RFC: 統計與實作

論文領域: 其他—標準

黃俊穎　林盈達

國立交通大學資訊科學系
新竹市大學路 1001 號
TEL：（03）5712121 EXT. 56667
FAX：（03）5712121 EXT. 59263
E-MAIL：huangant@cyber.cs.ntou.edu.tw, ydlin@cis.nctu.edu.tw

主要聯絡人：黃俊穎

摘要

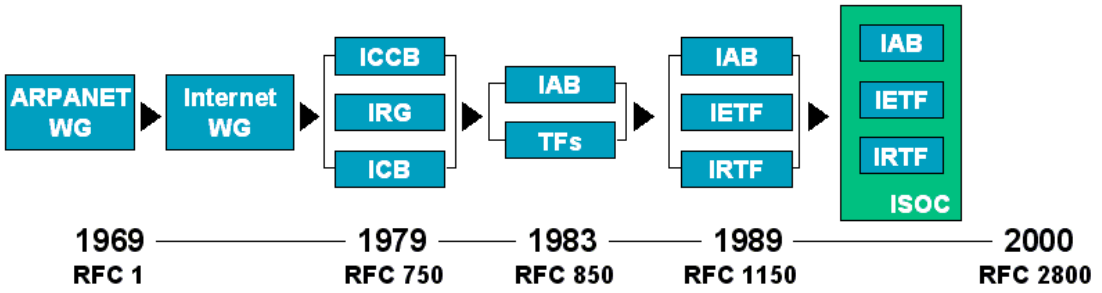
Internet 是由近 3000 份 RFC 所定義，要了解 Internet 的發展、架構、使用的通訊方式等相關資訊，或是參與 Internet 的發展，最直接的方式就是從閱讀、了解 RFC 下手。本文首先介紹 IETF 與 Internet 的歷史和 RFC 的形成過程，然後針對 RFC 的產量、分類及狀態做統計，並介紹 IETF 這個 Internet 社群的運作方式及活動(WGs, Meetings)，最後以 RIPv1/Router 相關之 RFC 文件與 Linux router 內部實作相對應爲例說明 RFC 與實作之關係。

關鍵字：IETF, RFC, WG

一、IETF 歷史與 RFC 形成過程

IETF 歷史

RFC 文件主要探討的議題就是與 Internet 相關的事物。舉凡通訊協定的定義、實作、傳輸資料的型態格式、FAQ，還有一些幽默小品文章等。早期的 RFC 是由 ARPANET Working Group、Internet Working Group 所提出產生，而經過數年的演變後，這些組織有的改名，有的分開，有的合併，現在的 RFC 大多是由 IETF(Internet Engineering Task Force) 的 Working Groups 所提出(也有以個人名義提出)的，然後經由 IESG 審核後而成爲 RFC。IETF 這個組織的形成與 RFC 篇數的關係如圖一所示。



圖一：IETF 與 RFC 年代表[1]

Internet 社群的源起就從 ARPANET 時代開始。研究人員利用網路上的服務如 e-mail、檔案共享、遠端存取等互相連絡進行計劃，每個計畫就形成一個 Working Group (WG)，通稱爲 ARPANET Network Working Group，只由 ARPANET 提供各個計劃的基礎需求。當 Internet 開始演進時，Network Working Group 便更名爲 Internet Working Group。

1970 年代末期，由於 Internet 的成長使得單一社群無法負荷，因此，在 DARPA 負責管理 Internet 計畫的 Vint Cerf 便將 Internet WG 拆分爲三個社群，分別爲 Internet Cooperation Board (ICB)—負責歐洲國家方面的計劃；Internet Research Group (IRG)—提供交換資訊的環境；以及 Internet Configuration Control Board (ICCB)—協助 Cerf 管理發展迅速的 Internet 活動。

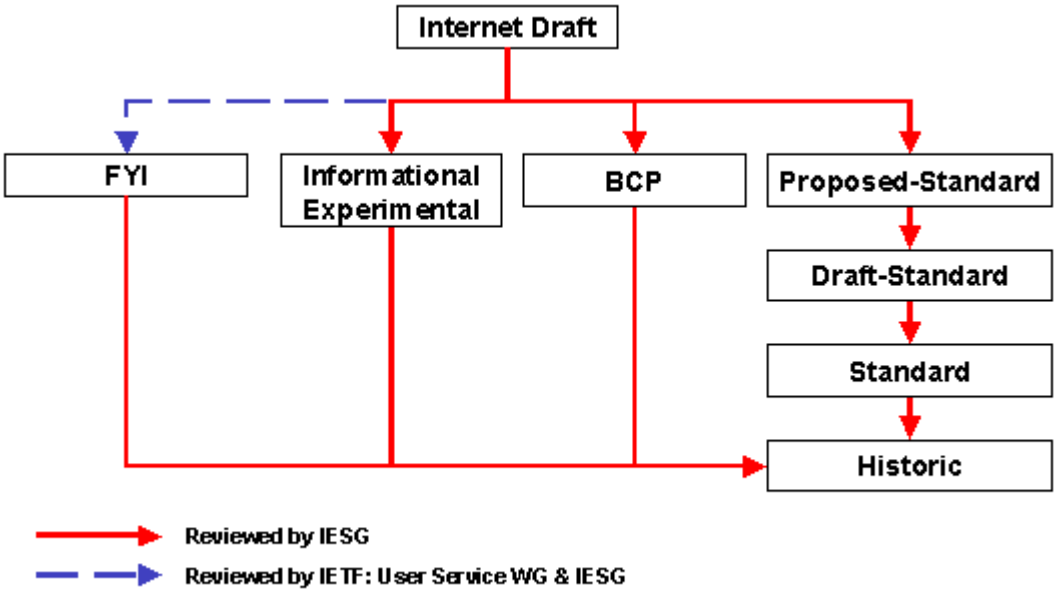
1983 年，在 DARPA 負責管理 Internet 計畫的 Barry Leiner 認爲這些社群工作需要重新分配，因此原來的 ICCB 被 Task Forces (TFs) 所取代，其他的社群就解散了。此外，再由各個 TF 的領導者組成 Internet Activities Board (IAB)。

IAB 經過一些人事上的變動後，當 Phill Gross 成爲 TFs 中 Internet Engineering Task Force (IETF) 的領導者後，到了 1985 年之後，由於 Internet 技術有長足的進步，愈來愈多人參加 IETF 的研討會，使得 IETF 愈來愈龐大而需要在內部再分割出更多的小 WG。IETF 持續的成長，因此後來又在 IETF 之下區分幾個領域(Area)，將數個 WGs 合併至同一個領域下分層管理。而由這些領域的領導者再組成 Internet Engineering Steering Group (IESG)。而其他 TFs 就合併成爲 Internet Research Task Force (IRTF)。

1992 年，IAB 被重組及更名，使得 IAB 成爲 Internet Society (ISOC) 的下層社群，也使得 IAB 與 IESG 有較「對等」的關係。

RFC 形成過程

IETF 這個組織裡共分爲八大領域(Area)[2]，每個領域裡有一到二個 Director，領域之下還有許許多多不同的 WGs 。WGs 內部或是 WGs 之間是透過 mailing list 來互相討論、溝通的，而每一年 IETF 會舉辦三次研討會，讓來自世界各地不同 WGs 的人可以聚在一起面對面的討論。Internet 上的任何人都可以參加 IETF 且不需要費用，只要加入感興趣的 WG，訂閱他們的 mailing list，參與他們的討論，如果有時間，再參加 IETF 所舉辦的研討會，這樣就算是加入 IETF 了。大部份的 RFC 就是這些 WGs 討論出一定的成果後所提出的。一般 RFC 形成的大略過程如圖二所示。



圖二：RFC 形成過程示意圖

要發佈一份 RFC 需要經過一些程序[3][4]，而每個階段程序都有 Internet Engineering Steering Group (IESG) 從旁負責審核的工作。要提出一份 RFC，得先提出一份 Internet Draft(ID) 的文件，這份文件會被放在 IETF 的 Internet-Draft 目錄中供大家檢視。ID 提出一段時間(至少二週)之後，ID 的作者可以送封 e-mail 給 RFC-Editor(由 Internet Society 提供資金維護) 註明要將 ID 升爲 Informational 或 Experimental RFC，由 RFC-Editor 要求 IESG 對 ID 進行審核的工作，在成爲正式 RFC 之前，ID 的作者隨時都可以將該 ID 做更新，如果在這份 ID 被送出或更新後的六個月之內既沒有再被更新也沒成爲 RFC，那麼這份 ID 就會從 IETF 的

Internet-Drift 目錄被移除，同時作者也會被告知。如果 ID 已經審核通過準備要成爲 RFC 了，那麼 ID 的作者可以有 48 小時再次的檢查文件中是否需要有其他的修正如拼字錯誤、參考資料之類的。因爲一旦成爲 RFC 之後，其內容就不能再被修改了。

每個 RFC 都有不同的狀態，包括 STD(Standard)、Historic、BCP(Best Current Practice)、FYI(For Your Information) 和非標準(Informational/Experimental)五大系列。所謂 STD 就是已經成爲 Internet 上標準協定的 RFC 文件；而 BCP 則是可以達成某件事最好的方式，或是定義 IETF 的原則；FYI 則是包含一般的資訊像是 FAQ、Guidelines 之類的。STD、BCP 和 FYI 這三種特別狀態的 RFC 除了原來的 RFC 編號之外，還有另外的編號。比如說第一篇成爲 STD 的 RFC 就會有一個 STD0001 的編號；第一篇成爲 BCP 的 RFC 則會有一個 BCP0001 的編號，依此類推。

STD 這一系列的 RFC 文件分爲三個等級，分別是 Proposed-Standard，Draft-Standard 和 Standard。一份 RFC 要成爲 Standard 得經歷過這些等級，而每一個等級都有不同的限制。以 Proposed-Standard 來說，他的限制最少，只要是被認爲穩定、有價值且被感興趣的的文件就能被 IESG 提出成爲 Proposed-Standard(而非 Informational/Experimental 或是其他狀態的 RFC)，至於需不需要有任何實作或是使用的經驗，則由 IESG 自行決定。而要由 Proposed-Standard 再前進成爲 Draft-Standard 除了至少要等待 6 個月之外，至少還要有 2 份不同的實作和成功的使用經驗。而要從 Draft-Standard 再演進爲 Standard 則至少要等待 4 個月，同時得有明顯成功通用的實作和使用經驗，經過 IESG 認可後，才能成爲 Standard。一個已經成爲標準的 RFC 文件如果其內容已經過時或不適用了，那麼 IESG 可能就將該標準標示爲 Historic 的狀態。

如果是要成爲 BCP 狀態的 RFC，那麼過程就類似成爲 Proposed-Standard，只要要求文件成爲 BCP，經過 IESG 認可即可。FYI 的形成過程就比較不一樣[5]，如圖二所示，一份文件如果打算要成爲 FYI，在送交 IESG 審核之前，得先交由 IETF 的 User Service Working Group 審核，往後就和一般 RFC 的過程相同。另外有一點不同的是，FYI 的列表是由 SRI Network Information Center (NIC) 所維護的，並不是由 IETF 或是 RFC-Editor 所維護的。

RFC 的編號大致上是照審核通過的順序所編的，不過其中有些編號是有特別用途的。比如說編號爲 99 結尾的 RFC 通常就是前 100 篇的概要介紹；編號爲 00 結尾的通常就是 IAB

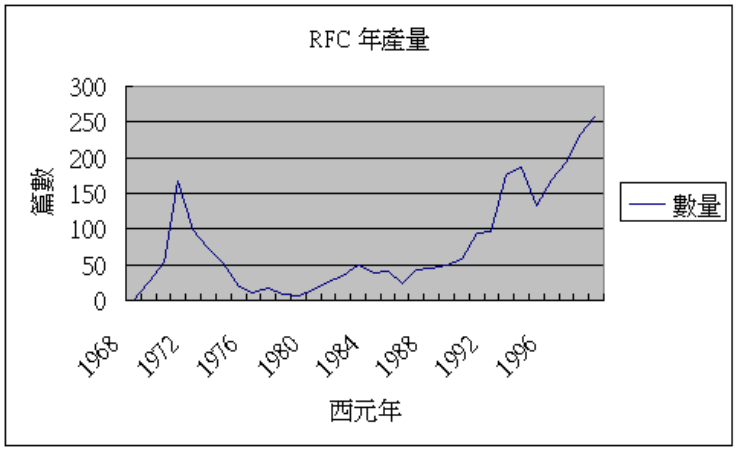
Official Protocol Standards，概述目前 RFC 標準的狀況。

二、RFC 相關的統計

產量

截至目前爲止，RFC 已經編到第 2877 號了，不過其中有許多號碼是沒有相對應的 RFC，有些是則是現在無法從線上取得。我們針對 RFC 2877 以前約 2700 篇做了簡單的分類統計，了解目前 RFC 的分佈情形。

首先是年產量的統計，從圖三中，我們可以看出 RFC 剛開始的幾年數量成長的很快，後來量就趨於平均的低，而近五年來，又成穩定快速的成長。我們分析其原因是由於剛開始要產生 RFC 比較容易，許多與技術無關事務如開會通知，或是 WG 內互相的意見交流等都弄成一篇 RFC，因此就有很多 RFC。後來的幾年規定比較明確，限制也比較多，所以量就降下來了。近年來，由於 Internet 的蓬勃發展，許許多多人投入網路相關的技術研究，因此 RFC 的數量又開始快速成長了。

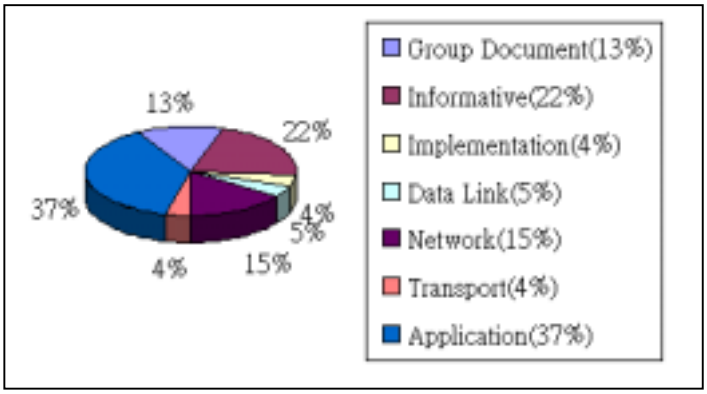


圖三：RFC 年產量

分類統計

接下來，我們將 RFC 分爲 Group document, Informative, Implementation 以及 TCP/IP Layer 四層(Data Link, Network, Transport, Application) 共 7 大類[6]，以了解在這麼多形形色色的不同文件中，到底大家都是在討論些什麼東西。其中，Group Document 這一類主要的內容包含會議通知、會議結論、RFC 的簡介、RFC 概略的整理，以及一些小作品如詩、短文等；Informative 這一類則包含了一些新點子、討論、檔案的格式、字元的表示方式等；Implementation 則是和實作相關的文件，包括一些經驗及測試的結果；而 TCP/IP 那四層則是依照該 RFC 所探討的技術是屬於哪一層而分類。

我們統計的結果如圖四所示，可以看出 RFC 文件中比例最多的是屬於 Application 層的討論，其次分別是 Network 層和 Group Document。



圖四：RFC 分類比例

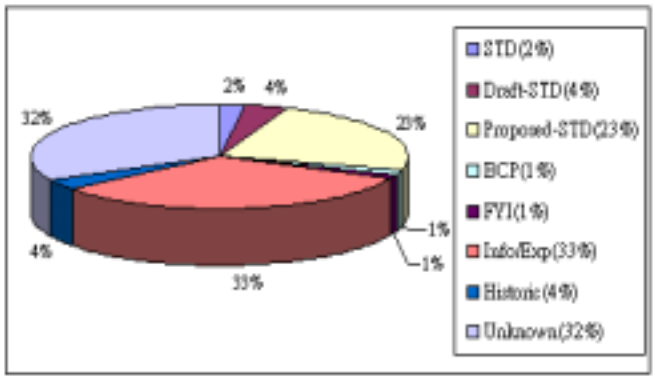
除此之外，我們也針對屬於 TCP/IP 四層的 RFC 做更細部的分類。如表一所示。

Layer	Category	Count	Layer	Category	Count
Data Link	ATM	15	Transport	TCP	30
	PPP	70		UDP	5
	SLIP	5		Others	45
	Others	30	Application	DNS	40
Network	ARP/RARP	10		FTP	30
	BOOTP/DHCP	20		HTTP/HTML	20
	ICMP/ICMPv6	10		MIME	50
	IP/IPv6	100		SMTP	30
	Multicast	20		SNMP/MIB	200
	RIP/BGP/OSPF	50		TELNET	70
	Others	200		Others	560

表一：TCP/IP 四層 RFC 細部分類

狀態統計

另外，我們再來看看在這麼多 RFC 中，有多少是屬於 STD、FYI、或是 BCP 系列的[7]，如圖五所示。其中，狀態爲 UNKNOWN 的 RFC 爲較早期的 RFC(皆在 RFC 1129 之前)，由於當時並沒有明確形成 RFC 的流程，因此許多 RFC 的狀態就維持在 UNKNOWN。一般的 RFC 要成爲標準要經過的時間是很長的，所以在 RFC 中成爲標準的文件並不多。

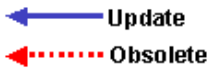


圖五：RFC 狀態比例

看完表一和圖五，我們可以發現在數以千記的 RFC 中，真正我們日常網路生活中有常常在使用的通訊協定，或是成爲 Internet 標準的 RFC 並不多，可是爲什麼有那麼多的 RFC 呢？其實，在這麼多 RFC 中，許多 RFC 都已經被更新、淘汰或是該技術不再使用了，RFC 中所探討的技術在日常生活中會用到的量其實並不大，再加上一份 RFC 一旦被產生出來後，就不能再做修改、或刪除的動作。經年累月的更新、淘汰下來，結果就是有一大堆同性質的或是「歷史性」的 RFC 供人留念。還有另一種可能是由於一個通訊協定的發展並不是一次就可以完全發展完成，因此同一個通訊協定可能就由好幾份 RFC 共同描述才得以完整。

以 TELNET 應用爲例，在 70 篇 TELNET 的文件中，除了通訊協定主幹本身之外，光是以單一 RFC 描述一個 TELNET Option 的文章就有將近 50 篇之多。剩下的有 8 篇是直接與 TELNET 協定本身有關，其他的就是對這個協定的建議、討論及應用之類的文章。TELNET 協定與其 RFC 的關係如圖六所示。

# of Option RFCs	Protocol RFCs	Year	Month	Title
1971~1980 15	137	1971	Apr	Telnet Protocol – A Proposed Document
	139		May	Telnet Protocol – A Proposed Document
	158		May	Telnet Protocols
	318	1972	Apr	Telnet Protocols
	435	1973	Jan	Telnet issues
	495		May	Telnet Protocol Specifications
	764	1980	Jun	Telnet Protocol Specifications
1981~1990 20	854	1983	May	Telnet Protocol Specifications
		~ 1990		
1991~2000 15		~ 2000		



圖六：TELNET 相關 RFC 之間的關係

三、IETF 活動

就如同之前所提到的，IETF 的運作方式主要是透過 mailing list 和一年三次的研討會。而

每一次研討會就是讓各個運作中的 WG 發表、討論成果。截至目前爲止，IETF 已經有 48 次的研討會了，每次爲期約 5 至 6 天，舉辦的地點幾乎每次都不一樣(由主辦單位決定，每次的主辦單位也不一樣)，不過大多是在美國。近 3 年研討會的時間地點如表三。

Meetings	Date	Year	Country	Attendees
48th	Jul.30-Aug.4	2000	USA	N/A
47th	Mar.26-31	2000	Australia	N/A
46th	Nov.7-12	1999	USA	2379
45th	Jul.11-16	1999	Norway	1710
44th	Mar.14-19	1999	USA	1705
43rd	Dec.7-11	1998	USA	2124
42nd	Aug.24-28	1998	USA	2106
41st	Mar.30-Apr.3	1998	USA	1775

表三：近三年的 IETF 研討會[8]

我們可以從每次 IETF 的研討會中看出這八大領域中哪些 WG 是比較活躍的。表四列出第 47 次 IETF 中有參與的 WG。

Area	Working Groups	Count
Application	calsch, cnrp, conneg, imp, fax, trade, ipp, ldap, ldapext, msgtrk, rescap, deltav, blocks, b2bxml, qualdocs, imapext, spatial, vpim	18
General	lww	1
Internet	dnsext, dhc, frnetmib, ipfc, ipcdn, ipngwg, idn, l2tpext, zeroconf, fickle, itrace, ipo	12
Operations And Management	adslmib, aaa, bmwg, snmpconf, dnsop, tewg, mboned, nasreg, ngtrans, policy, rmonmib, rap, snmpv3, rperfman, tmnsnmp	15
Routing	bgmp, gsmp, mobileip, idmr, manet, msdp, mpls, pim, udlr, vrrp,	10
Security	cat, ipsp, ipsec, ipsra, idwg, pkix, smime, xmldsig, syslog	9
Transport	avt, diffserv, ecm, ippm, iptel, issll, megaco, malloc, mmusic, nat, nfsv4, pilc, rmt, rohc, spirits, sip, sigtran, enum, tsvwg, ips, foglamps, sip323, vomlps	23
User Service	Uswg, weird	2

表四：47th IETF Meetings[9]

由於 WG 是由 IETF 爲了解決某些 Internet 上的問題而成立的[10]，一但問題有了顯著的結果而該 WG 不再有其他的任務後，該 WG 可能就被解散。不過如果一個 WG 有產生

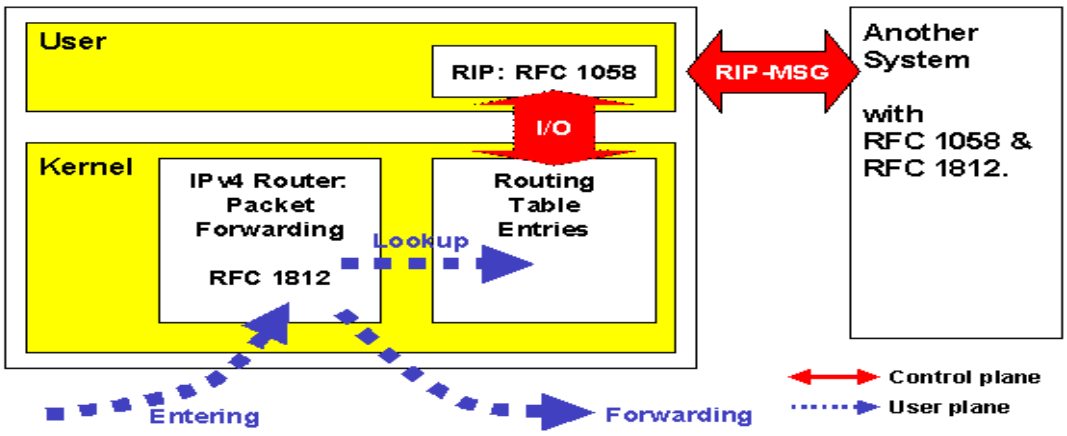
Proposed-Standard 或是 Draft-Standard，那麼這個 WG 就會暫時停止活動但不會被解散。因爲該 WG 還得繼續利用其 mailing list 來探討是否還要再繼續將 Proposed-Standard 或是 Draft-Standard 升爲 Draft-Standard 或 Standard。

如果一個 WG 沒有辦法順利的解決問題，那麼只有三個選擇，即：重新定位要解決的問題、更換 WG 主席、或是解散該 WG.。所以在 IETF 將近 130 個 WGs 裡，有 90 個 WGs 參加研討會，可以看出其實大部份的 WG 都是在「工作中」而且蠻活躍的。

四、閱讀 RFC 與實作

要讀 RFC 首先就得先知道去哪裡取得 RFC。我們可以送 e-mail 給 rfc-info@isi.edu，標題用「getting rfcs」，內容只要有一行「ways_to_get_rfcs」這樣就行了[11]。接下來，就會收到回信詳細的說明有哪些方法、位置可以取得 RFC。雖然每一份 RFC 的內容都不大一樣，不過大部份的 RFC 的架構都會有標頭、標題、索引(如果文件很長的話)、摘要介紹、再來就是主要的內容和附錄、索引。

RFC 並不一定要和實作扯上關係，但是大部份的 RFC 討論的都是 Internet 上的技術規格，所以多多少少都會有一、二份對應的成品出現。我們分別截取 RIP(RFC 1058)[12] 和 Requirements For IPv4 Routers (RFC 1812)[13] 二個 RFC 中的部份段落，來對應說明 RFC 文件中的敘述和實作上的關係。這二個 RFC 是構成一個 RIP router 的基本要求，他們的關係如圖七所示。由於現在的 RIPv1 多被 RIPv2 所取代了，所以 RFC 1058 的狀態已經是 Historic；而 RFC 1812 目前的狀態則是 Proposed-Standard。



圖七：RFC 1058 與 RFC 1812

Routing Information Protocol：RFC 1058

這一份 RFC 一開始就說明這份文件是把現存 router 之間所使用的資訊交換協定給明確的定義成文件，而不是嘗試定義一個新的協定。這份文件裡面所說的 RIP 是根據一開始 BSD 4.3 裡所附的「routed」[14]這支程式而來的。這份文件包含了 RIP 使用的通訊埠、訊息封包格式、更新資料及資料過期的時限、概略演算法(Distance Vector)的介紹、daemon 所需紀錄的基本欄位、一些維持 routing table 更新正確性及效率的方式(Split horizon、Split horizon with poison reversed 以及 Triggered Update) 等資訊。

以 Linux 裡使用的 netkit-routed-0.12 (實作 RFC 1058，即 RIPv1)爲例，這個 daemon 主要的工作就只有三件：接收路由更新訊息、發送路由更新訊息以及修改作業系統核心裡的 routing table。往後我們將拿 RFC 與 netkit-routed-0.12 及 Linux Kernel 的程式碼互相對照。

通訊埠與協定訊息

首先來看看通訊埠和通訊協定，RFC 1058 裡有著這一段：

```
3.1. Message formats
RIP is a UDP-based protocol. Each host that uses RIP has a routing process that sends and
receives datagrams on UDP port number 520.
```

所以在程式初始化的時候，在 bind 之前就指定要使用 UDP/520 這個通訊埠。

```
1:   sp = getservbyname("router", "udp");
2:   if (sp == NULL) {
3:        fprintf(stderr, "routed: router/udp: unknown service\n");
4:        exit(1);
5:   }
6:   addr.sin_family = AF_INET;
7:   addr.sin_port = sp->s_port;
```

程式碼列表一：netkit-routed-0.12/routed/main.c/main()

其中，getservbyname() 是從 /etc/services 檔案中取得系統對 router/udp 所定義的相關資訊後再指定給 sockaddr 資料結構。/etc/services 內就有一行定義 routed 所要使用的通訊埠：

```
# name      port/porto      comments
router      520/udp         route routed
```

接下來，routed 就在一個無窮迴圈內利用該 socket 接收及發送訊息。

訊息封包格式

另外，這份 RFC 中也定義了 RIP 的封包如表二所示。

0 1 2 3 4 5 6 7	8 9 0 1 2 3 4 5	6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Command(1)	Version(1)	Must be zero(2)
Address family identifier(2)		Must be zero(2)
IP Address(4)		
Must be zero(4)		
Must be zero(4)		
Metric(4)		
……		

表二：RIP 訊息封包格式

其中 Command 可以是 1 至 5，目前有在用的只有 1(REQUEST) 和 2(RESPONSE)，其他的都淘汰了，而 5 是保留不用的；而 Version 可能是 1 或 2，不過這支 routed 的程式只支援 RIPv1。在程式碼裡面，Command 和 Version 的定義、使用的資料結構、以及時限等都是定義在 /usr/include/protocols/routed.h 檔案中。如程式碼列表二所示。

```
1: #define RIPVERSION              1
2:
3: #define TIMER_RATE              30        /* alarm clocks every 30 seconds */
4: #define SUPPLY_INTERVAL         30        /* time to supply tables */
5: #define EXPIRE_TIME             180       /* time to mark entry invalid */
6: #define GARBAGE_TIME            240       /* time to garbage collect */
7:
8: #define RIPCMD_REQUEST          1         /* want info */
9: #define RIPCMD_RESPONSE         2         /* responding to request */
10: #define RIPCMD_TRACEON         3         /* turn tracing on */
11: #define RIPCMD_TRACEOFF        4         /* turn it off */
12: #define RIPCMD_MAX             5
13:
14: struct netinfo {
15:  struct    sockaddr rip_dst;   /* destination net/host */
16:  int       rip_metric;         /* cost of route */
17: };
18:
19: struct rip {
20:  u_char  rip_cmd;          /* request/response */
21:  u_char  rip_vers;         /* protocol version # */
22:  u_char  rip_res1[2];      /* pad to 32-bit boundary */
23:  union {
24:       struct  netinfo ru_nets[1];    /* variable length... */
25:       char    ru_tracefile[1];       /* ditto ... */
26:  } ripun;
27: #define rip_nets          ripun.ru_nets
28: #define rip_tracefile     ripun.ru_tracefile
29: };
```

程式碼列表二：/usr/include/protocols/routed.h

我們可以看出這些常數和資料結構的定義和 RFC 裡所定義的訊息封包格式完全是相符

合的。所有的 routing daemon 只要是遵照 RFC 所定義的資訊，不管是誰做出來的 routing daemon 應該都是可以順利的互相合作無誤的。

當 routed 從網路上接收到訊息時，會將收到的封包拆解，檢查其版本、命令，然後做適當的回應。比如說收到 1(REQUEST) 時，要送出自己知道的路徑資訊；收到 2(RESPONSE) 時，要適當的更新自己作業系統核心內的路徑資訊。在 RFC 1058 裡面雖然定義 3(TRACEON) 和 4(TRACEOFF) 這兩個命令已經被淘汰了，不過 routed 這支程式裡面還是有實作這一部份。這一點是實作和 RFC 裡有點不同的地方。

路徑計算(Distance Vector Algorithm)及更新

而在更新自己的 routing table 部份，routed 使用的是由 Distance Vector (或稱 Ford-Fulkerson、Bellman-Ford)的演算法，在 RFC 1058 裡對這個演算法的基本定義爲：

$$D(i,i) = 0 \qquad \texttt{all i}$$

$$D(i,j) = \min_k [d(i,k) + D(k,j)] \qquad \texttt{otherwise}$$

其中 D(i, j) 表示由 i 到 j 的最短距離；而 d(i, j) 表示由 i 到 j 二個直接連接點的距離。根據這個演算法，RIP 就可以算出到某個特定地點要走的路徑。此外，這份 RFC 裡也描述了一些防止錯估路徑的方法，比如說 Split horizon 防止將其他 router 送來的路徑訊息回送；Triggered update 當更新自己的 routing table 時讓週遭的 router 可以儘快知道自己最新的狀況。以收到 RESPONSE 的封包爲例，routed 處理的方式如程式碼列表三。

我們在程式碼中適當的位置加上了註解。其中 8 至 11 行爲根據 distance vector 演算法計算新的路徑距離；第 19、20 行判斷週遭的 router 是否回送由自己發出的 routing 訊息，並加以拒收；27 至 30 行判斷新計算的結果是否比現有的資訊更佳，決定是否要更新；以及最後 36 至 40 行根據本身的 routing table 是否被更新而決定是否要通知週遭的 router。

```
1: void rip_input(struct sockaddr *from, struct rip rip, int size) {
2:   ……
3:   switch(rip->rip_cmd) {
4:   ……
5:   case RIPCMD_RESPONSE:
6:        ……
7:        /* Calculate new metric by distance vector: d(i,k)+D(k,j) */
8:        if ((unsigned) n->rip_metric < HOPCNT_INFINITY)
9:             n->rip_metric += ifp->int_metric;
10:       if ((unsigned) n->rip_metric > HOPCNT_INFINITY)
11:            n->rip_metric = HOPCNT_INFINITY;
```

```
12:       rt = rtlookup(&n->rip_dst);
13:       /* If the input routing information comes first time or
14:          it's an internal routing information */
15:       if (rt == 0 || (rt->rt_state & (RTS_INTERNAL|RTS_INTERFACE)) ==
16:            (RTS_INTERNAL|RTS_INTERFACE)) {
17:            /* If we're hearing a logical network route
18:             * back from a peer to which we sent it, ignore it. */
19:            if (rt && rt->rt_state & RTS_SUBNET && (*afp->af_sendroute)(rt, from))
20:                 continue;
21:            ……
22:       }
23:       ……
24:       if (equal(from, &rt->rt_router)) {      /* update from the same router */
25:            ……
26:            /* update metric or timer */
27:       } else if ((unsigned) n->rip_metric < (unsigned)rt->rt_metric ||
28:            (rt->rt_metric == n->rip_metric &&
29:            rt->rt_timer > (EXPIRE_TIME/2) &&
30:            (unsigned) n->rip_metric < HOPCNT_INFINITY)) {
31:            ……
32:            /* update metric and timer */
33:       }
34:       break;
35:  }
36:  if(changes && supplier &&
37:       now.tv_sec - lastfullupdate.tv_sec < SUPPLY_INTERVAL-MAX_WAITTIME) {
38:       /* Triggered update */
39:       ……
40:  }
41: }
```

程式碼列表三：netkit-routed-0.12/routed/input.c/rip_input()

其他的如每隔 30 秒要廣播一次路徑更新資訊、每筆紀錄超過 180 秒就算無效等也都定義在這一份 RFC 裡，而關於時間這一部份 routed 則是註冊一個 ALARM 的 signal，然後每隔 30 秒做一次檢查。至於如何去修改作業系統核心中的 routing table 其實有很多種方法，這一部份並沒有定義在 RFC 1058 之中，在 routed 裡，是利用 ioctl() 系統呼叫直接對 kernel 送出修改 routing table 的命令。

Requirements For IPv4 Routers: RFC 1812

接下來，我們來看另一份 RFC。Requirements For IPv4 Routers (RFC 1812) 這份 RFC 定義一個 IPv4 的 Router 需要做到哪些基本的功能。這份 RFC 的頁數一共有 175 頁之多，而我們要給大家介紹的部份是在 router forwarding 時以作業系統核心以 Classless Inter Domain Routing (CIDR) longest match 爲原則來找尋適當的路由。

Longest Prefix Match

在這一份 RFC 裡與 CIDR 相關的敘述只有一點點而已(在 175 頁中所佔的比例約一頁，算是十分少的)，大意就是說 Internet 成長的速度很快，傳統的 Class A、B、C 三種 network

的分法已經不敷使用了，因此 router 在做 packet forwarding 的時候得使用「Classless」的方式，讓 IP 可以被更充分的利用。

所以在作業系統裡的 routing table 得紀錄的不只是傳統固定長度的 netmask，而是長短不一的 netmask。同時在做 forwarding 的時候，要以 longest match 爲原則，將 netmask 運算後符合最長部份的結果做爲選擇。

在這份 RFC 裡面也對 CIDR 的 netmask 做了適當的定義，netmask 必需是從 MSB 開始放置連續的 1；而從 LSB 開始放置連續的 0，不能有 1 和 0 夾雜的情形發生。也就是說 network-id 部份必需是連續的，原文如下：

Architecturally correct subnet masks are capable of being represented using the prefix length description. They comprise that subset of all possible bits patterns that have

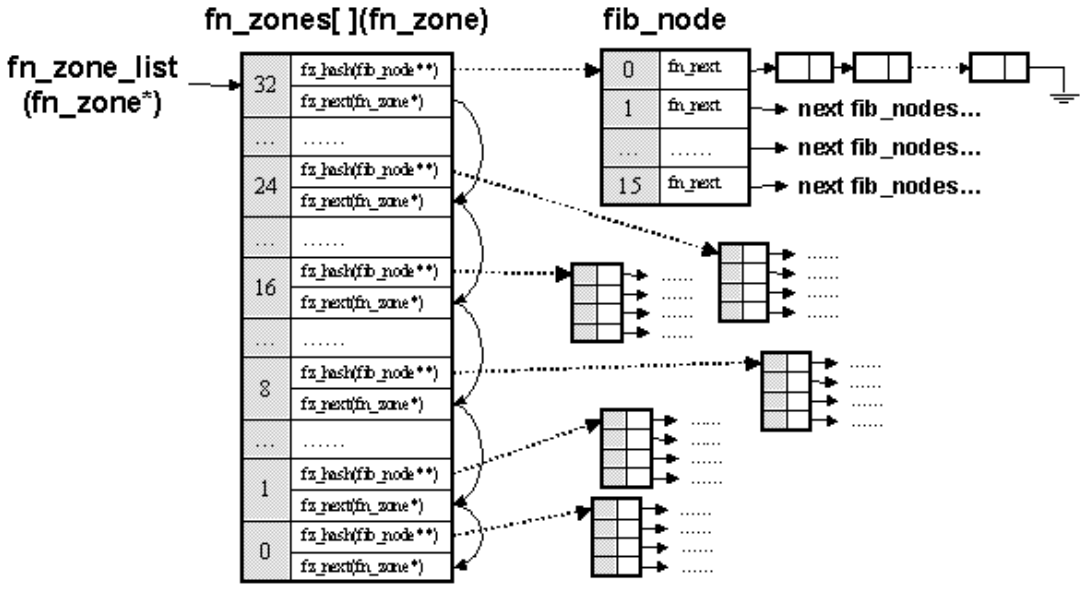
- **a contiguous string of ones at the more significant end,**
- **a contiguous string of zeros at the less significant end, and**
- **no intervening bits.**

Linux 實作

Linux kernel[15] 裡在 routing 的部份也是遵守 RFC 1812 裡所規範定義的原則。在查詢路徑表的時候，依照 longest match 的方法。Linux Kernel 將所有的路徑資訊依照其 netmask 的依其長度的不同 netmask 將路徑資料放在不同的串列中，當需要查表時，就從 netmask 長的串列開始查起，一旦找到符合的項目，那麼這一定就是最正確的答案。

這種實作的方式可以很明顯的受惠於 RFC 裡明確的定義。由於 RFC 1812 裡定義 network-id 一定是連續的，所以 Linux kernel 可以確定所有的 netmask 只有 33 種不同的長度。其資料結構如圖八所示。

fn_zone_list 是一個指向 fn_zone 的指標，他會指向一個有資料且 netmask 最長的 fn_zone，而每個 fn_zone 會再指向下一個 netmask 較自己爲短但最長的 fn_zone，如此串接起來，在搜尋時從 fn_zone_list 開始查詢便是由 netmask 較長的 zone 開始查詢。fn_zones[ ] 被初始化時，每個 fn_zone 會配置一個有 16 (0 至 15) 個位置的 fib_hash 陣列，做爲 fib_node 的 hash table，使得查詢每個 zone 時經過一次 hash 可以有更好的效率。此外，hash table 裡的每個 bucket 都是一個 fib_node 的 link list，裡面的每個 fib_node 在放入 list 的時候是照其 destination 32-bit 的值由大到小存放的，這也是爲了效率的考量而設計的。



圖八：Linux Routing Forwarding Table 之資料結構

有了這樣子的資料結構，在尋找路徑時，只要依序檢查每個串列，找到結果後回傳即可。在程式碼列表四中，我們可以在第 5 行看到依序對不同 netmask 所劃分的區域(zone)做檢查；在比較之前，目的地的 IP(key->dst) 先經過 fz_key() 函式做 IP & netmask 的運算後產生比對的根據 k。而在第 8 行的 fz_chain() 函式內計算 hash 值(hash 值的計算以 XOR、SHIFT 和 AND 求得)取得 fib_node hash table 的 bucket，再逐一呼叫 fn_key_eq() 函式比對該 link list 內每項資料是否符合 routing 的目的地。fn_key_eq() 函式只要單純的將兩個 32-bit 的無號整數用 == 做比較即可判斷是否找到適合的路徑。

```
1: static int fn_hash_lookup
2:   (struct fib_table *tb, const struct rt_key *key, struct fib_result *res) {
3:   struct fn_zone *fz;
4:   struct fn_hash *t = (struct fn_hash*)tb->tb_data;
5:   for (fz = t->fn_zone_list; fz; fz = fz->fz_next)  {     /* For each zone */
6:        struct fib_node *f;
7:        fn_key_t k = fz_key(key->dst, fz);
8:        for (f = fz_chain(k, fz); f; f = f->fn_next)  {    /* For each node */
9:             if (!fn_key_eq(k, f->fn_key)) {               /* not matched */
10:                 if (fn_key_leq(k, f->fn_key))
11:                      break;                          /* break to next zone */
12:                 else
13:                      continue;                       /* continue search this zone */
14:            }
15:            ……
16:            /* matched! fill result and return(0);*/
17:       }
18:  }
19:  return 1;
20: }
```

程式碼列表四：linux-2.2.13/net/ipv4/fib_hash.c/fn_hash_lookup()

其他的作業系統核心可能使用不同的資料結構與方法來達到 longest match 的目的，但是只要是 longest match，大家所表現出來的功能都會是相同的，差別只在於不同的實作有著不同的效率。

看完上面二份 RFC 之後，我們知道 RFC 文件裡面大部份的內容其實都是屬於「規格」的定義，對於實作沒有什麼嚴格的規定或限制。每個實作只要遵照 RFC 裡面所要求的格式及功能來做，就可以互相溝通無誤，這也是 RFC 的目的之一：規範 Internet 上的標準。

五、結論

經過以上的探討後，我們有下列幾點結論：

1. RFC 的產生的過程主要由 IETF 的 WGs 提出，經 IESG 審核後形成。
2. 日常在用的 RFC 協定或是標準在所有 RFC 中所佔的比例並不多(低於 30%)。
3. 更新、淘汰以及單一議題分項討論造就大量的 RFC。
4. RFC 技術文件裡大多是定義「規格」，對於實作部份並沒有嚴格的定義及限制。
5. IETF 上 WGs 常常變動，大多現存的 WGs 是很活躍的。

RFC 是尋覓 Internet 知識的最好地方，不論你是一般的使用者、軟體的開發者或是硬體製造商。加入 IETF WGs，閱讀、討論最新的 Internet 相關議題，是讓你熟悉現今 Internet 技術最直接的方式。此外，若大家都遵守 RFC 的規範來發展在 Internet 上應用的軟硬體，那麼 Internet 上就能真正做到無遠拂屆、世界大同的境界。

六、參考資料

[1] Barry M. Leiner, Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Kleinrock, Daniel C. Lynch, Jon Postel, Larry G. Roberts, Stephen Wolff, “A Brief History of the Internet”, http://www.isoc.org/internet-history/brief.html, ISOC, Aug 2000.

[2] “Active IETF Working Groups”, http://www.ietf.org/html.charters/wg-dir.html, IETF, Aug 2000.

[3] “Request For Comments(RFCs)—Overview”, http://www.rfc-editor.org/overview.html, RFC-Editor, Jan 2000.

[4] S. Bradner, “The Internet Standards Process -- Revision 3”, BCP 9, RFC 2026, Harvard University, Oct 1996.

[5] G. Malkin, Proteon, J.Reynolds, “FYI on FYI”, FYI 1, RFC 1150, ISI, Mar 1990.

[6] Ying-Dar Lin, “Classic Internet Protocols”(course slides), http://speed.cis.nctu.edu.tw/~ydlin/course/cn/part2/classicip.pdf, NCTU, May 1999.

[7] “Search the RFC Database”, http://www.rfc-editor.org/rfcsearch.html, RFC-Editor

[8] “Past Meetings of the IETF”, http://www.ietf.org/meetings/past.meetings.html, IETF.

[9] “Proceedings of The 47th IETF”, http://www.ietf.org/proceedings/99nov/index.html, IETF, Nov 1999.

[10] S. Bradner, “IETF Working Group Guidelines and Procedures”, BCP 25, RFC 2418, Harvard University, Sep 1998.

[11] W. Richard Stevens, “TCP/IP Illustrated, Volume 1—The Protocols”, Addison-Wesley, Feb 1998.

[12] C. Hedrick, “Routing Information Protocol”, RFC 1058, Rutgers University, Jun 1988.

[13] F. Baker, Editor, “Requirements for IP Version 4 Routers”, RFC 1812, Cisco, Jun 1995.

[14] David A. Holland, “netkit-routed-0.12”, Berkeley, Jun 1993.

[15] “Linux Kernel 2.2.13”, http://www.kernel.org/,Oct 1999.