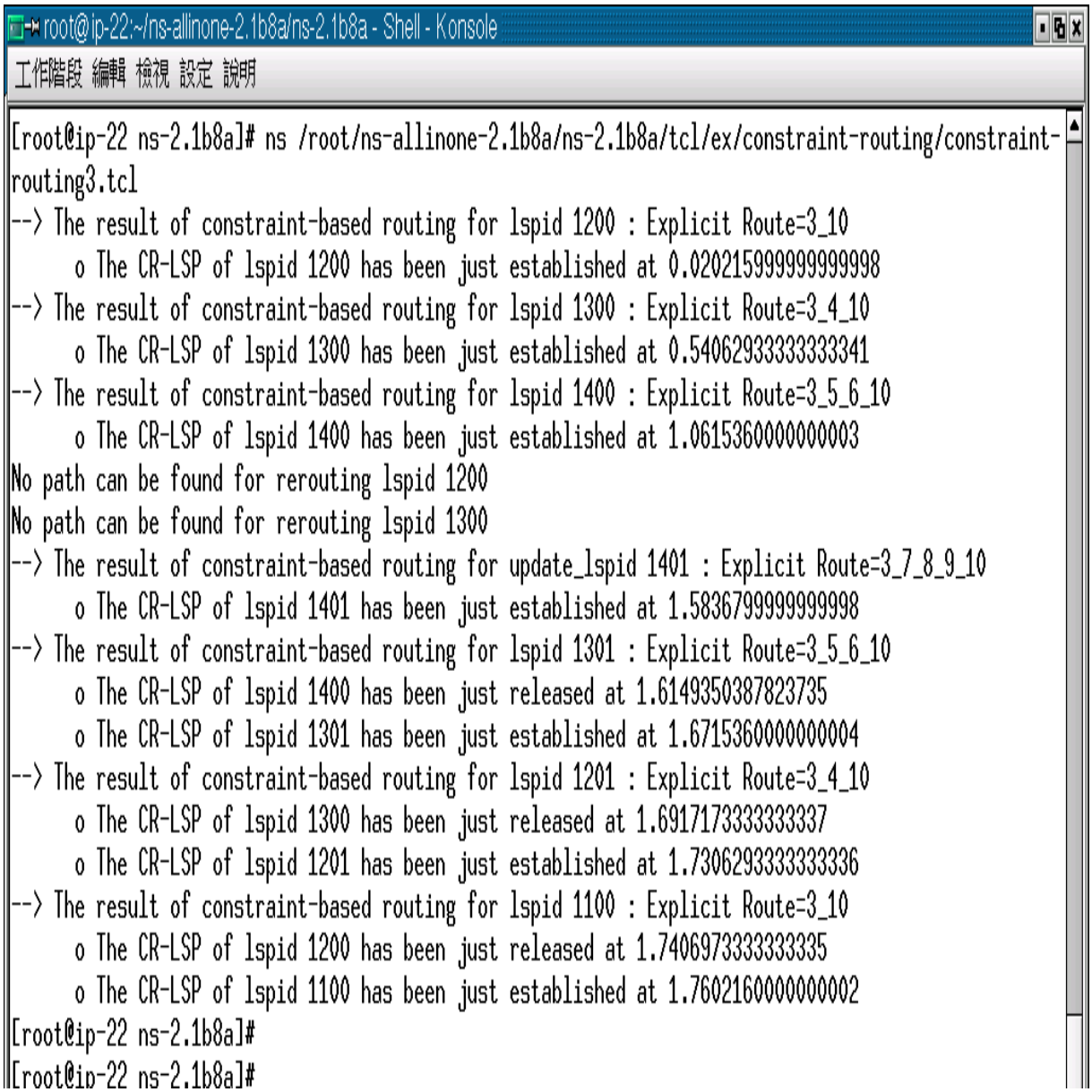


圖 19 使用 DLB 演算法模擬圖 17 得到之每一條 CR-LSP 之路徑、建立時間和被釋放時間

### 3.3.3 原作者之 DLB 演算法和我們修正後之 DLB 演算法之比較

我們對原作者之 DLB 演算法做了部份的修改，修改部份如下：

1. 我們增加了設定允許再路由的最高次數 TSH 參數：此 TSH 參數是原 DLB 演算法所沒有的，原 DLB 演算法雖然當負載變重時，頻寬需求較小的資料流會被重新路由至其它適合的路徑，使得能保留高容量的路徑給予頻寬需求大的資料流。因此 DLB 演算法可以明顯的增加網路輸出量和資源利用率。但我們發現了一個問題：當頻寬需求較大的新資料流利用 TSLB

演算法找不到足夠頻寬容量可建 CR-LSP 時，可能許多頻寬需求較小的 CR-LSP 必須連鎖地做重新路由的動作，才能騰出高容量的路徑給予此新資料流使用。此過程可能需要花費非常長的時間。此複雜度的分析，我們將在第六章複雜度的分析中會有詳盡的說明。因此將會影響整個網路的傳輸品質，爲了避免此一情況的發生，我們在原 DLB 演算法中，增加了設定允許重新路由的最高次數 TSH 參數。當演算法計算出，重新路由的次數必需超過允許重新路由的最高次數 TSH 值才能替新資料流找到路徑時，我們就放棄替新資料流建立 CR-LSP，以避免如上面所述，影響整個網路的傳輸品質之情形發生。

2. 原 DLB 演算法的另外一個缺失：我們將原 DLB 演算法有錯誤的部分摘錄如下：

   - 如果 NLC 不是空的，首先選一 CR-LSP，它的 UB 值是在 NLC 裡面最小的，然後，我們推入 pending-Traffic 值至 TS 頂端，將剛剛所選到的 CR-LSP 的 LSPid 分配給 pending-Traffic。

   而我們將其更正爲：

   - 如果 NLC 不是空的，首先選一 CR-LSP，它的 UB 值是在 NLC 裡面最小的，然後沿著此一選到之 CR-LSP 路徑，額外保留一些頻寬，此頻寬大小爲（目前 pending-Traffic 上的資料流需求頻寬減去此一 CR-LSP 的資料流保留頻寬）。之後，我們推入(push) pending-Traffic 值至 TS 頂端，將剛剛所選到的 CR-LSP 的 LSPid 分配給 pending-Traffic。

✧ 說明：二者的差別爲我們加入沿著此一選到之 CR-LSP 路徑，額外保留一些頻寬的觀念，我們將詳加說明於後。

我們爲了說明原 DLB 演算法的缺失，而又爲什麼我們將原 DLB 演算法如此修正，我們利用下圖（圖 20）的網路拓撲及情境來說明。

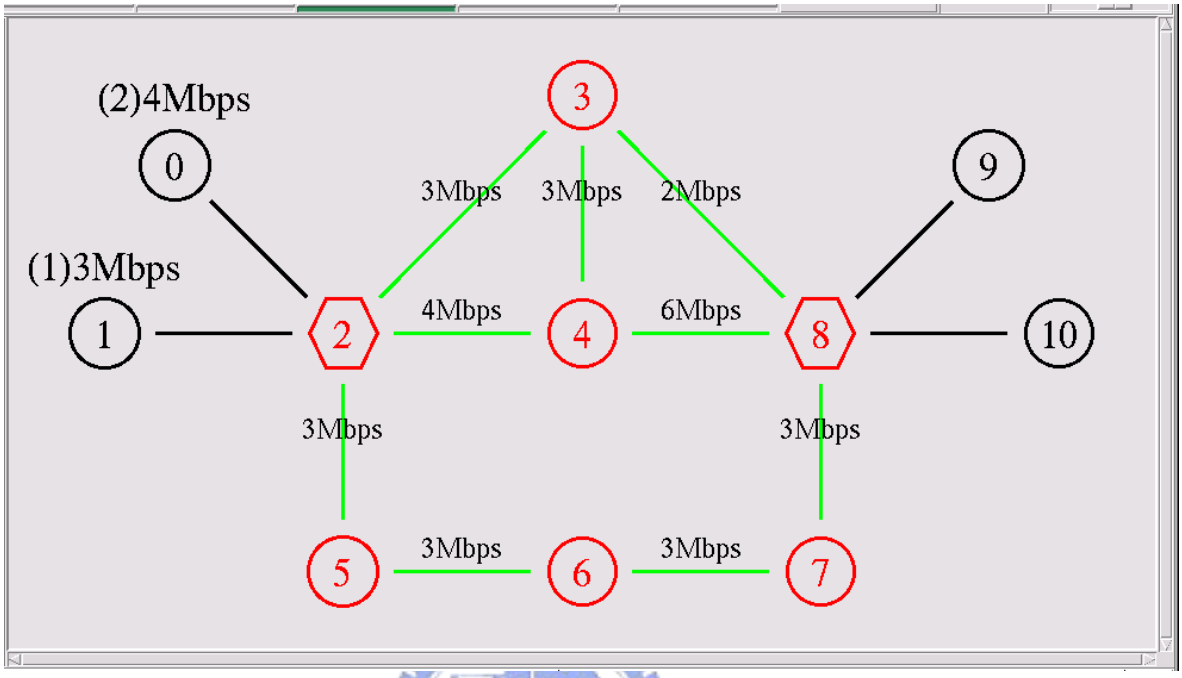


圖 20 用來說明原 DLB 演算法缺失及我們如何修正之網路拓撲及情境

在圖 20 的網路拓撲及情境中，我們的模擬環境參數設定如下：

資料流傳送資料的型態與方式：Poisson process/UDP。

資料流的傳送順序：依序爲N1，t=0秒，頻寬需求爲3Mbps,其所建之CR-LSP之LSPid爲1200。

N0，t=0.5秒，頻寬需求爲4Mbps,其所建之CR-LSP之LSPid爲1100。

模擬過程中所產生的資料流總數：2個。

每個資料流傳送資料的時間：2秒。

總共觀察的時間：2秒。

我們根據圖20來分析，第一個資料流（頻寬需求爲3Mbps）根據TSLB演算法順利的建立CR-LSP（LSPid爲1200，路徑爲2_4_8），並且順利地將資料流傳送。但Ingress LSR在第二個資料流（需求頻寬爲4Mbps）要求建立CR-LSP時，執行TSLB無法替此資料流找到路徑，此時如果根據原DLB演算法，則將重新路由LSPid=1200之CR-LSP。此時整個網路剩餘可用頻寬如圖21所示（扣除替LSPid=1200所保留的頻寬），其選擇的重新路由路徑

將會是2_3_4_8（重新路由之LSPid=1201），但在資料流重新路由後，釋放原路由（即LSPid=1200，路徑爲2_4_8）保留之頻寬，其目的爲騰出足夠的頻寬給新資料流使用，可惜的是此時路徑2_4_6的可用頻寬仍不足新資料流使用，造成新資料流無法找到路徑，傳送失敗。追究其原因，我們發現是因爲4_8之鏈接頻寬被LSPid=1201之CR-LSP用掉了。

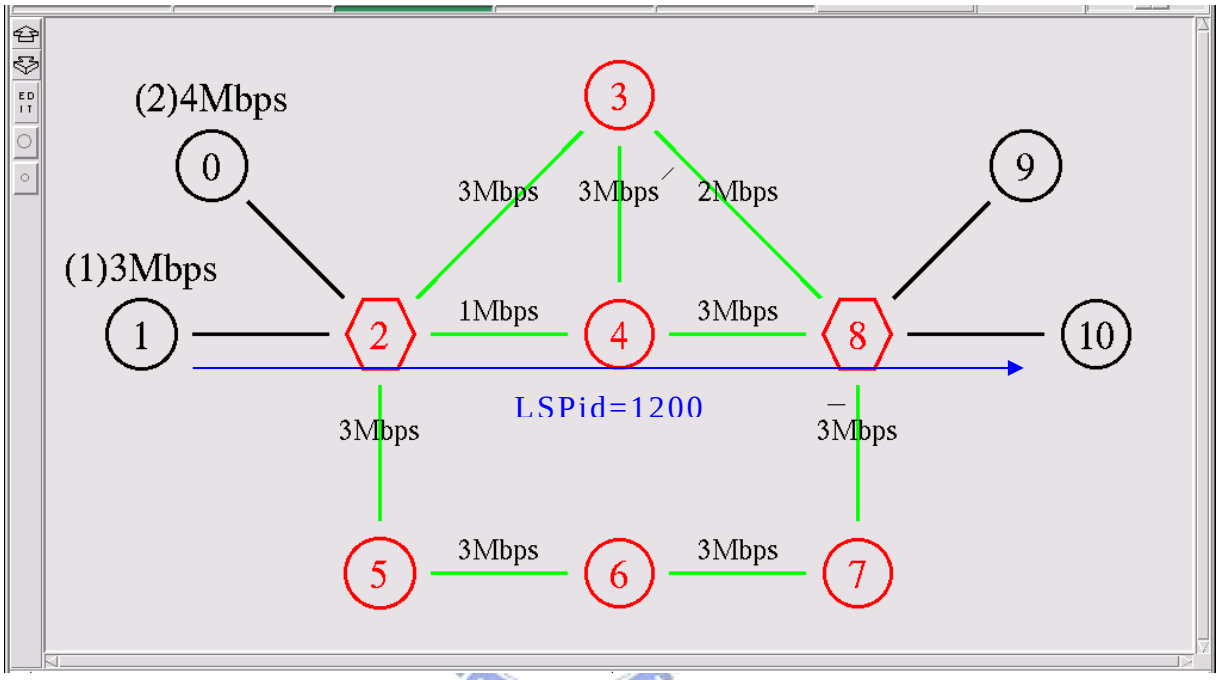


圖 21 LSPid=1200 建立後，整個網路剩餘可用頻寬分佈情形

爲了解決上述之困擾，我們在流程圖中增加了一個動作：沿著從NLC中選到之CR-LSP路徑，額外保留一些頻寬，此頻寬大小爲（目前pending-Traffic上的資料流需求頻寬減去爲此一被選到之CR-LSP保留的頻寬）。此動作的目的是爲避免釋放了選到之CR-LSP，此CR-LSP路徑的頻寬仍不夠新資料流使用。我們利用下圖（圖22）來說明這種想法。

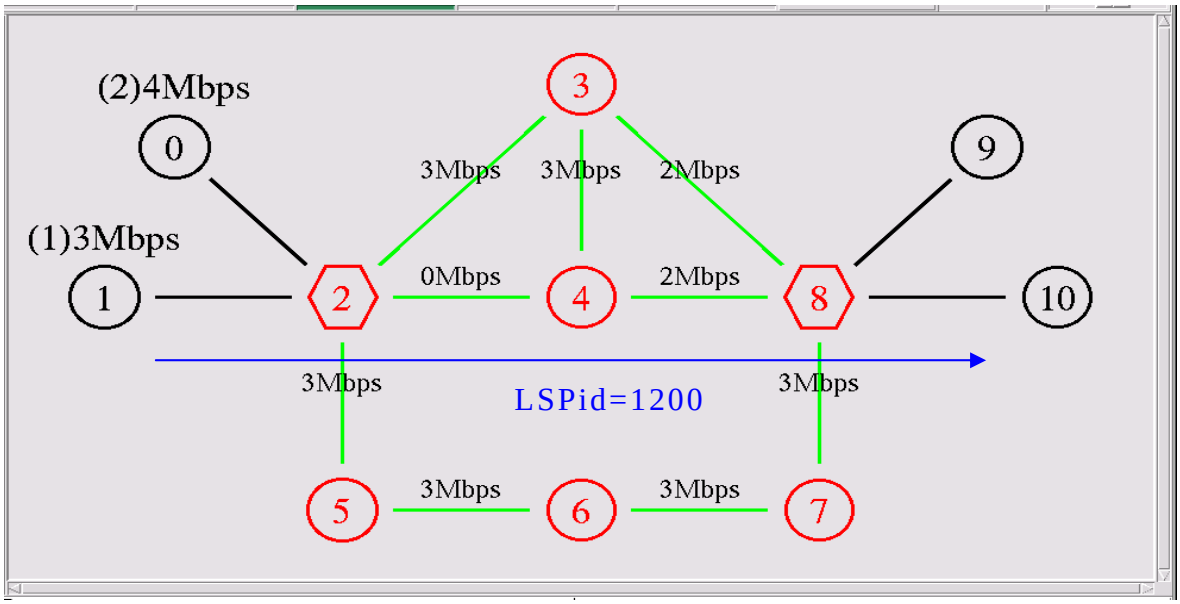


圖 22 LSPid=1200 建立及此路徑額外保留 1Mbps 頻寬後，整個網路剩餘可用頻寬分佈情形

圖 22 爲 LSPid=1200 建立及此路徑額外保留 1Mbps 頻寬後，整個網路剩餘可用頻寬分佈情形。其中 1Mbps 是目前 pending-Traffic 上（即新資料流 LSPid=1100）的資料流需求頻寬減去爲此一被選到之 CR-LSP（即 LSPid=1200）保留的頻寬 （即 4Mbps-3Mbps=1Mbps）。此時被選到之 CR-LSP 之重新路由的路徑將會是選擇 2_5_6_7_8。然後，在釋放 LSPid=1200 路徑上額外保留的頻寬及重新路由 LSPid=1200 之後（重新路由後之 CR-LSP 之 LSPid=1201），新資料流（LSPid=1100）便順利地找到路徑，建立 CR-LSP（路徑爲 2_4_8）。圖 23 和圖 24 爲利用修正後之 DLB 演算法執行圖 20 之網路拓撲及情境的執行結果。由此執行結果，印証了我們上述的這些觀念。

```
[root@ip-22 root]# ns /root/ns-allinone-2.1b8a/ns-2.1b8a/tcl/ex/constraint-routing/reserve1.tcl
--> The result of constraint-based routing for lspid 1200 : Explicit Route=2_4_8
      o The CR-LSP of lspid 1200 has been just established at 0.040391999999999997
--> The result of constraint-based routing for update_lspid 1201 : Explicit Route=2_5_6_7_8
      o The CR-LSP of lspid 1201 has been just established at 0.58147199999999999
--> The result of constraint-based routing for lspid 1100 : Explicit Route=2_4_8
      o The CR-LSP of lspid 1200 has been just released at 0.60203423071930784
      o The CR-LSP of lspid 1100 has been just established at 0.64039200000000018
[root@ip-22 root]#
```

圖 23 使用修正後之 DLB 演算法模擬圖 20 得到之每一條 CR-LSP 之路徑、建立時間和被釋放時間

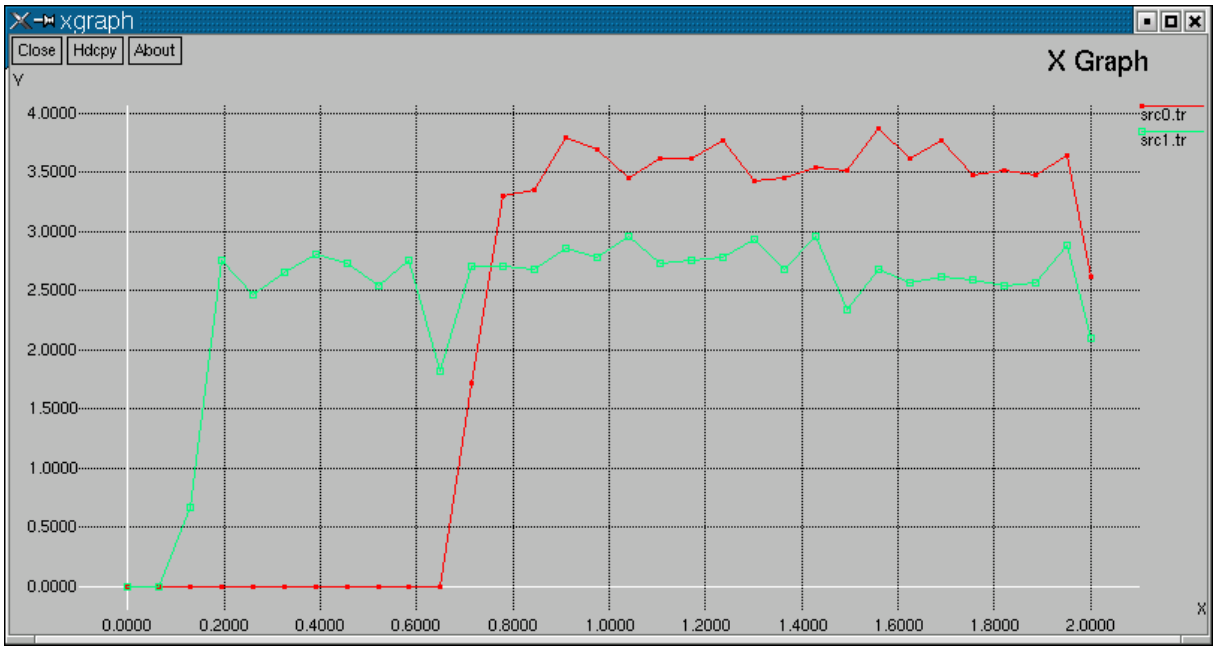


圖 24 使用修正後之 DLB 演算法模擬圖 20 得到之每一個資料流的輸出量

再來，我們將圖 20 中之 4_8 鏈接頻寬改成 7Mbps，其餘不變，如圖 25 所示。我們執行修正後之 DLB 演算法，得到圖 26 及圖 27 的結果。我們發現執行結果與圖 20 執行結果有很大的不同，LSPid=1201 路徑爲 2_3_4_8，LSPid=1100 之路徑爲 2_4_8。我們要強調的是重新路由的路徑可使用原路由的部份鏈接。

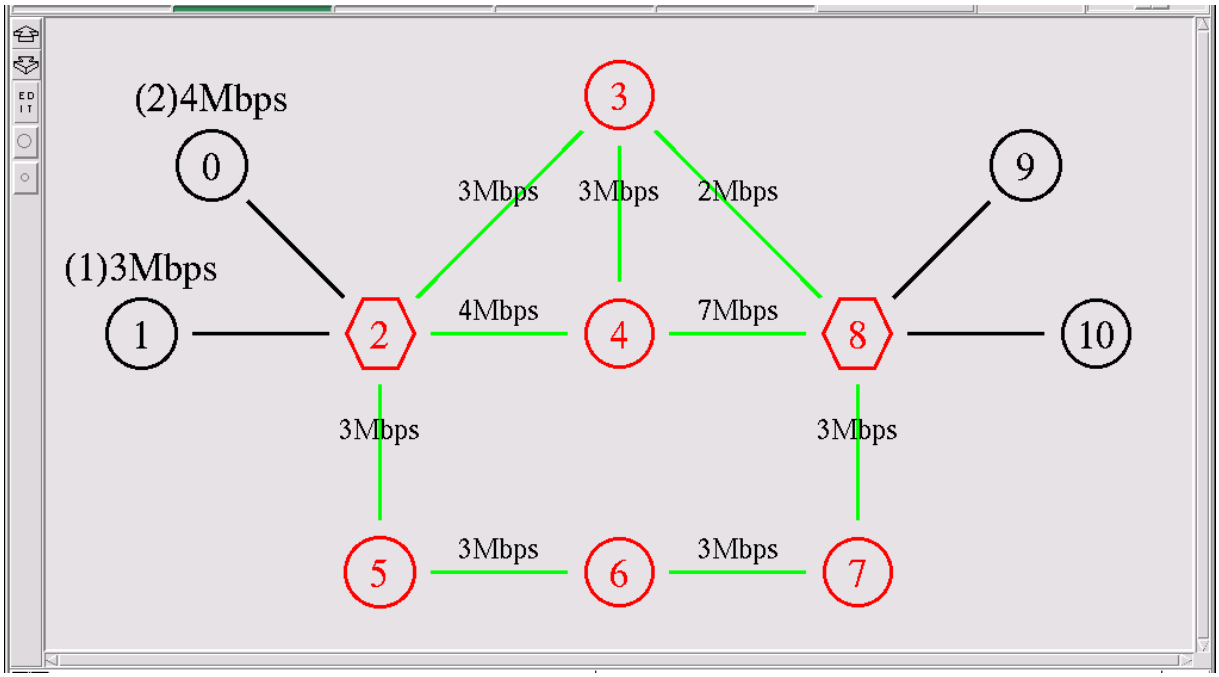


圖 25 將圖 20 中之 4_8 鏈接頻寬改成 7Mbps

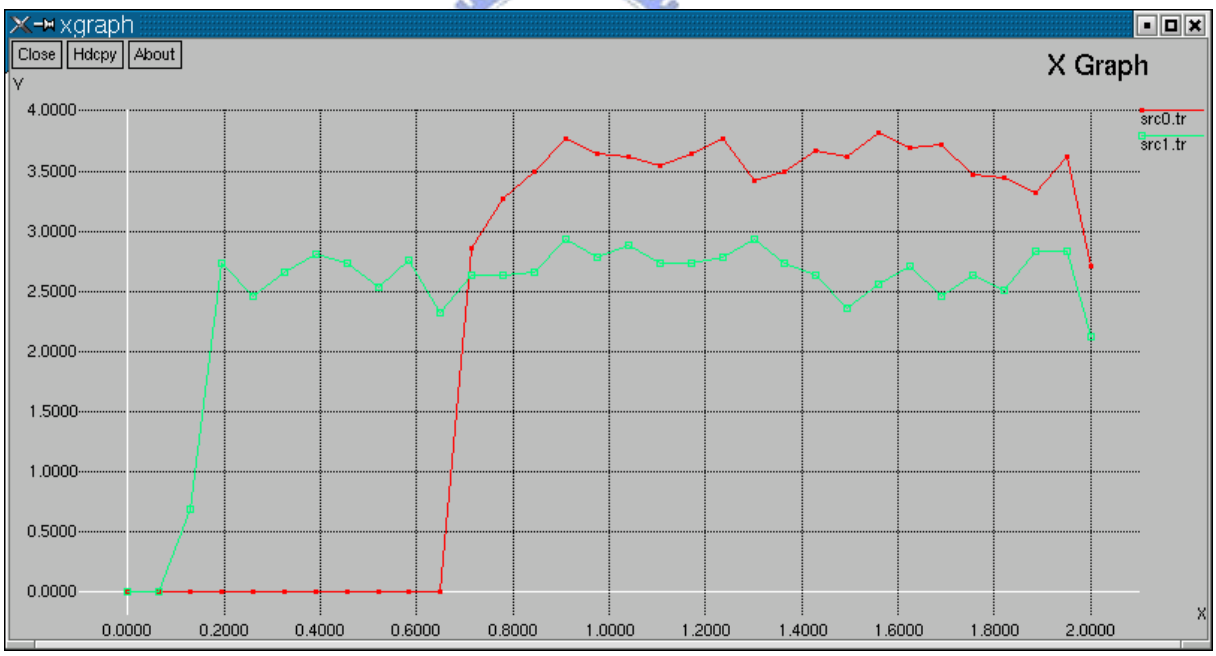


圖 26 使用修正後之 DLB 演算法模擬圖 25 得到之每一個資料流的輸出量

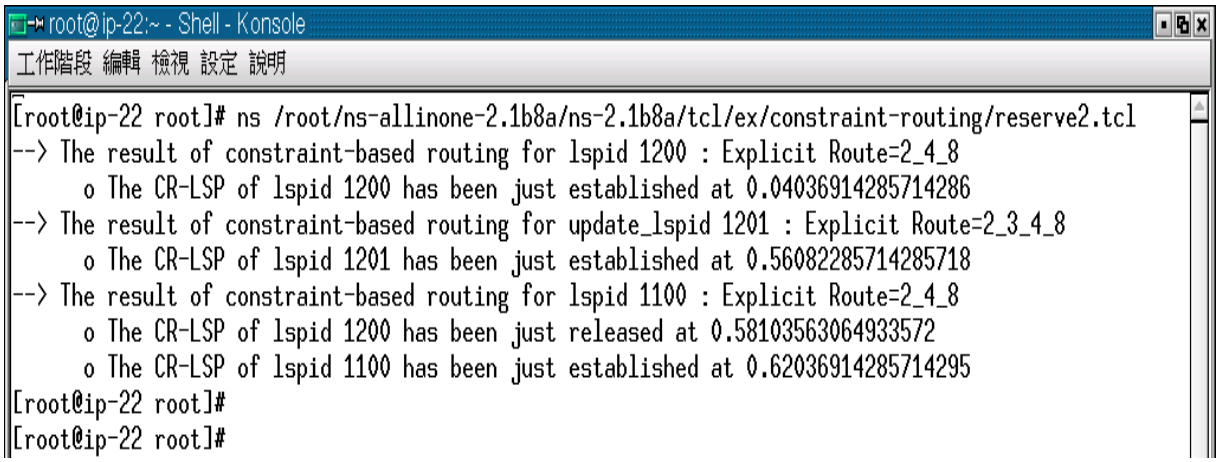


圖 27 使用修正後之 DLB 演算法模擬圖 25 得到之每一條 CR-LSP 之路徑、建立時間和被釋放時間