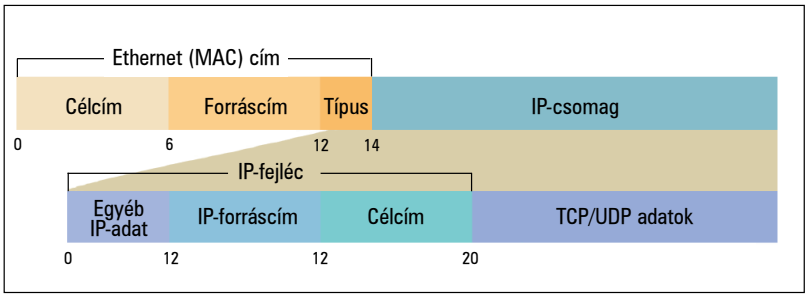
Csomagszűrő: bájtok leszippantása a hálózatról

A 2.2-es rendszermagban megjelent LSF segítségével a rendszermag beprogramozható, hogy melyik csomagot engedje be. Most megtudhatjuk, hogyan.

Abban az esetben, ha hálózati rendszergazda vagy, esetleg biztonsági kérdésekkel foglalkozol, illetve ha egyszerűen csak érdekel, hogy a helyi hálózatodon mi megy keresztül, néhány csomag leszippantása a hálózati kártyáról hasznos gyakorlat lehet a számodra is. Egy kis C programozással és alapvető hálózatos ismeretekkel felvértezve könnyen elfoghatod azokat az adatokat is, amelyek nem a te gépedre érkeznek. Ebben a cikkben ethernethálózatokról lesz szó, mert messze ez a legelterjedtebb a helyi hálózatok között. Később megvilágított okból azt is feltesszük, hogy a forrás- és a célgép ugyanahhoz a helyi hálózathoz tartozik.

Először is elevenítsük fel, hogyan működik egy átlagos ethernethálózati kártya. Akik már jártasak ezen a területen, azok nyugodtan ugorjanak a következő bekezdéshez. A felhasználói programokból származó IP-csomagok ethernetkeretekbe ágyazódnak be (így hívják az ethernetszakaszon átküldött csomagokat). Ezek egyszerűen nagyobb méretű, alacsonyabb szintű csomagok, amelyek tartalmazzák az eredeti IP-csomagot, valamint további adatokat is a célba juttatáshoz (lásd *1. ábra*). Ebben az esetben a cél IP-címe a cél 6 bájtos ethernetcímére képeződik le (ezt gyakran MAC-címnek hívják) az ARP-nak hívott átalakítás segítségével. Így a csomagot tartalmazó keret áthalad a forrást és a célt összekötő vezetéken. A keret valószínűleg hálózati eszközökön, például elosztókon vagy kapcsolókon is keresztülmegy, de mivel feltettük, hogy nem lépi át a helyi hálózatot, útválasztó vagy átjáró nincs benne.

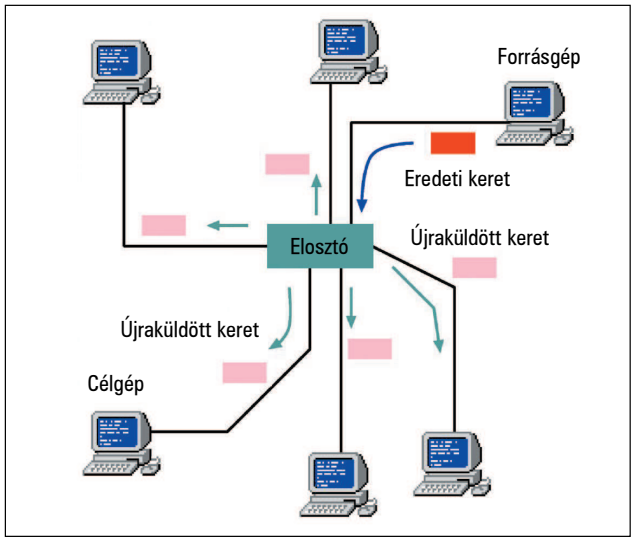


1. ábra IP-csomagok mint ethernetkeretek

Az ethernet szintjén nincs útválasztás. Más szavakkal: a forrásgép által küldött keret nem fog közvetlenül a célgépre jutni, ehelyett az a helyi hálózatot alkotó összes vezetéken megjelenik, és minden hálózati kártya látni fogja, amint elhalad (lásd *2. ábra*). Minden hálózati kártya elolvassa a keret első hat bájtját (amely az imént említett MAC-címet tartalmazza), de csak az a kártya olvassa be az egész keretet, amelyik a célcímben felismeri a saját címét. Ezen a ponton a keretet átveszi a hálózati meghajtó, visszaállítja az eredeti IP-csomagot, és továbbítja a fogadó alkalmazásnak a hálózati protokollvermen keresztül.

Pontosabban a hálózati meghajtó ellenőrzi a protokolltípus-mezőt az ethernetkeret belsejében (lásd 1. ábra), és ettől az értéktől teszi függővé, hogy melyik protokollfogadó függvénynek továbbítsa a csomagot. Az esetek többségében ez az IP protokoll lesz, a fogadó függvény kiveszi az IP-fejlécet, és a maradékot az UDP-t vagy TCP-t fogadó függvényeknek továbbítja. Ezek a protokollok azután továbbadják a csomagot a foglalatkezelő függvényeknek, amelyek végezetül a csomag adatait a felhasználói térben futó alkalmazásnak továbbítják. Az utazás során a csomag minden hálózattal kapcsolatos ismeretet elveszt, például a forrás címeit (IP és MAC), TCP-jellemzőket stb. Továbbá, ha a célgépen nincs nyitva a megfelelő kapu, a rendszermag eldobja a csomagot, amely így nem ér el az alkalmazások szintjére.

Ennek következtében a hálózaton utazó csomagok leszippantásához két különálló feladatot kell megoldanunk. Az első az ethernetcímzéssel kapcsolatos: nem tudjuk azokat a csomagokat elolvasni, amelyeket nem a mi gépünknek címeztek. A másik a protokollverem feldolgozásával függ össze: ha nem akarjuk elveszíteni a csomagot, minden egyes kaput nyitva kell tartani és figyelni kell. Ráadásul a csomag adatainak egy része a protokollverem feldolgozása során el is vész. Az első gond nem alapvető, ugyanis nem biztos, hogy érdekelnek minket a többi gépre küldött csomagok, lehet hogy csak a saját gépünkre küldötteket akarjuk leszippantani. A másodikat ellenben meg kell oldani. Megnézzük, hogyan célszerű kezelni ezt a két feladatot külön-külön, most kezdjük az utóbbival.



2. ábra Ethernetkeretek küldése helyi hálózaton keresztül

A PF_PACKET protokoll

Amikor egy foglalatot a szabványos `sock = socket (domain, type, protocol)` hívással megnyitsz, meg kell adnod, hogy melyik tartományt (vagy protokollcsaládot) fogod azzal a foglalattal használni. Gyakran használt család a PF_UNIX, amely a helyi gépen belüli párbeszédre használatos, és a PF_INET, amely az IPv4 protokollokon alapuló párbeszédekre jó. Meg kell továbbá adnod a foglalat típusát, melynek lehetséges értékei a megadott családtól függenek. A PF_INET család leggyakrabban használt típusai:

1. lista A leszippantott csomagok protokollveremszerű feldolgozása

```
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <linux/in.h>
#include <linux/if_ether.h>

int main(int argc, char **argv) {
  int sock, n;
  char buffer[2048];
  unsigned char *iphead, *ethhead;

  if ( (sock=socket(PF_PACKET, SOCK_RAW,
                    htons(ETH_P_IP)))<0) {
    perror("socket");
    exit(1);
  }

  while (1) {
    printf("----------\n");
    n = recvfrom(sock,buffer,2048,0,NULL,NULL);
    printf("%d bájt beolvasva\n",n);

    /* Ellenőrzi, hogy a csomag tartalmazza-e
    legalább
     * a teljes ethernet (14), IP (20) és
     * TCP/UDP (8) fejléceket.
     */
    if (n<42) {
      perror("recvfrom():");
      printf("Nem teljes csomag (errno is %d)\n",
             errno);
      close(sock);
      exit(0);
    }

    ethhead = buffer;
    printf("A forrás MAC-címe: "
           "%02x:%02x:%02x:%02x:%02x:%02x\n",
           ethhead[0],ethhead[1],ethhead[2],
           ethhead[3],ethhead[4],ethhead[5]);
    printf("A cél MAC-címe: "
           "%02x:%02x:%02x:%02x:%02x:%02x\n",
           ethhead[6],ethhead[7],ethhead[8],
           ethhead[9],ethhead[10],ethhead[11]);

    iphead = buffer+14; /* Az ethernetfejléc
                                    átugrása */
    if (*iphead==0x45) { /* Ellenőrzi az IPv4
                                    meglétét */
      printf("Forrásgép %d.%d.%d.%d\n",
             iphead[12],iphead[13],
             iphead[14],iphead[15]);
      printf("Célgép %d.%d.%d.%d\n",
             iphead[16],iphead[17],
             iphead[18],iphead[19]);
      printf("Forrás- és célkapuk %d,%d\n",
             (iphead[20]<<8)+iphead[21],
             (iphead[22]<<8)+iphead[23]);
      printf("Layer-4 protokoll
            %d\n",iphead[9]);
    }
  }

}
```

a SOCK_STREAM (ez a TCP-nek feleltethető meg) és a SOCK_DGRAM (az UDP megfelelője). A foglalat típusa befolyásolja a rendszermagot a csomagok kezelésében, mielőtt még azok az alkalmazáshoz kerülnének. Végül meg kell adnod a foglalaton átáramló csomagot kezelő protokollt (további részletek a socket(3) kézikönyvoldalon olvashatók).

A Linux-rendszermag újabb változataiban (a 2.0 utáni kiadásokban) új protokollcsaládot vezettek be, a PF_PACKET-et. Ez a család lehetővé teszi az alkalmazásoknak, hogy közvetlenül a hálózati kártya meghajtójával tárgyalva küldjenek és fogadjanak csomagokat, így elkerülik a szokásos protokollkezelést (például: TCP- és IP-, UDP-feldolgozást). Ez azt jelenti, hogy minden a foglalatra küldött csomag közvetlenül az ethernetcsatolóra kerül, és minden a csatolóra érkező csomag közvetlenül az alkalmazáshoz továbbítódik.

A PF_PACKET család két eléggé eltérő foglalattípust támogat, a SOCK_DGRAM és a SOCK_RAW típusokat. Előbbi a rendszermagra hagyja az ethernetszintű fejlécek hozzáadását és eltávolítását, utóbbi pedig teljesen az alkalmazásra. A protokollmező a socket() hívásban meg kell egyezzen a /usr/include/linux/if_ether.h-ban megadott ethernetazonosítók egyikével, amely az ethernetkeretben szállítható bejegyzett protokollokat adja meg. Hacsak nem nagyon sajátos protokollokról van szó, általában az ETH_P_IP-t kell használnod, amely magába foglalja az összes IP-protokollt (például: TCP, UDP, ICMP, nyers IP stb.).

Mivel ezeknek igen súlyos biztonsági következményeik lehetnek (például szerkeszthetsz egy hamis MAC-címmel rendelkező keretet), a PF_PACKET családot csak a rendszergazda használhatja.

A PF_PACKET család könnyedén megoldja a leszippantott csomagokkal kapcsolatos feladatainkat. Nézzük meg az *1. listán*, hogyan működik mindez. Megnyitunk egy PF_PACKET családhoz tartozó foglalatot, megadjuk a SOCK_RAW típust és az IP-vel kapcsolatos protokolltípust. Ezután elkezdünk olvasni a foglalatról, és néhány ellenőrzés után kinyomtatjuk az ethernetszintű adatokat, valamint az IP-fejlécet. A kinyomtatott címek és az *1. ábra* összevetéséből jól látható, milyen könnyű volt az alkalmazásból elérni a hálózati szintű adatokat.

Feltéve, hogy a géped ethernetes helyi hálózatba csatlakozik, kísérletezhetsz ezzel a kis példaprogrammal. Küldj csomagokat a gépedre egy másik gépről (pingelheted a gépedet vagy jelentkezz be Telnettel)! Látni fogod a neked szánt csomagokat, de nem fogod látni a többi gépre küldötteket.

Lehallgató és nem lehallgató mód

A PF_PACKET család lehetővé teszi, hogy az alkalmazások olyan formában szerezzék meg az adatcsomagokat, ahogy azok a hálózati kártyára érkeznek, de még ez sem engedi olyan csomagok elolvasását, amelyeket nem a gépnek címeztek. Ahogy az előbb láthattuk, ez amiatt történik, mert a hálózati kártya eldobja azokat a csomagokat, amelyek nem a saját MAC-címére érkeznek. Ezt a működési módot *nem lehallgató módnak* (nonpromiscuous) nevezik, ami egyszerűen azt jelenti, hogy az összes hálózati kártya csak a saját dolgával törődik és csakis a neki szóló kereteket olvassa el. Három

3. lista tcpdump -d eredménye

```
echer:~# tcpdump -d host 192.168.9.10
(000) ldh      [12]
(001) jeq      #0x800 jt 2   jf 6
(002) ld       [26]
(003) jeq      #0xc0a8090a   jt 12  jf 4
(004) ld       [30]
(005) jeq      #0xc0a8090a   jt 12  jf 13
(006) jeq      #0x806 jt 8   jf 7
(007) jeq      #0x8035jt 8   jf 13
(008) ld       [28]
(009) jeq      #0xc0a8090a   jt 12  jf 10
(010) ld       [38]
(011) jeq      #0xc0a8090a   jt 12  jf 13
(012) ret      #68
(013) ret      #0
```

4. lista tcpdump a -dd kapcsolóval

```
escher:~# tcpdump -dd host 192.168.9.01
{ 0x28, 0, 0, 0x0000000c },
{ 0x15, 0, 4, 0x00000800 },
{ 0x20, 0, 0, 0x0000001a },
{ 0x15, 8, 0, 0xc0a80901 },
{ 0x20, 0, 0, 0x0000001e },
{ 0x15, 6, 7, 0xc0a80901 },
{ 0x15, 1, 0, 0x00000806 },
{ 0x15, 0, 5, 0x00008035 },
{ 0x20, 0, 0, 0x0000001c },
{ 0x15, 2, 0, 0xc0a80901 },
{ 0x20, 0, 0, 0x00000026 },
{ 0x15, 0, 1, 0xc0a80901 },
{ 0x6, 0, 0, 0x00000044 },
{ 0x6, 0, 0, 0x00000000 },
```

kivétel van ez alól a szabály alól: ha a keret célcíme a különleges FF:FF:FF:FF:FF:FF MAC-cím – ezt minden kártya felveszi; ha a keret célcíme többszörös cím – ezt azok a kártyák veszik fel, amelyeken a többszörös cím fogadása engedélyezett; végül amelyik kártya lehallgató módban van – az összes csomagot felveszi, amit csak hall.

Természetesen céljaink elérése szempontjából az utóbbi a legérdekesebb. A hálózati kártyát egy nyitott foglalatára irányított megfelelő ioctl() hívással állíthatjuk lehallgató módba. Mivel ez a művelet biztonsági kockázatot hordoz magában, a hívást csak rendszergazdai jogosultságú felhasználó adhatja ki. Feltéve, hogy a „sock” egy már nyitott foglalat, a következő utasítás végzi el a feladatot:

```
strncpy(ethreq.ifr_name,"eth0",IFNAMSIZ);
ioctl(sock, SIOCGIFFLAGS, &ethreq);
ethreq.ifr_flags |= IFF_PROMISC;
ioctl(sock, SIOCSIFFLAGS, &ethreq);
```

(Az ethreq egy ifreq-szerkezet, amely a /usr/include/net/if.h-ban van megadva.) Az első ioctl kiolvassa az ethernetkártya pillanatnyi állapotjelzőit; ezt az értéket az IFF_PROMISC állandóval VAGY-oljuk, majd a második ioctl kiírja az új állapotjelzőt a kártyára. Nézzük meg ezt egy teljesebb példán (lásd a 2. listát a ➲ ftp://ftp.ssc.com/pub/lj/listings/issue86/ címen)! Lefordítás után rendszergazdaként futtatva egy helyi hálózathoz kapcsolt gépen az összes csomagot megláthatod, amely a vezetéken áramlik, akkor is, ha nem a te gépednek szánták őket. Ez azért lehetséges, mert hálózati kártyád lehallgató módban dolgozik. Könnyen meggyőződhetsz erről, ha kiadod az `ifconfig` parancsot és megfigyeled a kimenet harmadik sorát.

A Linux csomagszűrő

Mindkét leszippantással kapcsolatos gondunk mostanra megoldódott, van azonban még egy lényeges dolog, amelyet meg kell fontolnunk. Ha a valóságban is kipróbáltad a 2. listán bemutatott példát, mégha helyi hálózatodon nincs is túl nagy forgalom (néhány windowsos gép már elég ahhoz, hogy a sok NETBIOS csomagtól zsúfolt legyen a vezeték), észreveheted, hogy a szippantó túl sok adatot ír ki. Ahogy a hálózati forgalom nő, a szippantó kezdi elveszíteni a csomagokat, mert a számítógép nem tudja elég gyorsan feldolgozni őket.

A megoldás az lehetne, hogy szűröd az érkező csomagokat és csak azokról íratsz ki adatokat, amelyek érdekelnek. Az első ötlet ehhez az lehet, hogy *if* utasításokat szúrunk be a szippantó forrásába. Ez azonban noha szebbé teszi a szippantó kimenetét, a teljesítmény szempontjából nem lesz túl hatékony. A rendszermag ekkor is beolvassa a hálózaton áramló csomagokat, ami időpocsékolás, továbbá a szippantó ekkor is megvizsgálná az összes csomag fejlécét és döntene a csomaggal kapcsolatos adatok kinyomtatásáról.

Az eszményi megoldás az lenne, ha a szűrőt a lehető leghamarabb iktatnánk be a csomagfeldolgozó láncba (ez a hálózati meghajtó szintjén kezdődik és az alkalmazás szintjén ér véget, lásd a *3. ábrát*). A Linux-rendszermag lehetővé teszi, hogy berakjunk egy szűrőt (LPF a neve) közvetlenül a PF_PACKET protokollfeldolgozó műveletek belsejébe, amelyek röviddel azután futnak, hogy a hálókártya megszakítása kiszolgálásra került. A szűrő dönti el, hogy melyik csomag mehet tovább az alkalmazás felé, és melyik nem.

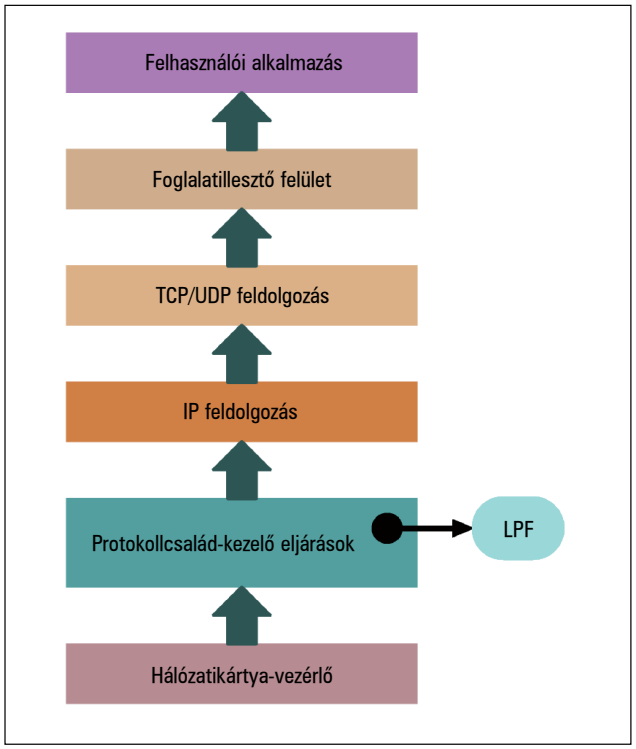
A lehető legnagyobb rugalmasság érdekében, és hogy a programozó keze ne legyen előre megadott feltételekkel megkötve, a csomagszűrőmotor számítógépszerűen megvalósított, amelyet a felhasználó programozhat. A programot egy különleges, gépi kódra emlékeztető nyelven kell megírni, amelyet BPF-nek hívnak (a Berkeley Packet Filter rövidítése), és amelyhez *Steve McCanne* és *Van Jacobson* egyik régi cikke adta az ötletet (lásd a *Kapcsolódó címeket*). A BPF tényleg Assembly nyelvnek látszik, egy pár regiszterrel és néhány utasítással, amelyek betöltik és tárolják az értékeket, számtani műveleteket és feltételes elágazást hajtanak végre.

A szűrőkód minden vizsgálandó csomagra lefut, és a BPF-feldolgozó a csomag adatait tartalmazó bájtokból álló tárterületen fejti ki működését. A szűrő eredménye egy egész szám, amely azt adja meg, hogy hány bájtot kell a csomagból a foglalatról az alkalmazás felé továbbítani. Ez további előnyökkel jár, mivel sokszor csak a csomag első néhány bájtja érdekes, így feldolgozási időt takaríthatunk meg, ha a többi másolását elkerüljük.

A szűrő programozása

Bár a BPF nyelv elég egyszerű és könnyű megtanulni, a legtöbben közülünk valószínűleg jobban szeretik az ember számára olvasható alakban megadott szűrőkifejezéseket. Ezért a BPF nyelv részleteinek és utasításkészletének bemutatása helyett (amelyet a fent említett cikkben megtalálhatsz) inkább azt fogjuk megtárgyalni, hogyan lehet egy működő szűrő kódját egy logikai kifejezésből előállítani.

Először is telepítened kell a tcpdump programot az LBL-től (lásd a *Kapcsolódó címeket*) – bár ha ezt a cikket olvasod, valószínűleg már ismered és használod a tcpdumpot. Az első változatokat ugyan-



3. ábra A csomagfeldolgozó lánc

azok írták, akik a BPF-cikket is, valamint a nyelv első megvalósítását. Valójában a tcpdump a BPF-et használja – a libpcap programkönyvtáron keresztül – a csomagok elkapására és szűrésére. A programkönyvtár operációs rendszertől független csomagoló a BPF-motor körül. Linuxon használva a BPF utasításait a Linux csomagszűrő hajtja végre.

A libpcap egyik leghasznosabb függvénye a pcap_compile(), amely logikai kifejezést tartalmazó karakterláncot alakít BPF szűrőkóddá. A tcpdump ezt a függvényt használja a felhasználó által a parancssorban megadott kifejezés működő BPF-szűrővé alakításához. A mi szempontunkból most a tcpdump egyik ritkán használt kapcsolója, a -d lesz érdekes, amely a szűrő kódját nyomtatja ki. Például a `tcpdump host 192.168.9.10` csak azokat a csomagokat szippantja le, amelyek forrás- vagy célcíme 192.168.9.10. A `tcpdump -d host 192.168.9.10` kiírja a szűrő BPF-kódját, ahogy az a *3. listán* látszik.

Fűzzünk rövid megjegyzéseket a kódhoz: a 0–1. és 6–7. sorok ellenőrzik a protokollazonosító alapján, hogy az elfogott keret IP, ARP vagy RARP protokollt hordoz-e (lásd /usr/include/linux/if_ether.h). A protokollazonosító a keret 12. bájtjánál van (lásd 1. ábra). Ha a próba sikertelen, a csomagot eldobjuk (13. sor).

A 2–5. és a 8–11. sorok összehasonlítják a forrás és a cél IP-címét 192.168.9.10-zel. A protokolltól függően a címek elhelyezkedése más, ha a protokoll az IP – ekkor 26 és 30, egyébként 28 és 38. Ha az egyik cím egyezik, akkor a szűrő elfogadja a csomagot, és az első 68 bájtot elküldi az alkalmazásnak (12. sor).

A szűrőkód nem mindig a legelőnyösebb, mivel egy általános BPF-gépre hozták létre, és nem igazították rá arra a rendszerre, amely a szűrőmotort futtatja. Az LPF esetében a szűrőt a PF_PACKET feldolgozófüggvények futtatják, amelyek esetleg már előbb ellenőrizték az ethernetprotokollt. Ez a kezdeti socket() hívásban megadott protokollmezőtől függ: ha az nem ETH_P_ALL (ami azt jelenti, hogy az összes ethernetkeretet el kell kapni), akkor csak azok a keretek érkeznek meg a szűrőhöz, amelyek itt vannak meghatározva. Például egy ETH_P_IP foglalat esetén így írhatjuk át a szűrőt, hogy gyorsabb és rövidebb legyen:

```
(000)   ld    [26]
(001)   jeq   #0xc0a8090a   jt 4    jf 2
(002)   ld    [30]
(003)   jeq   #0xc0a8090a   jt 4    jf 5
(004)   ret   #68
(005)   ret   #0
```

A szűrő telepítése

Egy LPF telepítése egyszerű művelet: csak annyit kell tenned, hogy létrehozol egy sock_filter szerkezetet, amely a szűrőt tartalmazza és csatolod egy nyitott foglalathoz. A szűrő szerkezete könnyen megkapható, ha a tcpdump -d kapcsolóját -dd-re cseréled. A szűrő C tömbként nyomtatódik ki, amelyet beilleszthetsz a saját kódodba, ahogy a *4. lista* mutatja. Utána a setsockopt() hívással csatolod a szűrőt a foglalathoz.

Egy teljes példa

Cikkünket egy teljes példával zárjuk (lásd az 5. listát a ➲ ftp://ftp.ssc.com/pub/lj/listings/issue86/ címen). Ez teljesen hasonló az első két példához, csak hozzá van adva az LSF-kód és a setsockopt() hívás. A szűrő csak azokat az UDP-csomagokat engedi át, amelyek forrás- vagy célcíme a 192.168.9.10 és az UDP-forráskapu 5000.

Ha ki akarod próbálni ezt a példát, szükséged lesz egy egyszerű módszerre, amellyel tetszőleges UDP-csomagot tudsz előállítani (ilyen a sendip vagy az apsend, megtalálhatók a ➲ http://freshmeat.net/ címen). Valószínűleg az IP-címet is át szeretnéd állítani egy olyanra, amely a helyi hálózatodon előfordul. Ehhez a szűrő kódjában a 0xc0a8090a számot le kell cserélned az IP-cím tizenhatos számrendszerbeli alakjára.

Az utolsó megjegyzés az ethernetkártya akkori állapotára vonatkozik, amikor kilépsz a programból. Mivel az ethernet-állapotjelzőket nem állítottuk alaphelyzetbe, a kártya lehallgató módban marad. A megoldás az, hogy meg kell valósítanod a Control-C (SIGINT) jel kezelőjét, amely az ethernet-állapotjelzőket megelőző értékükre állítja vissza (ezeket közvetlenül azelőtt kell elmentened, hogy a VAGY-ot végrehajtanád az IFF_PROMISC-kal), mielőtt a program befejezi működését.

Összegzés

A csomagok helyi hálózatról történő leszippantása hálózati gondok vagy mérések esetén felbecsülhetetlen értékű ismereteket adhat. Néha a közismert eszközök – például a tcpdump vagy az ethereal – nem illeszkednek pontosan az igényeinkhez. Ilyenkor nagy segítség, ha képesek vagyunk saját szippantót írni. Az LPF-nek köszönhetően ez egyszerűen és hatékonyan megtehető.

Gianluca Insolvibile (g.insolvibile@cpr.it.) a 0.99p14-es rendszermag-változat óta lelkes Linux-rajongó. Jelenleg hálózatokkal és digitális videókutatással, valamint -fejlesztéssel foglalkozik.

Kapcsolódó címek

➲ http://www.linuxpapers.org/
➲ http://www-nrg.ee.lbl.gov/nrg.html címen.
➲ http://www-nrg.ee.lbl.gov/nrg.html