Programowanie sieciowe

Obsługa tablicy routingu i pamięci podręcznej protokołu ARP

Michał Kalewski

Instytut Informatyki
Politechnika Poznańska
ul. Piotrowo 2, 60–965 Poznań

mkalewski@cs.put.poznan.pl
http://www.cs.put.poznan.pl/mkalewski

Poznań · 2 kwietnia 2019 r.

Plan wykładu

Pamięć podręczna protokołu ARP

Pamięć podręczna protokołu ARP (1/2)

Pamięć podręczna protokołu ARP (ang. *ARP cache*) przechowuje odwzorowania adresów IP w adresy warstwy łącza danych (np. adresy MAC), które protokół ten dotychczas rozwiązał.

Każdy wpis w pamięci podręcznej protokołu ARP zawiera następujące informacje:

- adres IPv4;
- adres warstwy łącza danych (np. MAC) — jeżeli jest on znany;
- typ adresu warstwy łącza danych (domyślnie jest to „`ether`”);
- maska adresu warstwy łącza danych;
- flagi: `C` („complete’‘), `M` („permanent’‘), `P` („publish”);
- interfejs sieciowy, poprzez który nastąpiło odwzorowanie.

zob. `arp(7)`, `arp(8)`, `if_arp.h`

Zawartość pamięci podręcznej protokołu ARP dostępna jest także w pliku `/proc/net/arp`.

zob. `proc(5)`

https://github.com/torvalds/linux/blob/master/include/uapi/linux/if_arp.h

Pamięć podręczna protokołu ARP (2/2)

Przykład:

```
$ arp -n
Address      HWtype  HWaddress          Flags  Mask  Iface
192.168.1.1  ether   00:05:97:84:9e:91  C            eth0
$ arp
Address      HWtype  HWaddress          Flags  Mask  Iface
sundown      ether   00:05:97:84:9e:91  C            eth0
$ cat /proc/net/arp
IP address   HW type  Flags  HW address         Mask  Device
192.168.1.1  0x1      0x2    00:05:97:84:9e:91  *     eth0
$ arp -d 192.168.1.1  ;  arp -n
Address      HWtype  HWaddress          Flags  Mask  Iface
192.168.1.1          (incomplete)                    eth0
$ arp -s 192.168.1.1 00:11:22:33:44:55  ;  arp -n
Address      HWtype  HWaddress          Flags  Mask  Iface
192.168.1.1  ether   00:11:22:33:44:55  CM           eth0
```


Tablica routingu

Tablica routingu (1/3)

Tablica routingu zawiera adresy IP sąsiednich (tj. działających w tej samej sieci IP) routerów, przez które wiedzie trasa do oddalonych (tj. niedostępnych bezpośrednio) sieci IP.

Każdy wpis w tablicy routingu zawiera m.in. następujące informacje:

- adres IP sieci docelowej;
- adres IP routera sąsiedniego na trasie do sieci docelowej;
- maskę adresu IP sieci docelowej;
- flagi;
- metrykę trasy („odległość” do sieci docelowej);
- liczbę referencji do danej trasy (wartość ta nie jest wykorzystywana przez jądro systemu operacyjnego GNU/Linux);
- liczbę trafień/chybień zapytań do pamięci podręcznej tablicy routingu.

zob. `route(8)`

Zawartość tablicy routingu dostępna jest także w pliku `/proc/net/routing`.

zob. `proc(5)`

Tablica routingu (2/3)

Flagi dla wpisów w tablicy routingu są następujące:

- `U` („route is up”),
- `H` („target is a host”),
- `G` („use gateway”),
- `R` („reinstate route for dynamic routing”),
- `D` („dynamically installed by daemon or redirect”),
- `M` („modified from routing daemon or redirect”),
- `A` („installed by addrconf”),
- `C` („cache entry”),
- `!` („reject route”).

Akceptacja komunikatów ICMP `redirect` wymaga ustawienia wartości `1` w pliku `/proc/sys/net/ipv4/conf/*/accept_redirects`.

Tablica routingu (3/3)

Przykład:

```
$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         150.254.31.1    0.0.0.0         UG    0      0        0 eth0
150.254.31.0    0.0.0.0         255.255.255.128 U     0      0        0 eth0
$ cat /proc/net/route
Iface   Destination     Gateway         Flags   RefCnt  Use     Metric  Mask            MTU     Window  IRTT
eth0    00000000        011FFE96        0003    0       0       0       00000000        0       0       0
eth0    001FFE96        00000000        0001    0       0       0       80FFFFFF        0       0       0
```


Pamięć podręczna tablicy routingu

Zawartość pamięci podręcznej tablicy routingu dostępna jest w pliku `/proc/net/rt_cache` (dla jąder do wersji 3.6).

zob. `proc(5)`

Przykład:

```
$ route -Cn
Kernel IP routing cache
Source          Destination     Gateway         Flags Metric Ref    Use Iface
150.254.30.30   212.191.241.20  150.254.30.1        0      0      6 eth0
$ cat /proc/net/rt_cache
Iface   Destination     Gateway         Flags   RefCnt  Use     Metric  Source          MTU     Window  IRTT    TOS     HHRef   HHUptod SpecDst
eth0    0FAA34B9        011EFE96        0       0       0       0       1E1EFE96        1492    0       0       00      -1      0       1E1EFE96
```


Żądania funkcji systemowej `ioctl(2)`

Pamięć podręczna protokołu ARP (1/2)

W systemach operacyjnych GNU/Linux, do obsługi pamięci podręcznej protokołu ARP dostępne są trzy żądania:

`SIOCSARP` dodaje nowy wpis do pamięci podręcznej;

`SIOCDARP` usuwa istniejący wpis z pamięci podręcznej;

`SIOCGARP` pobiera informację o odwzorowaniu z pamięci podręcznej.

Dla wszystkich trzech żądań, trzecim argumentem wywołania funkcji systemowej `ioctl(2)` jest wskaźnik na strukturę `arpreq`:

```
1 struct arpreq {
2   struct sockaddr arp_pa;       /* protocol address         */
3   struct sockaddr arp_ha;       /* hardware address         */
4   int             arp_flags;    /* flags                    */
5   struct sockaddr arp_netmask;  /* netmask (only for proxy) */
6   char            arp_dev[16];
7 };
```

Pamięć podręczna protokołu ARP (2/2)

Flagi dla wpisów w pamięci podręcznej protokołu ARP:

```
#define ATF_COM         0x02 /* completed entry (ha valid)  */
#define ATF_PERM        0x04 /* permanent entry             */
#define ATF_PUBL        0x08 /* publish entry               */
#define ATF_USETRAILERS 0x10 /* has requested trailers      */
#define ATF_NETMASK     0x20 /* want to use a netmask (only
                                for proxy entries) */
#define ATF_DONTPUB     0x40 /* don't answer this addresses */
```

zob. `if_arp.h`

https://github.com/torvalds/linux/blob/master/include/uapi/linux/if_arp.h

</> Pobranie odwzorowania protokołu ARP (1/2)

```
int fd;
struct arpreq arq;
struct sockaddr_in* sin;
unsigned char* mac;

memset(&arq, 0, sizeof(arq));
sin = (struct sockaddr_in*) &arq.arp_pa;
sin->sin_family = AF_INET;
sin->sin_addr.s_addr = inet_addr(argv[1]);
strcpy(arq.arp_dev, IFNAME);
fd = socket(PF_INET, SOCK_DGRAM, 0);
ioctl(fd, SIOCGARP, &arq);
printf("%s\t", inet_ntoa(sin->sin_addr));
```

cdn.

</> Pobranie odwzorowania protokołu arp (2/2)

ciąg dalszy

```
if (arq.arp_flags & ATF_COM) {
  mac = (unsigned char *) &arq.arp_ha.sa_data[0];
  printf("%02x:%02x:%02x:%02x:%02x:%02x",
         mac[0], mac[1], mac[2], mac[3], mac[4], mac[5]);
  printf("\tC");
  if (arq.arp_flags & ATF_PERM) printf("M");
  if (arq.arp_flags & ATF_PUBL) printf("P");
  if (arq.arp_flags & ATF_USETRAILERS) printf("T");
  if (arq.arp_flags & ATF_NETMASK) printf("N");
} else
  printf("(incomplete)");
printf("\n");
close(fd);
```


Tablica routingu (1/3)

W systemach operacyjnych GNU/Linux, do obsługi tablicy routingu dostępne są następujące żądania:

`SIOCADDRT` dodaje nowy wpis do tablicy routingu;

`SIOCDELRT` usuwa wpis z tablicy routingu.

Do obu żądań, trzecim argumentem wywołania funkcji `ioctl(2)` jest wskaźnik na strukturę `rtentry`.

zob. route.h

https://github.com/torvalds/linux/blob/master/include/uapi/linux/route.h

Tablica routingu (2/3)

```c
struct rtentry {
  unsigned long   rt_pad1;
  struct sockaddr rt_dst;     /* target address             */
  struct sockaddr rt_gateway; /* gateway addr (RTF_GATEWAY) */
  struct sockaddr rt_genmask; /* target network mask (IP)   */
  unsigned short  rt_flags;
  short           rt_pad2;
  unsigned long   rt_pad3;
  void            *rt_pad4;
  short           rt_metric; /* +1 for binary compatibility!*/
  char            *rt_dev;   /* forcing the device at add   */
  unsigned long   rt_mtu;    /* per route MTU/Window        */
#define rt_mss  rt_mtu       /* Compatibility :-(           */
  unsigned long   rt_window; /* Window clamping             */
  unsigned short  rt_irtt;   /* Initial RTT                 */
};
```


Tablica routingu (2/3)

Flagi dla wpisów do tablicy routingu:

```c
#define RTF_UP        0x0001  /* route usable                 */
#define RTF_GATEWAY   0x0002  /* destination is a gateway     */
#define RTF_HOST      0x0004  /* host entry (net otherwise)   */
#define RTF_REINSTATE 0x0008  /* reinstate route after tmout */
#define RTF_DYNAMIC   0x0010  /* created dyn. (by redirect)   */
#define RTF_MODIFIED  0x0020  /* modified dyn. (by redirect) */
#define RTF_MTU       0x0040  /* specific MTU for this route */
#define RTF_MSS       RTF_MTU /* Compatibility :-(           */
#define RTF_WINDOW    0x0080  /* per route window clamping    */
#define RTF_IRTT      0x0100  /* Initial round trip time      */
#define RTF_REJECT    0x0200  /* Reject route                 */
```


</> Dodanie bramy domyślnej (1/2)

```c
int fd;
struct rtentry route;
struct sockaddr_in* addr;

fd = socket(AF_INET, SOCK_DGRAM, 0);

memset(&route, 0, sizeof(route));

addr = (struct sockaddr_in*) &route.rt_gateway;
addr->sin_family = AF_INET;
addr->sin_addr.s_addr = inet_addr("192.168.1.1");

addr = (struct sockaddr_in*) &route.rt_dst;
addr->sin_family = AF_INET;
addr->sin_addr.s_addr = INADDR_ANY;
```

cdn.

</> Dodanie bramy domyślnej (2/2)

ciąg dalszy

```c
addr = (struct sockaddr_in*) &route.rt_genmask;
addr->sin_family = AF_INET;
addr->sin_addr.s_addr = INADDR_ANY;

route.rt_flags = RTF_UP | RTF_GATEWAY;

route.rt_metric = 0;

ioctl(fd, SIOCADDRT, &route));

close(fd);
```

Podsumowanie

Podsumowanie

- Pamięć podręczna protokołu ARP, tablica routingu oraz pamięć podręczna tablicy routingu.
- Żądania funkcji systemowej `ioctl(2)` do obsługi pamięci podręcznej protokołu ARP;
  - struktura `arpreq`;
  - flagi dla wpisów w pamięci podręcznej protokołu ARP.
- Żądania funkcji systemowej `ioctl(2)` do obsługi tablicy routingu;
  - struktura `rtentry`;
  - flagi dla wpisów do tablicy routingu.

Pytania i zadania

Pytania i zadania — praca własna

1. Protokół IPv6 nie wykorzystuje protokołu ARP, jak zatem w tym przypadku realizowane są odwzorowania adresów IP w adresy warstwy łącza danych?
2. Dowiedz się, dlaczego pamięć podręczna tablicy routingu została usunięta z jądra systemów operacyjnych GNU/Linux.
3. Zapoznaj się z dokumentacją modułu ARP jądra systemu operacyjnego GNU/Linux, *zob.* `arp(7)`.
4. Zapoznaj się wykorzystaniem polecenia `ip(8)` do manipulacji pamięcią podręczną protokołu ARP, *zob.* `ip-neighbour(8)`.
5. Zapoznaj się wykorzystaniem polecenia `ip(8)` do manipulacji tablicą routingu, *zob.* `ip-route(8)`.

Dziękuję za uwagę!