路由常駐程式的計算量分析

戴維炫 張舜理 林盈達
國立交通大學資訊工程學系
October 16, 2009

摘要

路由常駐程式(routing daemon)負責實現各個路由協定的功能，且根據交換路由資訊及計算最短路徑的結果，更改路由表格的內容，我們關切的是路由演算法執行耗費的時間比例，藉由得知此時間比例，進而了解路由常駐程式的運作。

在此選擇 Zebra 路由軟體作為實驗對象，針對提供的 RIP 常駐程式和 OSPF 常駐程式做量測，且建置在同樣為 250 台路由器之網路拓樸的環境下，藉此對同屬內部閘道協定(Interior Gateway Protocol)的兩個路由協定作其差異性上的分析。實驗結果顯示：RIP 常駐程式中 Bellman-Ford 路徑演算法所耗費的時間比重約為 2.66%、時間絕對值為 0.14 毫秒(ms)；而 OSPF 常駐程式中 Dijkstra 演算法所耗費的時間比重約為 56.75%、時間絕對值為 20.46 毫秒(ms)，由此數據的呈現，可讓我們得知在路由常駐程式中，主要做運算的演算法所耗費的時間比重與其時間絕對值的差異，進而分析其中的運作與主宰效能的關鍵因素。

關鍵詞：Zebra、路由常駐程式、RIP、OSPF、Bellman-Ford 路徑演算法、Dijkstra 演算法

一、 簡介

路由器[1][2][3]在互為連接的網路系統中扮演著交通樞紐般的重要角色，其中路由常駐程式負責實現各個路由協定(routing protocols)，如 RIP(Routing Information Protocol)[4]、OSPF(Open Shortest Path First)[5]、BGP(Border Gateway Protocol)等，且適當地根據交換路由資訊及計算最短路徑的結果，更改路由表格的內容，而主要的最短路徑演算法執行所耗費的時間比重是我們所關切的，藉由分析此時間比重，可了解路由常駐程式運作中主宰效能的因素。

在 Linux 上有幾個相當著名的路由程式，如：routed、gated、Zebra[6]等，由於業界在 Linux 為基礎的網路設備上，大都使用 Zebra，故在此選用 Zebra 作為實驗對象。Zebra 支援以 TCP/IP[7]為基礎的路由程式，且支援多數常見的路由協定。不同於傳統的路由程式以單一程式去提供所有路由協定的功能，Zebra 採取集合多個不同的路由協定，共同去建構核心路由表格。在 Zebra 的軟體架構下，RIPd 為處理 RIPv1、RIPv2 協定的常駐程式，OSPFd 則為處理 OSPFv2 協定的常駐程式，其餘以此類推，而 Zebra 常駐程式是負責寫入路由的核心表格。Zebra 的多程式架構下，每個路由常駐程式都擁有自己的設定檔和終端介面，為了管理和設定的方便，Zebra 有提供一個整合使用者介面的 Shell[8]稱為 Vtysh，可透

過此 Vtysh 切換至任何一個路由常駐程式，並針對相對應的路由協定作設定。

了解 Zebra 的整體架構後，我們將焦點放至 RIP 常駐程式和 OSPF 常駐程式，藉由兩者的差異性做比較與分析，以觀察在相同的網路環境下，運作不同的路由常駐程式，在耗費的時間絕對值和比重上有何差別。以下針對 RIP 和 OSPF 兩個路由協定的差異性做介紹：

RIP (Routing Information Protocol)

1. 距離向量路徑選擇 (Distance Vector Routing)

   每個路由器必須維護一個二維的向量表格(vector table)，表格內記錄本身至每個目的地之間的已知最短距離以及到達目的地所要經過的最佳下一節點。路由器只會和相鄰的路由器交換向量表格內容。當路由器接收到相鄰路由器傳來的向量表格後，會去修正本身的向量表格內容，修正完後再繼續和相鄰的路由器交換，依此類推，逐步地將整個網路的情況擴散到所有路由器。

2. 路由器與其鄰近路由器交換網路的路由資訊

   每個路由器透過廣播 UDP 表頭交換路由資訊，每 30 秒對鄰近的路由器發送路由訊號更新，記錄裡含有整個路由器的路由表格。因此，任何一個路由器可透過與鄰近路由器的路由資訊交換，建立出自己的路由表格。

3. Bellman-Ford 路徑演算法

   利用 Bellman-Ford 路徑演算法建立和更新路由表格。此演算法計算耗費時間短，時間複雜度為 O(nlogn)。

OSPF (Open Shortest Path First)

1. 鏈結狀態路徑選擇 (Link-State Routing)

   每個路由器必須維護一個複雜的鏈結狀態資料庫(Link State Database)，這個資料庫中的資訊包含整個網路的連線狀況及可用資源。每個路由器都要知道整個網路的拓樸資訊。而這些資訊傳遞是透過廣佈(flood)鏈結狀態公告(Link State Advertisement)的封包到整個網路上。而每個路由器會根據這些封包去計算建立自己的路由表格，直到建立整個網路拓樸的資訊為止。

2. 路由器與整個網路分享鄰近路由器的路由資訊

   每個路由器會定期計算和鄰近路由器之間的費用，這費用可能和佇列延遲、頻寬等因素有關。並建立鏈結狀態封包，廣播給網路上的所有路由器；同樣地，每個路由器也收到所有其他路由器的鏈結狀態封包，計算出到達其他路由器的最短路徑，並建構路由表格。因此，任何一個路由器都可透過網路收到所有鏈結狀態，以算出自己的路由表格。

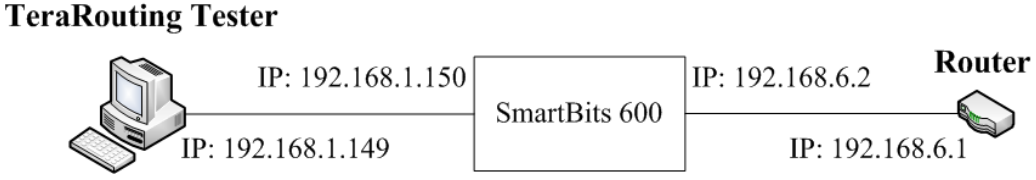
3. Dijkstra 演算法

   每一個路由器收到其他所有路由器的鏈路狀態封包時，必須計算到達任何一個路由器的最短路徑。而尋找最短路徑的方法是利用 Dijkstra 演算法，並建立路由表格。此演算法計算耗費時間長，時間複雜度為 $O(n^2)$。

在針對 RIP 和 OSPF 兩個路由協定的差異性做比較後，我們將建置相同的網路拓樸環境，分別量測兩者的執行時間，以觀察與分析路由常駐程式的運作。在下一節中，我們將介紹實驗環境的建置。第三節針對量測方法與結果做說明。第四節則對上一節的結果做分析探討。最後是我們的結論。

二、 環境建置

本實驗的路由器(Router)以 Cent-OS 5.3 架構的 Linux 為實驗平台路由器，其上安裝 Zebra 路由程式的最新版本，zebra-0.95a，並配備兩張網路介面卡。實驗環境包含此路由器，與一台 SmartBits 600 搭配 Terarouting Tester 軟體。Terarouting Tester 用以設定網路拓樸與路由協定，透過 SmartBits 600 傳送該流量給待測的路由器。在此模擬的網路拓樸為 250 台路由器。實驗環境的建置如圖一所示。



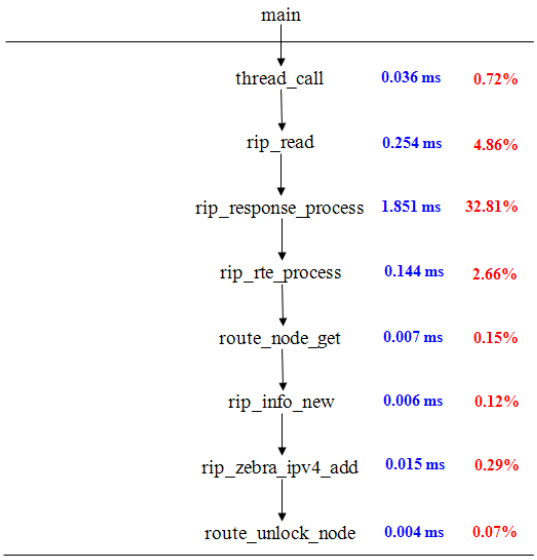
圖一 實驗環境建置圖

三、 量測方法與結果

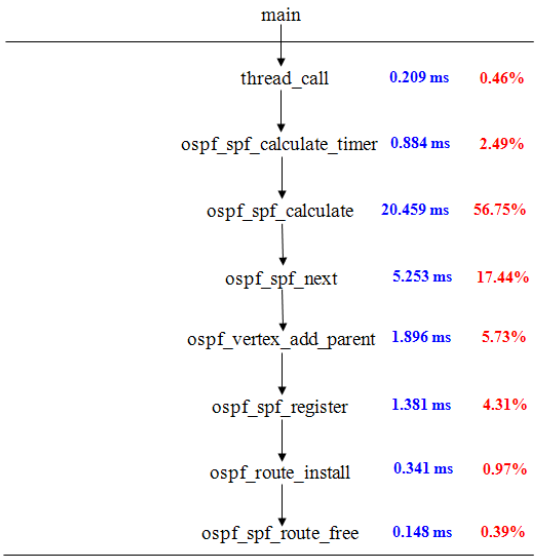
實驗目的在於量測 RIP 和 OSPF 兩個常駐程式中，實作最短路徑演算法之函式運作占整個常駐程式運作的時間比例，並比較其差異性。在 RIP 常駐程式中，實作 Bellman-Ford 路徑演算法的函式為 rip_rte_process()；而在 OSPF 常駐程式中，實作 Dijkstra 演算法的函式則為 ospf_spf_calculate()。

量測方法為先啟動 Zebra 常駐程式，接著執行 RIP 或 OSPF 常駐程式，同時利用 Terarouting Tester 模擬網路拓樸並透過 SmartBits 600 傳送該路由協定的流量，與待測的路由常駐程式交換路由資訊。藉由工具 gprof[9][10]的測試結果得知函式之間的呼叫關係以建構 call graph，詳見圖二和圖三。根據 call graph 呈現的資訊，並使用 clock_gettime()[11]函式來產生時間戳記，以得知程式執行中當時的 CPU 時間。作法是將其插入目標函式的前後，擷取起始點和結束點，並將數值做相減之動作，即可得到該函式的執行時間，時間單位為毫微秒(ns)。而各個函式的執行時間經過換算成占總時間的百分比後，將其實作最短路徑演算法和其他耗費時間比重較高的函式列出，結果如圖四和圖五所示。

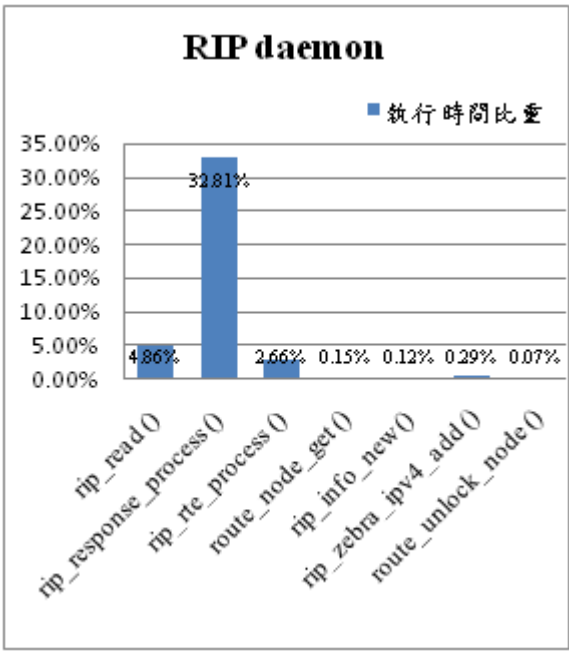
在網路拓樸模擬的部分，無論是運作 RIP 還是 OSPF 常駐程式，皆以模擬 250 台路由器的環境為基準，讓兩個路由常駐程式在相同的環境上作量測，藉此驗證兩個協定的差異。並針對實作最短路徑演算法的函式作對照方式的實驗，模擬在不同數量路由器下其函式執行時間的差異，以驗證兩個最短路徑演算法之時間複雜度的差異，並藉此觀察兩個路由常駐程式在運作時間耗費上的異同。



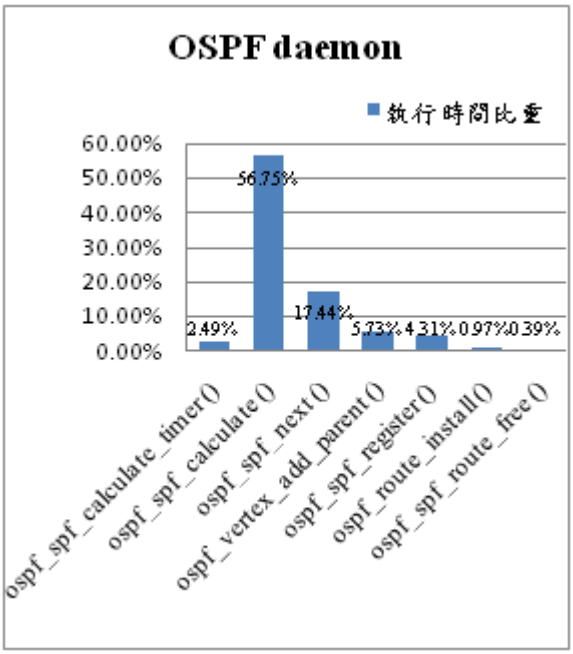
圖二 RIP daemon 的 call graph



圖三 OSPF daemon 的 call graph



圖四 RIP daemon 函式執行時間比重圖



圖五 OSPF daemon 函式執行時間比重圖

根據圖二可得知 RIP 常駐程式的運作流程，當收到路由封包時，會將封包中 rte(routing table entry)資訊當成參數傳入 rip_rte_process()函式，並藉由 route_node_get()函式取得路由表格中各節點的路由資訊，以開始執行 Bellman-Ford 路徑演算法。透過 rip_info_new()函式檢查到達目的地節點的路由資訊是否存在，沒有則會建立新的路由資訊，並將到達目的地的最短距離及所要經過的下一節點存入，最後再藉由 rip_zebra_ipv4_add()函式將新的路由資

訊加入路由表格中。

由圖三可得知 OSPF 常駐程式的運作流程，其中 ospf_spf_calculate()為實作 Dijkstra 演算法的函式，用以計算任一區域的最短路徑，其中各節點以自己為根(root)來建立最短路徑樹。而當收到鏈結狀態封包時，也會執行 ospf_spf_calculate()函式，首先將自己設成根，然後將它移到暫時節點列表，接著透過 ospf_spf_next()函式取得節點的列表(list)資訊，判斷暫時節點列表是否為空的，若為空的話則結束完成演算法；若是不為空的話，將暫時節點列表中具最短路徑的那個節點移到永久節點列表，由 ospf_vertex_add_parent()函式完成此動作，最後由 ospf_spf_register()函式將此節點加入最短路徑樹中。

四、 結果分析與探討

根據實驗結果，整理出以下幾點的觀察結果，並針對常駐程式的運作過程與 RIP 和 OSPF 特性的差異，予以分析與探討。

1. 交換路由資訊

RIP 和 OSPF 兩個協定在交換路由資訊的概念是相去甚遠的，RIP 只會與鄰近的路由器交換路由資訊，且資訊裡含有整個網路中路由器的路徑表；反觀 OSPF 則是傳送鏈結狀態封包給區域內所有其他的路由器，而藉由累積這些鏈結狀態封包來建立整個網路的拓撲資料庫，以達到交換路由資訊的動作。前者主要實作於 rip_response_process()函式，其執行時間比重約為 32.81%，時間為 1.85(ms)；後者則是實作於 ospf_spf_next()函式，其執行時間比重約為 17.44%，時間為 5.25(ms)。由於 RIP 交換的資訊裡包含了整個網路的路徑表，因而占用不少的網路資源，由數據所示，其時間比重明顯較 OSPF 的高。

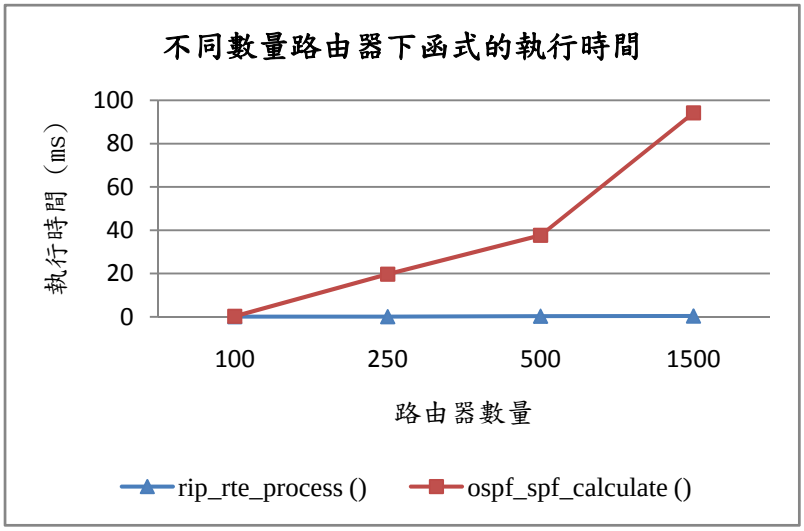
2. 計算最短路徑

RIP 使用 Bellman-Ford 路徑演算法來計算最短路徑，而 OSPF 則是使用 Dijkstra 演算法。前者是透過跳躍計數（hop count）作為尺度來衡量路由距離，跳躍計數是一個包括到達目標所必須經過的路由器的數目，最多支援的跳躍計數為 15，超過 16 者則視為不可到達；後者則是藉由網路拓樸資料庫以考量網路情況來計算其路徑的耗費，進而計算最短路徑。前者主要實作於 rip_rte_process()函式，其執行時間比重約為 2.66%，時間為 0.14(ms)；後者則是實作於 ospf_spf_calculate()函式，其執行時間比重約為 56.75%，時間為 20.46(ms)。由於 OSPF 須確認不在自己路徑樹中的每個節點的路徑資訊，其中路徑樹是由 n 個節點所組成，因此需要 n(n+1)/2 的比較動作，造成 $O(n^2)$的時間複雜度；相較之下，RIP 使用堆積(heap)資料結構，平均時間複雜度為 O(nlogn)，因此，RIP 時間的耗費少很多。

3. 時間複雜度比較

RIP 的時間複雜度為 O(nlogn)，OSPF 時間複雜度為 $O(n^2)$，相較起來可推論得知前者的計算時間增加較緩，但後者的則會有明顯地成長，為了印證

時間複雜度的差異，建置不同數量路由器的環境量測兩者的計算時間，結果如圖六所示。



圖六 不同數量路由器下函式的執行時間

4. 更改路由表格

在交換路由資訊及計算最短路徑後，接著就是根據結果來更改路由表格。RIP 是實作於 rip_zebra_ipv4_add()函式，其執行時間比重約為 0.29%、時間為 0.02(ms)，OSPF 是實作於 ospf_route_install()函式，其執行時間比重約為 0.97%、時間為 0.34(ms)。此兩者所占的時間比重皆很小，因為皆只是將結果資料當成參數傳給下層，由 Zebra 常駐程式負責寫入路由的核心表格，以達到更改路由表格的動作。

5. 其餘執行時間比重高的函式

在 RIP 常駐程式中，rip_packet_dump()函式占相當高的比重，是由 rip_read()函式所呼叫的，其執行時間比重約為 22.66%，時間為 1.27(ms)。其主要的功能是當接收到由路由資訊時，會將經過判讀過後的認可資料作轉儲的動作，將從別台路由器傳來的路由資訊儲存至本端中，利用自身建立的執行續結構來存放這些資訊，當成輸入資訊提供給之後的函式使用。因此耗費許多的時間。

五、 結論

路由常駐程式負責實現各個路由協定的功能，且根據交換路由資訊及計算最短路徑的結果，更改路由表格的內容。在此運作過程中，各個函式執行的時間不盡相同，所占的時間比重亦是未知。而且現今的路由軟體能提供多個路由協定的常駐程式運作，故其中的計算量是讓人好奇，但只知道單一路由常駐程式的計算量在意義上又甚為有限，因此本文針對最常用且具代表性的兩個路由常駐程式：RIP 和 OSPF，做量測與分析，藉由時間結果驗證其差異性，並從中得知各個函式執行的時間和比重，藉此達到分析路由常駐程式的運作與其中主宰的關鍵因素。

參考文獻

[1] 楊佳欣、林盈達；『追蹤與測試 Linux 路由器』；*網路通訊*；100 期，1999 年 11 月。

[2] 張政賢、陳恆毅、林盈達；『路由器工作原理』；*Internet Pioneer 光碟用刊*；57 期，1999 年 2 月。

[3] Linux Router Introduction, http://tryseo.info/6/linux_server/0230router.php.

[4] C. Hedrick, "Routing Information Protocol", *IETF RFC 1058*, June 1988.

[5] J. Moy, "OSPF Version 2", *IETF RFC 2328*, Apr. 1998.

[6] GNU Zebra, http://www.zebra.org/download.html.

[7] TCP/IP Introduction, http://www.study-area.org/network/network_ip.htm.

[8] Shell Introduction, http://linux.vbird.org/linux_basic/0320bash/csh/.

[9] GNU gprof, http://www.cs.utah.edu/dept/old/texinfo/as/gprof_toc.html.

[10] Ying-Dar Lin, ”Network Benchmarking Methodologies and Tools”, http://speed.cis.nctu.edu.tw/~ydlin/course/cn/nsd2009/Benchmarking.pdf.

[11] Linux Document “clock_gettime()”, http://linux.die.net/man/3/clock_gettime.