Csoportszórásos útválasztókód a rendszermagban

Tegyük lehetővé, hogy egyetlen csomag több címre is megérkezhessen, és takarítsunk meg jókora sávszélességet! – ezt ígéri a csoportszórás (IP multicasting). Lássuk, vajon hogyan kezeli a Linux!

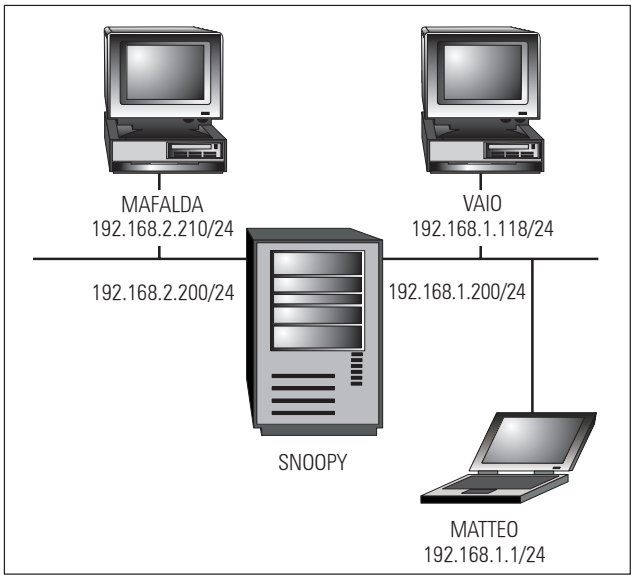
Ezeken az oldalakon arról fogok beszélni, miképpen kezeli a Linux-rendszermag a csoportszórásos (multicast) forgalmat, illetve hogyan lehet a rendszermagkód egyszerű foltozásával belenyúlni. Igaz, meglehetősen egyedi témát vetünk fel, az itt elhangzó dolgok azonban bárki számára izgalmasak lehetnek, akit érdekel a csoportszórásos útválasztás. Ha figyelni, esetleg módosítani szeretnénk valamilyen létező csoportszórásos protokollt, hasznos segítséget nyújthatnak az alábbiak.

A Milánói Egyetemen egy új, CAMP (Call Admission Multicast Protocol) névre keresztelt protokollt fejlesztünk, ami néhány fontos döntéséhez a csoportszórásos rendszermagkód által nyújtott adatokat használja fel. Képesnek kell lennünk tehát a fontos eseményekről (például a `JOIN`- és `LEAVE`-kérésekről) szóló értesítések fogadására. Mint azt valószínűleg már többen ismerik, a Linux-rendszermag csoportszórásos útválasztóként is képes működni, támogatva mind az 1-es, mind a 2-es PIM (Protocol Independent Multicast, ➲ http://netweb.usc.edu/pim) protokollt. Valamennyi MFC (Multicast Forwarding Cache) frissítési műveletet egy a rendszermaggal kapcsolatot tartó, felhasználói módban futó külső folyamat végzi el. Ebben a cikkben azt fogjuk vizsgálni, hogyan használja fel a rendszermag a felhasználó módú démon által küldött üzeneteket az MFC frissítésére. Rövid bemutatás után részletesebben is ismertetjük a saját megoldásunkat. *Ábránkon* a kipróbáláshoz használt alhálózatunk szerkezetét mutatjuk be. Mint látható, a Snoopy a `PIMd` (2.1.0-alpha 29.9 változatát) a Linux 2.4.18-as rendszermagváltozat felett futtatva működik csoportszórásos útválasztóként.

A rendszermag minden csoportszórással kapcsolatos kódja két fájlban található meg: ***ipmr.c*** (***net/ipv4/ipmr.c***) és ***mroute.h*** (***include/linux/mroute.h***). Mielőtt belenéznénk a kódba, két másik fájlt is meg kell említenünk: a ***/proc/net/ip_mr_vif-et*** és a ***/proc/net/ip_mr_cache***-t. Ahogy hamarosan látni fogjuk, ez a két fájl különösen fontos a csoportszórásos útválasztó pillanatnyi állapotának felmérése során. Az ***ip_mr_vif*** fájl felsorolja a csoportszórásos műveletekbe bevont valamennyi virtuális csatolófelületet, míg az ***ip_mr_cache*** az MFC pontos állapotáról ad felvilágosítást.

Példaképpen elkezdünk csoportszórásos forgalmat küldeni a Vaióról a `224.225.0.1` címre. A csomagokat a Snoopy fogadja, de nem küldi tovább az `eth1`-re, hiszen még nem érkezett `JOIN`-jel a Mafaldáról. Most nézzük azt az állapotot, amikor a Mafalda kiad egy `JOIN`-t a `224.225.0.1` címre. Az *1.* és *2. táblázatban* láthatjuk a két proc fájl tartalmát (a Snoopy gépen).

Az ***ip_mr_vif*** szerint két csatoló (`eth0` és `eth1`) vesz részt csoportszórásos útválasztásunkban, míg a harmadikként felbukkanó csatoló (pimreg) a csoportszórás-kezelő által bejegyzett virtuális eszköz. A 2. táblázatban a `PIMd` által felépített útvonalat láthatjuk. Az `eth0 (Iif: 0)` csatolón a `192.168.1.118` címmel azonos forrásból (Origin: 7601A8C0) érkező és `224.225.0.1 (Group: 0100E1E0)` címre szánt



A próbahálózat szerkezete

csomagok az `eth1` (`Oifs: 1:` – a második számot egyelőre hagyjuk figyelmen kívül) csatolóra irányítódnak át. Most a Mafaldáról adjunk ki egy `LEAVE` üzenetet. Néhány másodperc múltán a ***/proc/net/ip_mr_cache*** tartalma megváltozik. Az eredményt a *3. táblázatban* találjuk.

Továbbra is ugyanaz maradt a csatolónk és a csoportjelzésünk, a bemenő csatoló (input interface) viszont már `-1`-re változott, kimenő csatolnk pedig egyáltalán nincs. Ez azért van így, mert a csomagok továbbra is érkeznek az `eth0`-n, csak éppen miután a többi csatolón senki nem szándékozik megkapni őket, valamennyit el kell vetni. Amikor a csomagokat ilyen okból vetjük el, mindig új bejegyzés készül a feloldatlan címekhez szánt különleges sorban. A feloldatlan cím létét jelzi, hogy a bemenő csatoló értéke `-1` lesz, illetve nem fogunk kimenő csatolót találni.

A feloldatlan című csomagok ilyenfajta sorolásának megvan a maga oka: az `IGMP`-csomag fogadása és a megfelelő MFC-bejegyzés létrehozása ugyanis igen sok időt vesz igénybe (próbahálózatunkon is közel 2–3 másodpercet, egymással kapcsolatban álló, csoportszórásos útválasztókat tartalmazó nagyobb hálózatokon akár 20–30 másodpercet). Amikor előző példánkban a Vaio csoportszórásos csomagokat kezdett el továbbítani, igen nagy esély volt rá, hogy a Mafalda `JOIN` üzenetét a Snoopy gép még nem kezelte, és a csomagok továbbításához szükséges megfelelő MFC-bejegyzés sem készült még el. Ezért, hogy a Vaio által küldött csomagokat ne veszítsük el addig, amíg a `JOIN`-kérelem végrehajtódik és az új MFC-bejegyzés elkészül, egy különleges gyorstárba helyezzük őket. Amint

1. táblázat **A /proc/net/ip_mr_vif tartalma**

Interface	BytesIn	PktsIn	BytesOut	PktsOut	Flags	Local	Remote
0 eth0	129090	1655	1872	24	00000	C801A8C0	00000000
1 eth1	1872	24	129090	1655	00000	C802A8C0	00000000
2 pimreg	0	0	0	0	00004	C801A8C0	00000000

az MFC-bejegyzés elkészül, a sort felszabadítjuk, és a benne várakozó csomagokat a megfelelő célnak továbbítjuk. Nyilvánvaló, hogy teljesítmény- és memóriakorlátok miatt ez a sor nem nőhet túlságosan nagyra. Ezt egy nagyon egyszerű időzítő szolgáltatás hozzáadása oldja meg, ami időről időre törli a feloldatlan bejegyzéseket (`ipmr_expire_process()`).
Vizsgáljuk meg a folyamatban felhasznált adatszerkezeteket (lásd az *1. listán*).
A `vif_device` a valós hálózati csatolóhoz rendelt virtuális eszköz. A `dev` mező a valós csatolóeszközt (hardware interface) képviselő `net_device` szerkezetre irányított mutató. Számunkra azonban az `mfc_cache` szerkezete még érdekesebb. Mezői önmagukért beszélnek, és a 2., illetve a 3. táblázatban közölt adatoknak felelnek meg.
A ***ipmr.c***-ben használt három fő változó a következő:

```
/* Eszk z k                */
static struct vif_device vif_table[MAXVIFS];

/* TovÆbb t  gyorstÆr      */
static struct mfc_cache
➥*mfc_cache_array[MFC_LINES];

/* Feloldatlan bejegyzØsek sora */
static struct mfc_cache *mfc_unres_queue;
```

A `vif_table` nem más, mint a `PIMd` által készített virtuális csatolókat tartalmazó tömb; az `mfc_cache_array` az MFC-t képviseli; az `mfc_unres_queue` a fent leírt feloldatlan bejegyzések listája. A csoportszórás-kezelő kód elemzése előtt talán érdemes pár szót ejteni az `mfc_cache` szerkezet egyik alkotóeleméről: a `TTL` tömbről. A tömb minden egyes értéke közvetlenül a `vif_table`-hez van rendelve. Tulajdonképpen minden egyes csatolófelülethez rendelt összes csoportszórásos címhez tartozik egy-egy bájtérték, ami a `TTL`-határt adja meg. Mint azt később látni fogjuk, ezt az értéket hasonlítjuk majd minden IP-csomag `TTL` mezőjéhez, ha a csomag továbbításáról kell majd döntenünk.
Most, hogy már láttuk a csoportszórásos útválasztás adatainak alapszerkezetét, elkezdhetjük vizsgálni, miképpen kezeli őket a rendszermag. Az összes szolgáltatást egyetlenegy fájlban találjuk meg: az ***ipmr.c***-ben. Ne feledjük, hogy az itt található kód magát az útvonalválasztó kódot nem tartalmazza. Az ***ipmr.c***-ben található függvényeket a csoportszórásos útvonalválasztó démon (jelen esetben a `PIMd`) fogja felhasználni a fenti adatszerkezetek kezeléséhez. Egyszerűen szólva, amikor a `PIMd` úgy dönt, hogy itt az ideje felvenni vagy törölni egy adott útvonalat, csak küld egy üzenetet a rendszermagnak, amelyben megadja, hogy milyen feladatot kell végrehajtania. Hogy ezt megtehesse, a `PIMd`-nek képesnek kell lennie `IGMP` csomagok fogadására – ezeket a rendszermag adja át a felhasználótérnek. A `PIMd` két különböző módon tartja a kapcsolatot a rendszermaggal: `ioctl`-ek útján, illetve `setsockopt()` rendszerhívásokkal. A ***vif*** és ***mfc*** táblákat egyaránt `setsockopt()` rendszerhívásokkal éri el.
Hogy jobban megérthessük működésének a lényegét, nézzünk meg a `PIMd` kódjának néhány részletét. Különös figyelmet érdemelnek a rendszermaggal társalgó függvények, amelyeket a `PIMd` terjesztés ***kern.c*** fájljában találjuk meg. A `k_chg_mfc()` függvény felelős az új MFC-bejegyzések felvételéért, illetve a már létező elemek módosításáért, míg az elemek törlését a `k_del_mfc()` végzi. Hogy a rendszermagnak megmondhassuk, hogyan kell a csoportszórásos csomagokat továbbítania, néhány, az `mfc_cache` szerkezetekben felsoroltakhoz hasonló adatot át kell adnunk a felhasználói démonból. Lényeges, hogy ezt az adatot be kell csomagolnunk az `mfcctl`-ként meghatározott új szerkezetbe (*2. lista*).
A példánkban használt mezőneveknek maguktól értetődőknek kell lenniük. Mégis fontos kiemelni az `mfc_ttls` tömb szerepét. Mint azt korábban elmondtuk, ez az érték jelzi a `TTL`-határt; a felhasználó-térbeli démon azonban ezt a mennyiséget egy kicsit más módon kezeli. A `k_chg_mfc()` függvénynek kell a rendszermagnak megmondania, hogy melyik csatolófelületeken kell továbbítani a csoportszórásos csomagot. Ehhez viszont a kimenő felületek listáját kell átadni – ezt a szerepet a `mfcc_ttls` tölti be. Az elvet az alábbi kódrészlet mutatja be:

```
for (vifi = 0, v = uvifs;
     vifi < numvifs; vifi++, v++)
{

if (VIFM_ISSET(vifi, oifs))
             mc.mfcc_ttls[vifi] =
             ➥v->uv_threshold;
         else
             smc.mfcc_ttls[vifi] = 0;
}
```

Amennyiben a csatolófelület éppen egy csoportszórásos cím kimeneti csatolófelülete, a hozzá tartozó `TTL` határ valós értéket vesz fel; egyébként az értéke nulla lesz. A nulla értéket

2. táblázat **A /proc/net/ip_mr_cache tartalma**

Group	Origin	Iif	Pkts	Bytes	Wrong Oifs
0100E1E0	7601A8C0	0	755	58890	0 1:1

3. táblázat **A /proc/net/ip_mr_cache tartalma a LEAVE-üzenetet követően**

Group	Origin	Iif	Pkts	Bytes	Wrong Oifs
0100E1E0	7601A8C0	-1	4	0	0

1. lista A csoportszórásos útvonalválasztó kódban használt „vif_device” és „mfc_cache” szerkezetek

```
struct vif_device
{
    /* A hasznÆlt eszk z */
    struct net_device      *dev;

    /* KimutatÆsok */
    unsigned long     bytes_in,bytes_out;
    unsigned long     pkt_in,pkt_out;

    /* ForgalomirÆny tÆs (NI) */
    unsigned long     rate_limit;

    /* TTL hatÆr */
    unsigned char     threshold;

    /* vezØrlı zÆszl  */
    unsigned short     flags;

    /* C mek (remote a tunnelekhez)*/
    __u32         local,remote;

    /* fizikai csatol fel let-index */
    int         link;
};

struct mfc_cache
{
    /* a gyorstÆrsor k vetkezı bejegyzØse */
    struct mfc_cache *next;

    /* a bejegyzØs melyik csoportba tartozik */
    __u32 mfc_mcastgrp;

    /* a csomag forrÆsa */
    __u32 mfc_origin;

    /* forrÆscsatol  */
    vifi_t mfc_parent;

    /* jelzık */
    int mfc_flags;

    union {
        struct {
            unsigned long expires;
            /* Feloldatlan puffer     */
            struct sk_buff_head unresolved;
        } unres;
        struct {
            unsigned long last_assert;
            int minvif;
            int maxvif;
            unsigned long bytes;
            unsigned long pkt;
            unsigned long wrong_if;
            /* TTL hatÆr */
            unsigned char ttls[MAXVIFS];
        } res;
    } mfc_un;
};
```

2. lista A PIMd-ben használt mfcctl

```
struct mfcctl
{
    /* mcast eredete  */
    struct in_addr mfcc_origin;

    /* A kØrdØses csoport    */
    struct in_addr mfcc_mcastgrp;

    /* hova Ørkezik    */
    vifi_t     mfcc_parent;

    /* hova megy     */
    unsigned char mfcc_ttls[MAXVIFS];

    /* csomagszÆmlÆl  a forrÆscsoporthoz */
    unsigned int mfcc_pkt_cnt;
    unsigned int mfcc_byte_cnt;
    unsigned int mfcc_wrong_if;
    int          mfcc_expire;
};
```

a rendszermag az adott csoporthoz tartozó, nem kimenő csatolófelületnek fogja értelmezni, következésképpen az `mfc_cache` szerkezet megfelelő bájtját a 255-ös (decimális) értékre állítja be, és a csomagot nem továbbítja.

Most pedig nézzük meg, mit is tesz a rendszermag, amikor egy új MFC-bejegyzés beillesztését kérő üzenetet kap. A kérést az `ipmr_mfc_add()` függvény kezeli. A rendszermag ellenőrzi az MFC jelenlegi bejegyzéseit, hátha csak egyszerű frissítésről van szó. Ha talál megfelelő elemet, az új `TTL` értékeket a már meglévő `mfc_cache` szerkezetbe másolja, illetve egyúttal a `minvif` és `maxvif` értékeket is frissíti. Ezek az értékek egy adott üzenetszórt címhez tartozó összes kimenő csatoló index-értékeinek két határértékét tartalmazzák. Ezt a munkát az `ipmr_update_thresholds()` végzi. Az érthetőség kedvéért a 3. listában ezt a függvényt is bemutatjuk, mivel így jobban látható a `minval` és `maxval` mezők jelentése.

De térjünk vissza az `ipmr_mfc_add()` függvényünkhöz! Tételezzünk fel egy olyan esetet, ahol nem találunk létező MFC-bejegyzést. Ez esetben új szerkezetet kell lefoglalni, majd beszúrni az MFC-táblába. A művelet befejezése után a rendszermagnak már csak egy dolgot kell elvégeznie: továbbítania kell az összes olyan, az `mfc_unres_queue` sorban várakozó feloldatlan csoportszórásos csomagot, amelynek a frissen beillesztett csomópont lett volna a célja. Ha talál ilyen csomagot, eltávolítja és továbbítja az új csatolófelületre.

A másik feladat, amit végre kell hajtani: az MFC törlése. Ez eléggé magától értetődő – az adatszerkezetek alapjában véve ugyanazok, mint amiket eddig láttunk. A gyorstárbejegyzés eltávolításához a felhasználó módú démon a `k_del_mfc()` függvényt hívja meg, a rendszermagmód-kezelő (handler) pedig meghívja az `impr_mfc_delete()`-t. Ez a függvény

3. lista Így frissíti a meglévő MFC-bejegyzéseket a rendszermag

```
static void
ipmr_update_thresholds(struct mfc_cache *cache,
                       unsigned char *ttls)
{
    int vifi;

    cache->mfc_un.res.minvif = MAXVIFS;
    cache->mfc_un.res.maxvif = 0;
    memset(cache->mfc_un.res.ttls, 255, MAXVIFS);

    for (vifi=0; vifi<maxvif; vifi++) {
        if (VIF_EXISTS(vifi) &&
            ttls[vifi] && ttls[vifi] < 255) {
            cache->mfc_un.res.ttls[vifi] =
              ttls[vifi];
            if (cache->mfc_un.res.minvif > vifi)
                cache->mfc_un.res.minvif = vifi;
            if (cache->mfc_un.res.maxvif <= vifi)
                cache->mfc_un.res.maxvif = vifi + 1;
        }
    }
}
```

egyszerűen törli a megadott bejegyzést az MFC-ből.
Most már tudjuk, hol kerülnek be, hogyan törlődnek és módosulnak az MFC-bejegyzések, itt az ideje, hogy a csoportszórásos útválasztókód eltérítésével (hooking) is elkezdjünk foglalkozni. A függvényeltérítés (hooking) meglehetősen egyszerű eljárás. Tulajdonképpen arról van szó, hogyha a korábban bemutatott módon egy függvényt illesztünk a kód közepébe, a rendszer a rendszermagmodulban megadott külső függvényre vált. Ezáltal a modulban megadott függvényt visszahívott függvénynek (callback) tekinthetjük, ami minden esetben meghívódik, valahányszor MFC-bejegyzés kerül be, módosul vagy törlődik. Eltérítő megoldásunk kivitelezését a 2.4.x rendszermagokban eleve megtalálható, jól ismert eltérítő megoldásra, a Netfilterre alapoztuk. Ha esetleg valaki még sose használta, annyit mindenképpen meg kell említeni, hogy népszerű programról van szó, amelyet csomagszűrési feladatok ellátására építettek a rendszermagba. Amit a Netfilter a csomagokkal végez, megtehető tetszőleges adatszerkezetekkel is. A mi esetünkben ez az `mfc_caches` lesz. A visszahívott függvény mintapéldánya a következő:

```
typedef void nf_nfy_msg(
    unsigned int hook,
    unsigned int msgno,
    const struct net_device
    *dev,
    void* moreData);
```

A `hook` jelen esetben a tartományt jelöli (ez az érték minden esetben a `PF_INET` értéket veszi fel); a `msgno` az üzenet számát jelöli (a rendszermag által végrehajtott művelet, például `mfc_cache`-bejegyzés hozzáadása); a `dev` a műveletben résztvevő bármelyik pillanatnyi `net_device` értékét felveheti; végül a `moreData` egy általános adatszerkezetet célzó, `void*` típusú mutató. Esetünkben ez az adatszerkezet az `mfc_cache` lesz.

Most, hogy már ismerjük a visszahívás formátumát, nézzük meg, miképpen hívja meg a rendszermag. Ez lényegében igen egyszerűen történik; a következő sorokat a rendszermagkódba helyezve az adott művelethez bármilyen bejegyzett eltérítő függvényt meghívhatunk:

```
#ifdef NF_EVT_HOOK && NF_MCAST_HOOK
     NF_NFY_HOOK(
          PF_INET,
          NF_NFY_IP_MCAST_MSG,
          IPMR_MFC_ADD,
          NULL,
          (void*) c);
#endif*
```

Az eltérítő függvény szerkezetének részletes ismertetése sajnos meghaladja cikkünk kereteit. Az olvasó tisztább képet alkothat magáról a kódról, ha megtekinti az általunk módosított rendszermagfájlokat (amik megtalálhatók a ➲ http://www.linuxvilag.hu/multi oldalon), vagy az eredeti Netfilter-megoldást.
Most, hogy végre teljes képet kaphattunk a linuxos csoportszórásos útvonalválasztás megvalósításáról, illetve pár dolgot megjegyeztünk az eltérítő függvények megoldása kapcsán, érdemes pár szót szánni a próbáink során felmerült néhány gondra is.
Térjünk vissza próbahálózatunkhoz, és képzeljük el a következő felállást: a Snoopy csoportszórásos forgalmat küld ki, miközben Mafalda és Vaio egyaránt `JOIN`-kérelmet továbbít. Azt várnánk, hogy új csoportszórásos útvonal jön létre az `eth0` és `eth1` kimenő csatolókkal. Sajnos nem ez fog történni. Ha vetünk egy pillantást a */proc/net/ip_mr_cache*-re, csak egyetlen útvonalat látunk a Mafaldára, ugyanakkor a csoportszórásos forgalom mégis zavartalanul eléri a Mafaldát és Vaiót egyaránt. Íme az ok: a Snoopyról kimenő csomagok forráscímként a `192.168.1.200`-as számot használják. Ezért aztán amikor az `eth0`-n küld adatot kifelé, a Snoopy úgy fog viselkedni, mintha a csoportszórásos forgalmat a helyi hálózaton bonyolítaná le. Ez azt jelenti, hogy Snoopy még azelőtt elkezd csomagokat küldeni az `eth0`-n, mielőtt a Vaio kiadná a `JOIN`-kérelmet, hiszen a rendszermag nem képes kihasználni a `PIM` démon azon képességét, amivel a más gépekről érkező `IGMP`-csomagokat is értelmezni tudja. Hogy a többi munkaállomás is megkapja a csoportszórásos forgalmat, egyszerűen kiküldi a csomagokat. Ebből következik, hogy az ugyanezen a hálózati szelvényen található bármely más gép – csoportszórásos hálózati kártyába épített szűrőjét bekapcsolva – a kívánt adatot leszedheti a drótról. Ugyanilyen okokból a csoportszórásos `JOIN`- és `LEAVE`-kérelmek sem irányíthatók az elsődleges csatolóra, mivel azon a csatolón a rendszermag nem végez pontos csoportszórásos útvonalválasztást.

Linux Journal 2002. november, 103. szám

Matteo Pelati
(matteo@dolce.it) tanulmányai mellett kutatói segédmunkatárs a milánói egyetemen. Elsődleges érdeklődési körébe tartoznak a hálózati szabványok, az operációs rendszerek és az utazás.