Linux 策略性路由应用及深入分析

策略性路由

策略性是指对于 IP 包的路由是以网络管理员根据需要定下的一些策略为主要依据进行路由的。例如我们可以有这样的策略：“所有来直自网 A 的包，选择 X 路径；其他选择 Y 路径”，或者是“所有 TOS 为 A 的包选择路径 F；其他选者路径 K”。

Cisco 的网络操作系统（Cisco IOS）从 11.0 开始就采用新的策略性路由机制。而 Linux 是在内核 2.1 开始采用策略性路由机制的。策略性路由机制与传统的路由算法相比主要是引入了多路由表以及规则的概念。

多路由表（multiple Routing Tables）

传统的路由算法是仅使用一张路由表的。但是在有些情形底下，我们是需要使用多路由表的。例如一个子网通过一个路由器与外界相连，路由器与外界有两条线路 相连，其中一条的速度比较快，一条的速度比较慢。对于子网内的大多数用户来说对速度并没有特殊的要求，所以可以让他们用比较慢的路由；但是子网内有一些特 殊的用户却是对速度的要求比较苛刻，所以他们需要使用速度比较快的路由。如果使用一张路由表上述要求是无法实现的，而如果根据源地址或其它参数，对不同的 用户使用不同的路由表，这样就可以大大提高路由器的性能。

规则（rule）

规则是策略性的关键性的新的概念。我们可以用自然语言这样描述规则，例如我门可以指定这样的规则：

规则一：“所有来自 192.16.152.24 的 IP 包，使用路由表 10， 本规则的优先级别是 1500”

规则二：“所有的包，使用路由表 253，本规则的优先级别是 32767”

我们可以看到，规则包含 3 个要素：

什么样的包，将应用本规则（所谓的 SELECTOR，可能是 filter 更能反映其作用）；

符合本规则的包将对其采取什么动作（ACTION），例如用那个表；

本规则的优先级别。优先级别越高的规则越先匹配(数值越小优先级别越高)。

策略性路由的配置方法

传统的 linux 下配置路由的工具是 route，而实现策略性路由配置的工具是 iproute2 工具包。这个软件包是由 Alexey Kuznetsov 开发的，软件包所在的主要网址为 ftp://ftp.inr.ac.ru/ip-routing/。
这里简单介绍策略性路由的配置方法，以便能更好理解第二部分的内容。详细的使用方法请参考 Alexey Kuznetsov 写的 ip-cfref 文档。策略性路由的配置主要包括接口地址的配置、路由的配置、规则的配置。

接口地址的配置 IP Addr

对于接口的配置可以用下面的命令进行：

```
Usage: ip addr [ add | del ] IFADDR dev STRING
```

例如：

```
router>#  ip addr add 192.168.0.1/24 broadcast 192.168.0.255 label eth0
dev eth0
```

上面表示，给接口 eth0 赋予地址 192.168.0.1 掩码是 255.255.255.0(24 代表掩码中 1 的个数)，广播地址是 192.168.0.255

路由的配置 IP Route

Linux 最多可以支持 255 张路由表，其中有 3 张表是内置的：

表 255 本地路由表（Local table） 本地接口地址，广播地址，已及 NAT 地址都放在这个表。该路由表由系统自动维护，管理员不能直接修改。

表 254 主路由表（Main table） 如果没有指明路由所属的表，所有的路由都默认都放在这个表里，一般来说，旧的路由工具（如 route）所添加的路由都会加到这个表。一般是普通的路由。

表 253 默认路由表（Default table）一般来说默认的路由都放在这张表，但是如果特别指明放的也可以是所有的网关路由。

表 0 保留

路由配置命令的格式如下：

```
Usage: ip route list SELECTOR
ip route { change | del | add | append | replace | monitor } ROUTE
```

如果想查看路由表的内容，可以通过命令：

```
ip route list table table_number
```

对于路由的操作包括 change、del、add 、append 、replace 、 monitor 这些。例如添加路由可以用：

```
router># ip route add 0/0 via 192.168.0.4 table main
router># ip route add 192.168.3.0/24 via 192.168.0.3 table 1
```

第一条命令是向主路由表（main table）即表 254 添加一条路由，路由的内容是设置 192.168.0.4 成为网关。

第二条命令代表向路由表 1 添加一条路由，子网 192.168.3.0（子网掩码是 255.255.255.0）的网关是 192.168.0.3。

在多路由表的路由体系里，所有的路由的操作，例如网路由表添加路由，或者在路由表里寻找特定的路由，需要指明要操作的路由表，所有没有指明路由表，默认是对主路由表（表 254）进行操作。而在单表体系里，路由的操作是不用指明路由表的。

规则的配置 IP Rule

在 Linux 里，总共可以定义 个优先级的规则，一个优先级别只能有一条规则，即理论上总共可以有 条规则。其中有 3 个规则是默认的。命令用法如下：

```
Usage: ip rule [ list | add | del ] SELECTOR ACTION
SELECTOR := [ from PREFIX ] [ to PREFIX ] [ tos TOS ]
[ dev STRING ] [ pref NUMBER ]
ACTION := [ table TABLE_ID ] [ nat ADDRESS ]
[ prohibit | reject | unreachable ]
[ flowid CLASSID ]
TABLE_ID := [ local | main | default | new | NUMBER
```

首先我们可以看看路由表默认的所有规则：

```
root@netmonster# ip rule list
0: from all lookup local
32766: from all lookup main
32767: from all lookup default
```

规则 0，它是优先级别最高的规则，规则规定，所有的包，都必须首先使用 local 表（254）进行路由。本规则不能被更改和删除。

规则 32766，规定所有的包，使用表 main 进行路由。本规则可以被更改和删除。

规则 32767，规定所有的包，使用表 default 进行路由。本规则可以被更改和删除。

在默认情况下进行路由时，首先会根据规则 0 在本地路由表里寻找路由，如果目的地址是本网络，或是广播地址的话，在这里就可以找到合适的路由；如果路由失 败，就会匹配下一个不空的规则，在这里只有 32766 规则，在这里将会在主路由表里寻找路由;如果失败,就会匹配 32767 规则,即寻找默认路由表。如果 失败，路由将失败。重这里可以看出，策略性路由是往前兼容的。

还可以添加规则：

```
router># ip rule add [from 0/0] table 1 pref 32800
router >#ip rule add from 192.168.3.112/32 [tos 0x10] table  2  pref 1500
prohibit
```

第一条命令将向规则链增加一条规则，规则匹配的对象是所有的数据包，动作是选用路由表 1 的路由，这条规则的优先级是 32800。

第二条命令将向规则链增加一条规则，规则匹配的对象是 IP 为 192.168.3.112，tos 等于 0x10 的包，使用路由表 2，这条规则的优先级是 1500，动作是。添加以后，我们可以看看系统规则的变化。

```
router># ip rule
0: from all lookup local
1500 from 192.168.3.112/32 [tos 0x10] lookup 2
32766: from all lookup main
32767: from all lookup default
32800: from all lookup 1
```

上面的规则是以源地址为关键字，作为是否匹配的依据的。除了源地址外，还可以用以下的信息：

From -- 源地址

To -- 目的地址（这里是选择规则时使用，查找路由表时也使用）

Tos -- IP 包头的 TOS（type of sevice）域

Dev -- 物理接口

Fwmark -- 防火墙参数

采取的动作除了指定表，还可以指定下面的动作：

Table 指明所使用的表

Nat 透明网关

Action prohibit 丢弃该包，并发送 COMM.ADM.PROHIITED 的 ICMP 信息

Reject 单纯丢弃该包

Unreachable 丢弃该包， 并发送 NET UNREACHABLE 的 ICMP 信息

策略性路由的应用

基于源地址选路（ Source-Sensitive Routing）

如果一个网络通过两条线路接入互联网，一条是比较快的 ADSL，另外一条是比较慢的普通的调制解调器。这样的话，网络管理员既可以提供无差别的路由服务，也可以根据源地址的不同，使一些特定的地址使用较快的线路，而普通用户则使用较慢的线路，即基于源址的选路。

根据服务级别选路（Quality of Service）

网络管理员可以根据 IP 报头的服务级别域，对于不同的服务要求可以分别对待对于传送速率、吞吐量以及可靠性的有不同要求的数据报根据网络的状况进行不同的路由。

节省费用的应用

网络管理员可以根据通信的状况，让一些比较大的阵发性通信使用一些带宽比较高但是比较贵的路径一段短的时间，然后让基本的通信继续使用原来比较便宜的基 本线路。例如，管理员知道，某一台主机与一个特定的地址通信通常是伴随着大量的阵发性通信的，那么网络管理员可以安排一些策略，使得这些主机使用特别的路 由，这些路由是按需拨号，带宽比较高的线路，通信完成以后就停止使用，而普通的通信则不受影响。这样既提高网络的性能，又能节省费用。

负载平衡（Load Sharing）

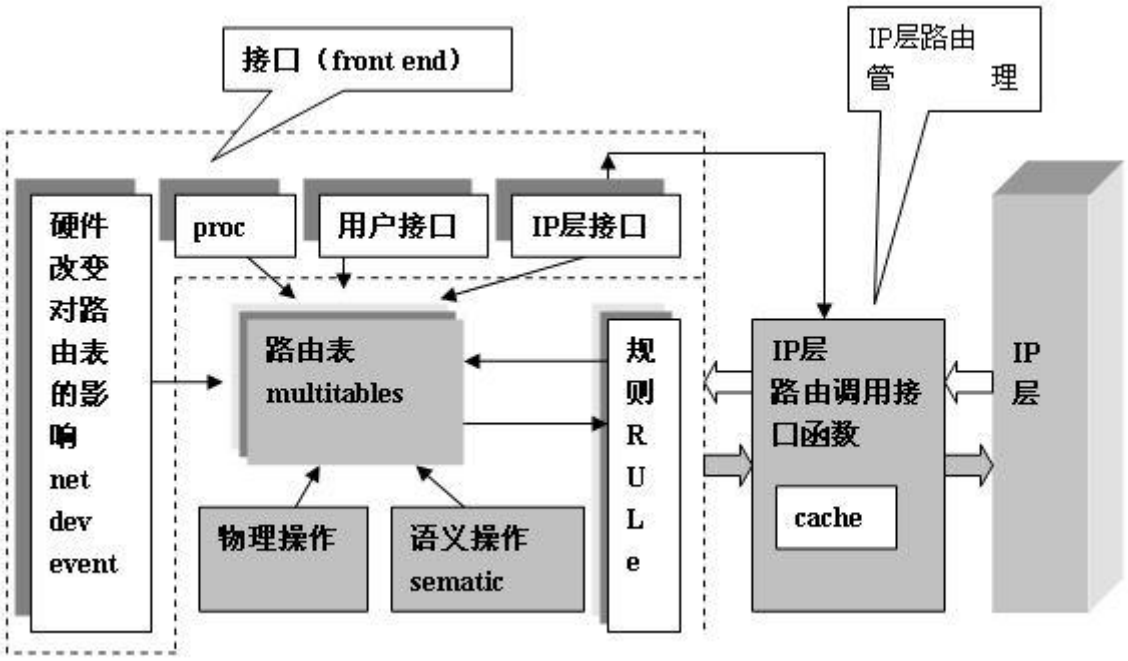
根据网络交通的特征，网络管理员可以在不同的路径之间分配负荷实现负载平衡。

Linux 下策略性路由的实现--RPDB（Routing Policy DataBase）

在 Linux 下，策略性路由是由 RPDB 实现的。对于 RPDB 的内部机制的理解，可以加深对于策略性路由使用的理解。这里分析的是 linux 2.4.18 的 RPDB 实现的细节。主要的实现文件包括：

fib_hash.c
fib_rules.c
fib_sematic
fib_frontend.c
route.c

RDPB 主要由多路由表和规则组成。路由表以及对其的操作和其对外的接口是整个 RPDB 的核心部分。路由表主要由 table，zone，node 这些主要的数据结构构成。对路由表的操作主要包含物理的操作以及语义的操作。路由表除了向 IP 层提供路由寻找的接口以外还 必须与几个元素提供接口：与用户的接口（即更改路由）、proc 的接口、IP 层控制接口、以及和硬件的接口（网络接口的改变会导致路由表内容的改变）。处 在 RDPB 的中心的规则，由规则选取表。IP 层并不直接使用路由表，而是通过一个路由适配层，路由适配层提供为 IP 层提供高性能的路由服务。



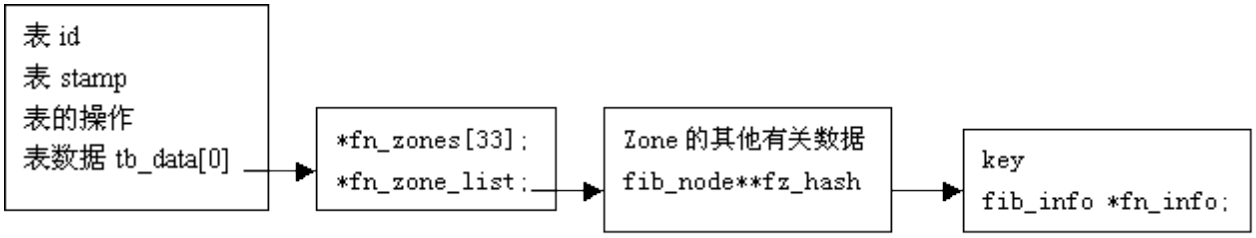
路由表（Fib Table）

数据结构：

在整个策略性路由的框架里，路由表是最重要的的数据结构，我们在上面以及对路由表的概念和结构进行了清楚的说明。Linux 里通过下面这些主要的数据结构进行实现的。

主要的数据结构	作用	位置
struct fib_table	路由表	ip_fib.h 116
struct fn_hash	路由表的哈希数据	fib_hash.c 104
struct fn_zone	zone 域	fib_hash.c 85
struct fib_node	路由节点	fib_hash.c 68
struct fib_info	路由信息	ip_fib.h 57
struct fib_result	路由结果	ip_fib.h 86

数据结构之间的主要关系如下。路由表由路由表号以及路由表的操作函数指针还有表数据组成。这里需要注意的是，路由表结构里并不直接定义 zone 域，而是通过一个数据指针指向 fn_hash。只有当 zone 里有数据才会连接到 fn_zone_list 里。



系统的所有的路由表由数组变量*fib_tables[RT_TABLE_MAX+1]维护，其中系统定义 RT_TABLE_MAX 为 254， 也就是说系统最大的路由表为 255 张，所有的路由表的操作都是对这个数组进行的。。同时系统还定义了三长路由表 *local_table; *main_table。

路由表的操作：

Linux 策略路由代码的主要部分是对路由表的操作。对 于路由表的操作，物理操作是直观的和易于理解的。对于表的操作不外乎就是添加、删除、更新等的操作。还有一种操作，是所谓的语义操作，语义操作主要是指诸 如计算下一条的地址，把节点转换为路由项，寻找指定信息的路由等。

1、物理操作(operation)：

路由表的物理操作主要包括如下这些函数：

路由标操作	实现函数	位置
新建路由表		
删除路由表		
搜索路由	fn_hash_lookup	fib_hash.c 269
插入路由到路由表	fn_hash_insert	fib_hash.c 341
删除路由表的路由	fn_hash_delete	fib_hash.c 433

	fn_hash_dump	fib_hash.c 614
更新路由表的路由	fn_hash_flush	fib_hash.c 729
显示路由表的路由信息	fn_hash_get_info	fib_hash.c 750
选择默认路由	fn_hash_select_default	fib_hash.c 842

2、语义操作(semantics operation)：

语义操作并不涉及路由表整体框架的理解，而且，函数名也是不言自明的，所以请大家参考 fib_semantics.c。

3、接口(front end)

对于路由表接口的理解，关键在于理解那里有

IP

首先是路由表于 IP 层的接口。路由在目前 linux 的意义上来说，最主要的还是 IP 层的路由，所以和 IP 层的的接口是最主要的接口。和 ip 层的衔接主要是向 IP 层提供寻找路由、路由控制、寻找指定 ip 的接口。

Fil_lookup
ip_rt_ioctl fib_frontend.c 286;″ f
ip_dev_find 145

Inet

路由表还必须提供配置接口，即用户直接操作路由的接口，例如增加和删除一条路由。当然在策略性路由里，还有规则的添加和删除。

inet_rtm_delroute 351
inet_rtm_newroute 366
inet_check_attr 335

proc

在/proc/net/route 里显示路由信息。
fib_get_procinfo

4、网络设备（net dev event）

路由是和硬件关联的，当网络设备启动或关闭的时候，必须通知路由表的管理程序，更新路由表的信息。

fib_disable_ip 567
fib_inetaddr_event 575
fib_netdev_event

5、内部维护（ magic）

上面我们提到，本地路由表（local table）的维护是由系统自动进行的。也就是说当用户为硬件设置 IP 地址等的时候，系统自动在本地路由表里添加本地接口地址以及广播地址。

fib_magic 417
fib_add_ifaddr 459
fib_del_ifaddr 498

Rule

1、数据结构

规则在 fib_rules.c 的 52 行里定义为 struct fib_rule。而 RPDB 里所有的路由是保存在 101 行的变量 fib_rules 里的，注意这个变量很关键，它掌管着所有的规则，规则的添加和删除都是对这个变量进行的。

2、系统定义规则：

fib_rules 被定义以后被赋予了三条默认的规则：默认规则，本地规则以及主规则。

u 本地规则 local_rule
94 static struct fib_rule local_rule = {
r_next: &main_rule, /*下一条规则是主规则*/
r_clntref: ATOMIC_INIT(2),
r_table: RT_TABLE_LOCAL, /*指向本地路由表*/
r_action: RTN_UNICAST, /*动作是返回路由*/
};

u 主规则 main_rule
86 static struct fib_rule main_rule = {
r_next: &default_rule,/*下一条规则是默认规则*/
r_clntref: ATOMIC_INIT(2),
r_preference: 0x7FFE, /*默认规则的优先级 32766*/
r_table: RT_TABLE_MAIN, /*指向主路由表*/
r_action: RTN_UNICAST, /*动作是返回路由*/
};
u 默认规则 default rule

```
79 static struct fib_rule default_rule = {
r_clntref: ATOMIC_INIT(2),
r_preference: 0x7FFF,/*默认规则的优先级 32767*/
r_table: RT_TABLE_DEFAULT,/*指默认路由表*/
r_action: RTN_UNICAST,/*动作是返回路由*/
};
```

规则链的链头指向本地规则。

RPDB 的中心函数 fib_lookup

现在到了讨论 RPDB 的实现的的中心函数 fib_lookup 了。RPDB 通过提供接口函数 fib_lookup，作为寻找路由的入口点，在这里有必要详细讨论这个函数，下面是源代码：，

```
310 int fib_lookup(const struct rt_key *key, struct fib_result *res)
311 {
312 int err;
313 struct fib_rule *r, *policy;
314 struct fib_table *tb;
315
316 u32 daddr = key->dst;
317 u32 saddr = key->src;
318
321 read_lock(&fib_rules_lock);
322 for (r = fib_rules; r; r=r->r_next) {/*扫描规则链 fib_rules 里的每一条规则直到匹配为止*/
323 if (((saddr^r->r_src) & r->r_srcmask) ||
324 ((daddr^r->r_dst) & r->r_dstmask) ||
325 #ifdef CONFIG_IP_ROUTE_TOS
326 (r->r_tos && r->r_tos != key->tos) ||
327 #endif
328 #ifdef CONFIG_IP_ROUTE_FWMARK
329 (r->r_fwmark && r->r_fwmark != key->fwmark) ||
330 #endif
331 (r->r_ifindex && r->r_ifindex != key->iif))
332 continue;/*以上为判断规则是否匹配，如果不匹配则扫描下一条规则，否则继续*/

335 switch (r->r_action) {/*好了，开始处理动作了*/
336 case RTN_UNICAST:/*没有设置动作*/
337 case RTN_NAT: /*动作 nat ADDRESS*/
338 policy = r;
339 break;
```

```
340 case RTN_UNREACHABLE: /*动作 unreachable*/
341 read_unlock(&fib_rules_lock);
342 return -ENETUNREACH;
343 default:
344 case RTN_BLACKHOLE:/* 动作 reject */
345 read_unlock(&fib_rules_lock);
346 return -EINVAL;
347 case RTN_PROHIBIT:/* 动作 prohibit */
348 read_unlock(&fib_rules_lock);
349 return -EACCES;
350 }
351 /*选择路由表*/
352 if ((tb = fib_get_table(r->r_table)) == NULL)
353 continue;
/*在路由表里寻找指定的路由*/
354 err = tb->tb_lookup(tb, key, res);
355 if (err == 0) {/*命中目标*/
356 res->r = policy;
357 if (policy)
358 atomic_inc(&policy->r_clntref);
359 read_unlock(&fib_rules_lock);
360 return 0;
361 }
362 if (err < 0 && err != -EAGAIN) {/*路由失败*/
363 read_unlock(&fib_rules_lock);
364 return err;
365 }
366 }
368 read_unlock(&fib_rules_lock);
369 return -ENETUNREACH;
370 }
```

上面的这段代码的思路是非常的清晰的。首先程序从优先级高到低扫描所有的规则，如果规则匹配，处理该规则的动作。如果是普通的路由寻址或者是 nat 地址转换的换，首先从规则得到路由表，然后对该路由表进行操作。这样 RPDB 终于清晰的显现出来了。

IP 层路由适配（IP route）

路由表以及规则组成的系统，可以完成路由的管理以及查找的工作，但是为了使得 IP 层的路由工作更加的高效，linux 的路由体系里，route.c 里完成大多数 IP 层与 RPDB 的适配工作，以及路由缓冲（route cache）的功能。

调用接口

IP 层的路由接口分为发送路由接口以及接收路由接口：

发送路由接口

IP 层在发送数据时如果需要进行路由工作的时候，就会调用 ip_route_out 函数。这个函数在完成一些键值的简单转换以后，就会调用 ip_route_output_key 函数，这个函数首先在缓存里寻找路由，如果失败就会调用 ip_route_output_slow，ip_route_output_slow 里调用 fib_lookup 在路由表里寻找路由，如果命中，首先在 缓存里添加这个路由，然后返回结果。

ip_route_out route.h
ip_route_output_key route.c 1984;
ip_route_output_slow route.c 1690;″

接收路由接口

IP 层接到一个数据包以后，如果需要进行路由，就调用函数 ip_route_input，ip_route_input 现在缓存里寻找，如果失败则 ip_route_inpu 调用 ip_route_input_slow， ip_route_input_slow 里调用 fib_lookup 在路由表里寻找路由，如果命中，首先在缓存里添加这个路由，然后返回结果。

ip_route_input_slow route.c 1312;″ f
ip_route_input route.c 1622;″ f

cache

路由缓存保存的是最近使用的路由。当 IP 在路由表进行路由以后，如果命中就会在路由缓存里增加该路由。同时系统还会定时检查路由缓存里的项目是否失效，如果失效则清除。