Linux 内核源码剖析

—— TCP/IP实现

下 册

樊东东 莫 澜 编著

机械工业出版社
CHINA MACHINE PRESS

ISBN 978-7-111-32373-0

◎策划 车忱 ◎封面设计 旭洲企划 刘吉维

Linux内核源码剖析
——TCP/IP实现
下 册

本书详细论述了Linux内核2.6.20版本中TCP/IP的实现。书中给出了大量的源代码，通过对源代码的详细注释，帮助读者掌握TCP/IP的实现。本书根据协议栈层次，从驱动层逐步论述到传输层，包括驱动的实现、接口层的输入输出、IP层的输入输出以及IP选项的处理、邻居子系统、路由、套接口及传输层等内容，全书基本涵盖了网络体系架构全部的知识点。特别是TCP，包括TCP连接的建立和终止、输入与输出，以及拥塞控制的实现。

- 套接口缓存
- 网络设备
- IP：网际协议
- ICMP：Internet控制报文协议
- IP组播
- IGMP：Internet组管理协议
- 邻居子系统
- 路由表
- 套接口层
- TCP：传输控制协议
- TCP连接的建立
- TCP拥塞控制的实现
- TCP的输出
- TCP的输入
- TCP连接的终止
- UDP：用户数据报

上架建议：操作系统/Linux

ISBN 978-7-111-32373-0

地址：北京市百万庄大街22号 邮政编码：100037
电话服务 网络服务
社服务中心：(010)88361066 门户网：http://www.cmpbook.com
销售一部：(010)68326294 教材网：http://www.cmpedu.com
销售二部：(010)88379649
读者服务部：(010)68993821
封面无防伪标均为盗版

定价：142.00元(上、下册)

Linux 内核源码剖析

——TCP/IP 实现

下册

樊东东　莫　澜　编著

机械工业出版社

下册目录

第20章 路由缓存

路由缓存用于提高路由查找的命中率，减少路由表查找的时间。缓存的主要工作是存储使路由子系统能够找到报文目的地的信息，并通过一组函数向更高层提供该信息。路由缓存还提供了一些函数用于缓存的清理。路由缓存存储的信息为可应用于所有三层协议的路由缓存项，所以可以包含表示路由缓存项的任意数据结构。

路由缓存项的管理及查找会涉及以下文件：

- include/net/route.h，定义目的路由缓存项的部分结构、宏、函数等。
- include/net/flow.h，定义查询路由缓存的条件组合结构、宏和函数原型等。
- include/net/dst.h，定义目的路由缓存项的部分结构、宏、函数等。
- net/ipv4/route.c，实现路由缓存项的操作函数。

20.1 系统参数

系统参数如下：

（1）flush，控制路由缓存的刷新。当对参数进行写操作时，并不是简单地调整参数，而是当用户将 n 写入该文件时，会触发一个动作，ipv4_sysctl_rtcache_flush()被调用。将在 n 秒后对路由表缓存进行一次刷新。当写入一个负值到 flush 中时，内核调度在默认延迟 min_delay 秒之后进行一次刷新。

（2）gc_elasticity，在添加路由缓存项到缓存时，当路由缓存散列表桶链表长度超过此值时，便会释放最老的缓存项，参见 20.5 节。

（3）gc_min_interval 与 gc_min_interval_ms，用于控制路由缓存垃圾回收的频率和行为，gc_min_interval_ms 标识路由表垃圾回收的最小间隔，不得小于 0.5s。

gc_min_interval 已废弃，被 gc_min_interval_ms 取代。

（4）gc_thresh，用来控制路由缓存垃圾回收机制的频率和行为。当缓存中的路由条数超过此值时，开始回收垃圾。

（5）max_delay

（6）min_delay，分别是在用户调度进行刷新和内核实际刷新缓存之间的最短和最长时间间隔，min_delay 默认为 2s，而 max_delay 默认为 10s。

（7）max_size，路由缓存数量的最大值，该值在初始化路由模块时被设置。当缓存数量达到该值后，老的路由表项会被清除。

（8）min_adv_mss，用于初始化路由表项的度量值中的 MSS 值，取设备的 MTU 和 min_adv_mss 之间的较大值。

（9）mtu_expires，PMTU 缓存在路由缓存项度量值中的时间，超过该值后会过期，参见 20.8.1 节。默认值为 600s。

（10）min_pmtu，路径 MTU 发现协议可以为路由表项设置的最小 PMTU 值。

（11）secret_interval，定时刷新路由缓存的周期，每隔 secret_interval 秒会刷新一次，参见

20.9 节。

20.2 路由缓存的组织结构

如图 20-1 所示，路由缓存是用一张散列表来实现的，路由缓存散列表的类型是 rt_hash_bucket 结构，该结构只包含指向缓存元素链表的一个指针。缓存项的类型为 rtable 结构。

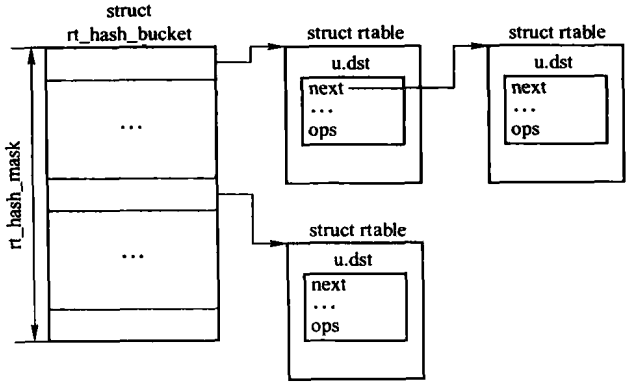

图 20-1 路由缓存结构

dst_entry 结构包含了缓存与邻居层的接口、transformers（诸如 Ipsec）以及路由缓存管理。

20.2.1 rtable 结构

IPv4 使用 rtable 结构来存储缓存内的路由表项。可以通过查看/proc/net/rt_cache 文件，或者通过 ip route list cache 和 route -C 命令来列出路由缓存的内容。

```
51 struct rtable
52 {
53    union
54    {
55       struct dst_entry   dst;
56       struct rtable      *rt_next;
57    } u;
58
59    struct in_device   *idev;
60
61    unsigned       rt_flags;
62    __u16          rt_type;
63    __u16          rt_multipath_alg;
64
65    __be32          rt_dst;    /* Path destination    */
66    __be32          rt_src;    /* Path source       */
67    int          rt_iif;
68
69    /* Info on neighbour */
70    __be32          rt_gateway;
71
72    /* Cache lookup keys */
73    struct flowi       fl;
```

```
74
75     /* Miscellaneous cached information */
76     __be32          rt_spec_dst; /* RFC1122 specific destination */
77     struct inet_peer    *peer; /* long-living peer info */
78 };
```

53 union {...} u

dst_entry 结构作为一部分嵌入到 rtable 结构中，而 dst_entry 结构中的第一个成员 next 就是用于链接分布在同一个散列桶内的 rtable 实例，为了便于访问 next，因此将 dst 和 rt_next 联合起来。虽然指针的名称不同，但它们所指向的内存位置是相同的。

59 struct in_device *idev

指向输出网络设备的 IPv4 协议族中的 IP 配置块。注意：对送往本地的输入报文的路由，输出网络设备设置为回环设备。

61 unsigned rt_flags

用于标识路由表项的一些特性和标志，见表 20-1。

表 20-1 路由表项的标志

rt_flags	描述
RTCF_NOTIFY	路由表项的所有变化通过 netlink 通知给感兴趣的用户空间应用程序，该选项还没有完全实现。利用诸如 ip route 等命令来设置该标志
RTCF_REDIRECTED	由接收到的 ICMP_REDIRECT 消息作出响应而添加一条路由缓存项，参见 20.10 节
RTCF_DOREDIRECT	表示并不是最优路由。ip_forward()依据该标志和其他信息，决定是否需要发送 ICMP 重定向消息。例如，如果报文是基于源站的路由，就不应当生成 ICMP 重定向消息
RTCF_DIRECTSRC	不正确的源地址。ICMP 模块不会对来自此源地址的地址掩码请求消息作出回应。每当调用 fib_validate_source()检查到接收报文的源地址通过一个本地作用范围（RT_SCOPE_HOST）的下一跳是可达时，就设置该标志
RTCF_SNAT RTCF_DNAT RTCF_NAT	已废除
RTCF_BROADCAST	路由的目的地址是一个广播地址
RTCF_MULTICAST	路由的目的地址是一个多播地址
RTCF_LOCAL	路由的目的地址是一个本地地址（即本地接口上配置的某个地址）。对本地广播地址和本地多播地址也设置该标志，参见 ip_route_input_slow()和 ip_route_input_mc()
RTCF_REJECT	未被使用。依据 IPROUTE2 软件包的 ip rule 命令的语法，在该命令中有一个关键字 reject，但该关键字还未被接受
RTCF_TPROXY	未使用
RTCF_DIRECTDST	未使用
RTCF_FAST	已废除
RTCF_MASQ	IPv4 已不使用

62 __u16 rt_type

路由表项的类型，见表 20-2。它间接定义了当路由查找匹配时应采取的动作。

表 20-2 路由类型

rt_type	描述
RTN_UNSPEC	定义一个未初始化的值。例如，当从路由表中删除一个表项时使用该值，这是因为删除操作不需要指定路由表项的类型

（续）

rt_type	描述
RTN_LOCAL	目的地址被配置为一个本地接口的地址
RTN_UNICAST	该路由是一条到单播地址的直连或非直连（通过一个网关）路由。当用户通过 ip route 命令添加路由但没有指定其他路由类型时，路由类型默认设置为 RTN_UNICAST
RTN_MULTICAST	目的地址是一个多播地址
RTN_BROADCAST	目的地址是一个广播地址。匹配的 ingress 报文以广播方式送往本地，匹配的 egress 报文以广播方式发送出去
RTN_ANYCAST	匹配的输入报文以广播方式送往本地，匹配的输出报文以单播发送出去。IPv4 没有该类型
RTN_BLACKHOLE RTN_UNREACHABLE RTN_PROHIBIT RTN_THROW	这些值与特定的管理配置而不是与目的地址类型相关联
RTN_NAT	已废弃
RTN_XRESOLVE	有一个外部解析器来处理该路由，目前尚未实现该功能

```
63 __u16 rt_multipath_alg
```

标识多路径缓存算法，在创建路由表项时根据相关路由项的配置来设置。

```
65 __be32 rt_dst
66 __be32 rt_src
```

目的 IP 地址和源 IP 地址。

```
67 int rt_iif
```

输入网络设备标识，从输入网络设备的 net_device 数据结构中得到。对本地生成的流量（因此不是从任何接口上接收到的），该字段被设置为出设备的 ifindex 字段。对本地生成的报文，fl 中的 iif 字段被设置为 0。

```
70 __be32 rt_gateway
```

当目的主机为直连时，即在同一链路上，rt_gateway 表示目的地址。当需要通过一个网关到达目的地时，rt_gateway 被设置为路由项中的下一跳的网关。

```
73 struct flowi fl
```

用于缓存查找的搜索的条件组合，参见 20.2.2 节。

```
76 __be32 rt_spec_dst
```

首选源地址。

添加到路由缓存内的路由缓存项是单向的。但是在一些情况下，接收到报文可能触发一个动作，要求本地主机选择一个源 IP 地址，以便在向发送方回送报文时使用。这个地址，即首选源 IP 地址，必须与路由该输入报文的路由缓存项保存在一起。首选源 IP 地址被保存在 rt_spec_dst 字段内，下面是使用该地址的两种情况：

1）当一个主机接收到一个 ICMP 回显请求消息时（常用的 ping 命令），如果主机没有明确配置为不作出回应，则该主机返回一个 ICMP 回显应答消息。对该输入 ICMP 回显请求消息选择路由，路由项的 rt_spec_dst 被用作路由 ICMP 回显请求消息而进行路由查找的源地址。参见 14.6.2 节的 icmp_reply()和 11.11.2 节的 ip_send_reply()。

2）记录路由 IP 选项和时间戳 IP 选项要求途经主机的 IP 地址记录到选项中。

```
77 struct inet_peer *peer
```

指向与目的地址相关的对端信息块。

20.2.2 flowi 结构

利用 flowi 数据结构，就可以根据诸如输入网络设备和输出网络设备、三层和四层协议报头中的参数等字段的组合对流量进行分类。它通常被用作路由查找的搜索条件组合，IPsec 策略的流量选择器以及其他高级用途。

```
13 struct flowi {
14     int    oif;
15     int    iif;
16     __u32    mark;
17
18     union {
19         struct {
20             __be32          daddr;
21             __be32          saddr;
22             __u8          tos;
23             __u8          scope;
24         } ip4_u;
25
26         struct {
27             struct in6_addr       daddr;
28             struct in6_addr       saddr;
29             __be32          flowlabel;
30         } ip6_u;
31
32         struct {
33             __le16          daddr;
34             __le16          saddr;
35             __u8          scope;
36         } dn_u;
37     } nl_u;
38 #define fld_dst       nl_u.dn_u.daddr
39 #define fld_src       nl_u.dn_u.saddr
40 #define fld_scope    nl_u.dn_u.scope
41 #define fl6_dst       nl_u.ip6_u.daddr
42 #define fl6_src       nl_u.ip6_u.saddr
43 #define fl6_flowlabel    nl_u.ip6_u.flowlabel
44 #define fl4_dst       nl_u.ip4_u.daddr
45 #define fl4_src       nl_u.ip4_u.saddr
46 #define fl4_tos       nl_u.ip4_u.tos
47 #define fl4_scope    nl_u.ip4_u.scope
48
49     __u8    proto;
50     __u8    flags;
51 #define FLOWI_FLAG_MULTIPATHOLDROUTE 0x01
52     union {
53         struct {
54             __be16    sport;
55             __be16    dport;
56         } ports;
57
58         struct {
```

```
59          __u8    type;
60          __u8    code;
61       } icmpt;
62
63       struct {
64          __le16   sport;
65          __le16   dport;
66       } dnports;
67
68       __be32      spi;
69
70 #ifdef CONFIG_IPV6_MIP6
71       struct {
72          __u8    type;
73       } mht;
74 #endif
75    } uli_u;
76 #define fl_ip_sport    uli_u.ports.sport
77 #define fl_ip_dport    uli_u.ports.dport
78 #define fl_icmp_type    uli_u.icmpt.type
79 #define fl_icmp_code    uli_u.icmpt.code
80 #define fl_ipsec_spi    uli_u.spi
81 #ifdef CONFIG_IPV6_MIP6
82 #define fl_mh_type    uli_u.mht.type
83 #endif
84    __u32         secid;    /* used by xfrm; see secid.txt */
85 } __attribute__((__aligned__(BITS_PER_LONG/8)));
```

```
14 int oif
15 int iif
```

输出网络设备索引和输入网络设备索引。

```
18 union {...} nl_u
```

该联合对应三层，目前支持的协议为 IPv4、IPv6 和 DECnet。

这里应该说明一下 RTO_ONLINK 标志，该标志通过 TOS 变量传递，但该变量与 IP 包头中的 TOS 域无关。此处只使用 TOS 域的一个无用的比特位。当该标志被设置时，表示目的地位于本地子网，所以无须路由查找。

```
49 __u8 proto
```

标识四层协议。

```
50 __u8 flags
```

该变量只定义了一个标志，FLOWI_FLAG_MULTIPATHOLDROUTE，它最初用于多路径代码，但现已废弃不再使用。

```
52 union {...} uli_u
```

该联合对应四层，目前支持的协议为 TCP、UDP、ICMP、DECnet 和 IPsec。

20.2.3 dst_entry 结构

dst_entry 结构被用于存储缓存路由项中独立于协议的信息。三层协议在另外的结构中存储本协议中更多的私有信息（例如，IPv4 使用 rtable 结构）。

```
38 struct dst_entry
39 {
40    struct dst_entry      *next;
41    atomic_t       __refcnt;    /* client references    */
42    int           __use;
43    struct dst_entry    *child;
44    struct net_device      *dev;
45    short          error;
46    short          obsolete;
47    int          flags;
48 #define DST_HOST       1
49 #define DST_NOXFRM       2
50 #define DST_NOPOLICY       4
51 #define DST_NOHASH       8
52 #define DST_BALANCED          0x10
53    unsigned long       lastuse;
54    unsigned long       expires;
55
56    unsigned short       header_len;    /* more space at head required */
57     unsigned short        nfheader_len;    /* more non-fragment space at head
       required */
58    unsigned short       trailer_len;    /* space to reserve at tail */
59
60    u32          metrics[RTAX_MAX];
61    struct dst_entry    *path;
62
63    unsigned long       rate_last;    /* rate limiting for ICMP */
64    unsigned long       rate_tokens;
65
66    struct neighbour    *neighbour;
67    struct hh_cache       *hh;
68    struct xfrm_state    *xfrm;
69
70    int          (*input)(struct sk_buff*);
71    int          (*output)(struct sk_buff*);
72
73 #ifdef CONFIG_NET_CLS_ROUTE
74    __u32          tclassid;
75 #endif
76
77    struct  dst_ops          *ops;
78    struct rcu_head       rcu_head;
79
80    char          info[0];
81 };
```

40 struct dst_entry *next

用于将分布在同一个散列表桶内的 dst_entry 实例链接在一起。

41 atomic_t refcnt

引用计数。

42 int __use

该表项已经被使用的次数（即缓存查找返回该表项的次数）。

注意： 不要将这个值与 rt_cache_stat[smp_processor_id( )].in_hit 混淆：后者表示针对某个

CPU 的全局缓存命中次数。

```
43 struct dst_entry *child
56 unsigned short header_len
57 unsigned short nfheader_len
58 unsigned short trailer_len
61 struct dst_entry *path
68 struct xfrm_state *xfrm
```

这些字段与 IPsec 相关，本书不作论述。

```
44 struct net_device *dev
```

输出网络设备（即将报文送达目的地的发送设备）。

```
45 short error
```

当 fib_lookup()查找失败时，错误码值会被保存在这个字段中，在之后 ip_error()中使用该值来决定如何处理本次路由查找失败，即决定生成哪一类 ICMP 消息。

```
46 short obsolete
```

用于标识本 dst_entry 实例的可用状态，见表 20-3。

表 20-3 obsolete 取值

obsolete	描述
0（默认值）	表示所在结构实例有效而且可以被使用
2	表示所在结构实例将被删除因而不能被使用
-1	被 IPsec 和 IPv6 使用但不被 IPv4 使用

```
47 int flags
```

标志集合，见表 20-4。

表 20-4 flags 取值

flags	描述
DST_HOST	表示主机路由，即不是到网络或到一个广播/多播地址的路由
DST_NOXFRM DST_NOPOLICY DST_NOHASH	只用于 IPsec，本书不作论述

```
53 unsigned long lastuse
```

记录该表项最后一次被使用的时间戳。当缓存查找成功时更新该时间戳，垃圾回收程序使用该时间戳来决定最应该被释放表项。

```
54 unsigned long expires
```

表示该表项将过期的时间戳。

```
60 u32 metrics[RTAX_MAX]
```

多种度量值，TCP 中多处使用。

```
63 unsigned long rate_last
64 unsigned long rate_tokens
```

这两个字段被用于对两种类型的 ICMP 消息限速。