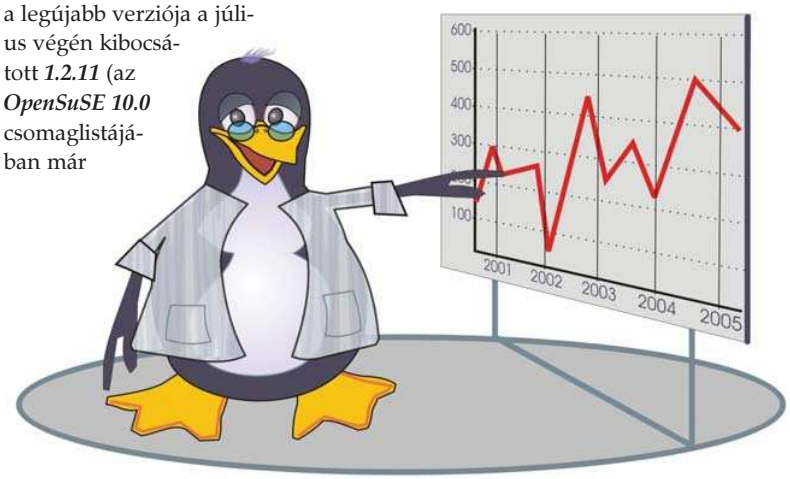
Grafikonok minden mennyiségben – RRDTool (1. rész)

A rendszergazdáknak előbb-utóbb ingere támad arra, hogy a rendszereik minden mérhető jellemzőjét – szeretve dédelgetett rendszerük minden szívdobbanását – naplóba vezessék. Ezek az adatok azonban nem igazán látványosak, s nem látszik bennük vizuálisan a gépeik minden rezdülése. A begyűjtött és megőrzött – akár percre pontos – adatok többnyire feleslegesen foglalják a helyet, mivel hónapok vagy évek távlatában nem igazán gyakori igény pontosan utánanézni a szerver átlagos terhelésének.

Tobias Oetiker keze nyomán 1998 januárjában megjelenő *RRDTool* első verziója rendkívül hiánypótló, s rendkívül kezdetleges program volt. Alapkoncepciója azonban ***zseniálisan egyszerű***: egy-egy időszakról (órák, napok, hetek, stb.) csak a neki megfelelő részletességgel tárol adatokat, s ezeket is körbeforgatja: az adatbázis mérete így állandó marad. Ez a módszer adatvesztéssel jár, ugyanis a körbeforgatás esetén az időszak legrégebbi értékét a legújabb felülírja. Az *RRD* név is ezt a jellegzetességet takarja: ***Round-Robin Database***.
Az *RRDTool* közel egy évtizedes fejlődése remek eszközöket ad a kezünkbe, teljesen belesimul a ***UNIX KISS*** (***Keep It Simple & Stupid***) filozófiába. A legtöbb terjesztés több éve tartalmazza ***rrdtool*** néven, amelynek a legújabb verziója a július végén kibocsátott ***1.2.11*** (az ***OpenSuSE 10.0*** csomaglistájában már benne van). Ha mégsem találjuk meg ezt a programot csomag formájában, akkor a http://www.rrdtool.org oldalról indulva le tudjuk tölteni a forrását, amelyet lefordítva és telepítve már használhatjuk is.
Bármilyen adatokat szeretnénk beletölteni az adatbázisba, tudnunk kell, hogy az *RRDTool* nem képes összegyűjteni azokat. Az *RRDTool* kész adatokat vár bizonyos időközönként, amely lehetnek rendszertelenek, de leggyakrabban rendszeres időközönként kell beletölteni ezeket. Ha tudunk rendszeres adatokat prezentálni, akkor nagyobb kihagyás esetén az *RRDTool* képes interpolálni köztes értékeket, ha így hozzuk létre az adatbázist.
Általában az *RRDTool* működése nem szakad meg, mivel egyszerűségénél fogva nem kell sok hibalehetőségtől tartanunk: a program rendkívül stabil és robusztus működésű. Mivel az ördög nem alszik, érdemes rendszeresen bináris és szöveges (*XML*) mentést készíteni az összes *rrd* állományunkról. A mentést leszámítva a program egyetlen kimeneti módja a grafikonok készítése, amelyek tetszetős és átlátható formába öntik az adatbázis aktuális állapotát.

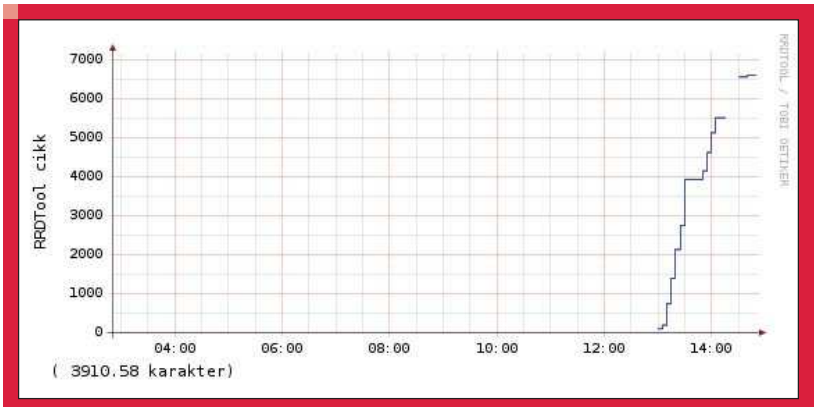


RRD állomány készítése

Ha grafikonon akarjuk ábrázolni valamilyen rendszeresen előálló változó értéket, akkor legelső dolgunk azon adatok meghatározása, amelyeket ábrázolni szeretnénk. Nagyon gondoljuk át, hogy milyen adatokat rögzítünk, mivel – jelenleg még – nincs lehetőségünk egyszerű módszerekkel a működő adatbázisunkat kibővíteni.
Kezdeti ujjgyakorlatként hozzunk létre egy egyszerű *RRD* adatbázist, amely e cikk írásának egyetlen jellemzőjét tárolja, mégpedig a leütött karakterek számát. Ezzel szépen követhetjük az időtengelyen, hogy miképp is állt össze a teljes cikk.
Első látásra kissé átláthatatlan betűkből és számokból álló parancsot kell kiadnunk, hogy ez az adatbázis létrejöjjön:

```
$ rrdtool create rrdcikk.rrd
➥--step 300 DS:karakter:GAUGE:
➥600:0:100000
➥RRA:AVERAGE:0.5:1:1200
```

A paraméterlista tömör és egyszerű, mint ahogy maga a program is. Látható, hogy az első várt paraméter egy tevékenység neve, amely most a létrehozás (`create`). Minden ilyen tevékenységnek más a paraméterlistája, más-más adatokat kell megadnunk.
A létrehozás esetén a második paraméter az adatbázis állomány neve, amelynek általában ***rrd*** a kiterjesztése. Nagyon fontos paraméter a `--step`, amellyel a legkisebb kezelt lépésközt adhatjuk meg másodpercben. Ezt a feladat függvényében kell megválasztanunk, nekem nem volt kedvem percenként megnézni, hogy hány karaktert írtam eddig (mivel aljas módon ***OpenOffice.org*** alatt dolgozom, és nem konzolon): így közel öt percenként elég megadnom új adatot (ez gyakorlatilag egy-egy bekezdés végét jelenti).
A `DS` szöveg a ***Data Source*** rövidítése, ezzel kell bevezetnünk minden egyes adatforrást és annak paramétereit, ezek kettősponttal van elválasztva.
Az első paraméter az adatforrás neve, majd ezt követi az adatforrás típusa. Ez többféle lehet, nekünk most a `GAUGE` kell, amely egyszerűen csak a kapott számot tárolja. Lehetőségünk van még számláló (`COUNTER`) típusra is, amely folyamatosan növekvő számokkal dolgozik (ideális a hálózati forgalom ábrázolására). Innen ismét elágazik a paraméterek feladata, mivel minden egyes adatforrás típusnak saját opciói vannak. A `GAUGE` opciói sorra a `heartbeat`, a minimum és a maximum.
Az első érték (szívdobbanás) másodpercben értendő, és rendszeres adatfeltöltés esetén általában duplája a megadott lépésnek, hiszen nem kell túl nagy kieséstől tartanunk. Ha nem érkezik adat a megadott időtartam alatt, akkor `UNKNOWN` (ismeretlen) kerül érték helyett a következő adat helyére.
Ez kihagyásként jelentkezik a grafikonon, s nem néz ki túl szépen (*1. ábra*). Ha a lépésköz többszöröse ez az idő, s két adatfeltöltés között több lépésköz is kimarad, de még nem telt le egy szívdobbanásnyi idő, akkor nem jelenik meg a fenti kihagyás, a program interpolál az előző és jelenlegi adatfeltöltés értékeiből. Úgy válasszuk meg ezt az időt, hogy a kisebb hibákat elfedje, de a nagyobb hibák kiderüljenek.
A minimum és a maximum érték önmagáért beszél, itt tudjuk megadni az adott adatforrás legkisebb és legnagyobb lehetséges értékét. Ha valamelyiket nem ismerjük, egy U betűt is írhatunk bármelyik helyen.



1. ábra Szakadás a grafikonban

A `DS` blokk után megadhatunk újabb adatforrás definíciókat, egyszerűen csak szóközzel elválasztva. Nekünk jelenleg nem kell több adatforrás, tehát következhet az eltárolt adatok körforgásának meghatározása (***RRA***, ***Round-Robin Archives***).
Az adatok eltárolásának van négy különböző módja: `AVERAGE`, `MIN`, `MAX` és `LAST`. Ezek rendre az adott lépésközben a beírt adatok átlagát, minimumát, maximumát vagy az utolsó értéket tárolják. Gyakori az átlag használata, mivel ez nagyobb léptékű (havi vagy éves) grafikon esetén is szépen ábrázolja az értékeket, s a kisebb idők esetén is jó eredményt ad. Természetesen eltárolhatjuk a többi értéket is, amelyek jól jönnek majd a grafikonkészítés során.
Az eltárolás típusát követi három szám, amelynek az `xff`, a `steps` és a `rows` nevet adta a program készítője. Ezek közül az első paraméter értéke általában 0.5, s a programot az ismeretlen értékek kezelésében befolyásolja. Sokkal fontosabb paraméter a lépésköz és a letárolt sorok száma.
A lépésköz itt nem másodperc, hanem az adatbázis létrehozásakor megadott lépésköz egész többszöröse. Esetünkben ennek értéke 1, így öt percenként tárol egy-egy értéket az adatbázisunk.
A sorok száma önmagáért beszél.
Az eltárolás megadásakor készíthetünk több lépésközt is, amelyeket érdemes a hozzáigazítani a grafikonok rajzolásához. Általában 1 órás, 6 órás, 12 órás, 1 napos, 1 hetes, 1 hónapos, illetve 1 éves intervallumra készítünk grafikont. A grafikonok felbontása gyakorlatilag tetszőleges méretű lehet, de általában képernyőhöz méretezzük, ahol a képpontok száma közelít az adatsorok számához. Egy egész napos grafikon esetén – egy perces lépésközzel – 1440 adatunk áll rendelkezésre, amelyet többnyire 600 vagy 800 képpont széles grafikonon ábrázolunk. Ez azt jelenti, hogy 60 másodpercel lépésköz esetén szükségünk lesz egy

```
RRA:AVERAGE:0.5:1:1440
```

paramétersorra, amely egy perces gyakorisággal 1440 értéket rögzít, pont egy egész napot.
Egy hét adatainak grafikonon való ábrázolásához tíz perces rögzített adatok már szép eredményt adnak, így írhatunk egy

```
RRA:AVERAGE:0.5:10:1008
```

paramétersort is, ez egy teljes hét.
Egy hónap ábrázolásához elég fél óránként egy érték, s így is 1440 sort tudunk rögzíteni:

```
RRA:AVERAGE:0.5:30:1440
```

Az éves értékekhez hat óránként szükséges egy adat, s így rögzíthetünk 1600 sort, amely bőven sok a szép grafikonhoz:

```
RRA:AVERAGE:0.5:360:1600
```

Ha a két utolsó paramétert egymással összeszorozzuk, majd az adatbázis lépésközével is megszorozzuk,

megkapjuk az adott tárolás körbefordulási idejét. Például a legutolsó esetben ez ***360*1600*60***, ami ***34560000*** másodperc, s ez pontosan ***400*** napnak felel meg. Ha egy évnél túl szeretnénk grafikont rajzolni, akkor ahhoz megfelelő RRA értékeket kell készítenünk, különben az adatok elvesznek a körbefordulás miatt.
A példában csak egy ***RRA***-t hoztunk létre, amely az utolsó 100 órát rögzíti. Nem szeretném 100 óránál tovább írni ezt a cikket, így előre láthatólag elég lesz.

Adatok beírása

Az adatbázist azért hoztuk létre, hogy adatokat írjunk be, erre szolgál a frissítés (`update`) tevékenység. Ennek nagyon egyszerű formája van:

```
$ rrdtool update rrdcikk.rdd
➥N:9324
```

Az `N` paraméter annyit tesz, hogy az aktuális idő (***now***) szerint szúrjuk be az adatokat. A program a ***UNIX*** időmeghatározást is ismeri, amely az 1970. január 1-e óta eltelt másodpercek számát jelenti. Ez utóbbira akkor van szükség, ha egy új adatbázisba régebbi adatokat akarunk tölteni (a `'date +%s'` mutatja meg ezt).
Ha több adatot is meg akarunk adni egyszerre, akkor többször meg kell ismételnünk az utolsó paramétert (természetesen ekkor meg kell adnunk időt):

```
$ rrdtool update rrdcikk.rrd
➥1129988409:9452
➥1129988865:9512
➥1129989103:9643
```

Az adatok mentése

Az ***RRDTool*** program régebben szöveges állományba mentette a megadott adatbázis tartalmát, mostanában már ***XML*** a kimenet formátuma. Ezt sokkal egyszerűbb más programhoz illeszteni, illetve leírja a saját szerkezetét.
Az adatok mentésre a `dump` tevékenység szolgál, amely minden esetben a kimenetre írja az elkészült ***XML*** adatfolyamot:

```
$ rrdtool dump rrdcikk.rrd
<!-- Round Robin Database
➥Dump -->
<rrd>
        <version> 0003 </version>
        <step> 300 </step> <!--
        ➥Seconds -->
        <lastupdate> 1129989064
        ➥</lastupdate> <!--
        ➥2005-10-22 15:51:04
        ➥CEST -->
[...]
```

Az adatok visszatöltése

A kiexportált ***XML*** állományból `restore` tevékenységgel tudunk ***rrd*** formátumú bináris adatbázist készíteni. Ehhez a következő parancsot tudjuk használni:

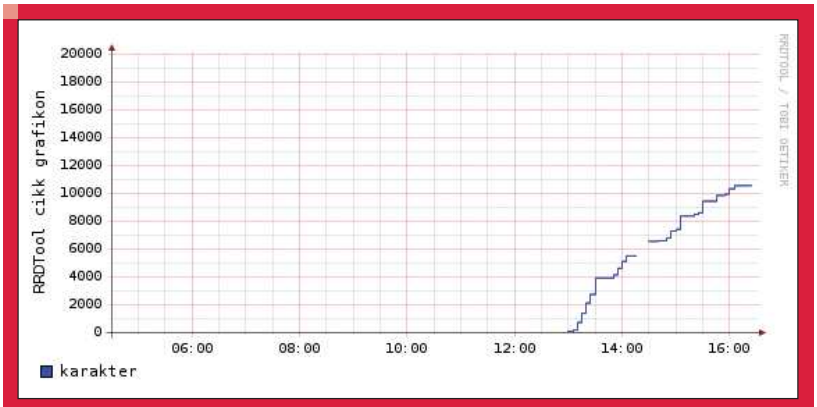
```
$ rrdtool restore rrdcikk.xml
➥rrdcikk2.rrd
```

Kénytelenek vagyunk más nevet megadni a bináris formának, mivel létező adatbázist nem ír felül a program.

Grafikon készítése

A program egyik fő feladata a grafikonok készítése, így kellően összetett és bonyolult első látásra a kiadandó parancs (*2. ábra*):

```
$ rrdtool graph cikkRRD.png
➥--imgformat PNG --width 464
```



2. ábra Az elkészült grafikon

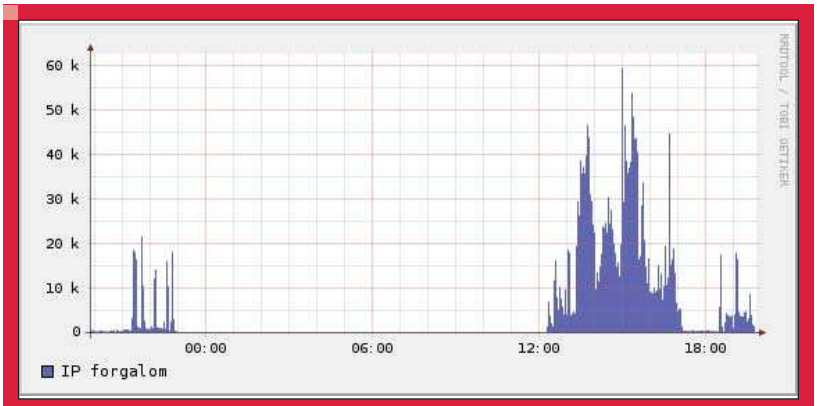
```
➥--height 200 --end now
➥--start end-12hours
➥--vertical-label "RRDTool
➥cikk grafikon" --lower-limit
➥0 --units-exponent 0 --lazy
➥--color "SHADEA#ffffff"
➥--color "SHADEB#ffffff"
➥--color "BACK#ffffff"
➥"DEF:karakter=rrdcikk.rrd:
➥karakter:AVERAGE"
➥"LINE1:karakter#0000FF:
➥karakter"
```

Néhány példa után már nem fog olyan bonyolultnak és átláthatatlannak látszani a fenti parancs, azonban vannak olyan mély rejtelmei az ***RRDTool*** programnak, amelyek még a profikat is zavarba hozhatják.
A legbiztosabb, hogy meg kell adnunk a rajzolás (`graph`) tevékenységet és ezt követően annak a grafikus állománynak a nevét, amely a grafikont fogja tartalmazni. Ezek után rendet kell vágnunk a „kusza” parancsok között.
Kezdjük az ismerkedést olyan egyszerű paraméterekkel, mint amilyen a grafikon mérete és a típusa. A `--width` és a `--height` után megadott szám lesz a grafikon mérete, amely nem tartalmazza a tengely feliratait, a díszítéseket és a magyarázó szövegeket. Ezen viselkedése egy kicsit kiszámíthatatlanná teszi a program által készített képek végső méretét, ezért a készítés során a kimeneten láthatjuk a kép tényleges méretét (a fenti esetben ez ***561x268***).
A `--imgformat` alapértelmezett értéke a `PNG`; de ezen kívül a program képes `SVG`, `EPS` vagy `PDF` állományt is készíteni.

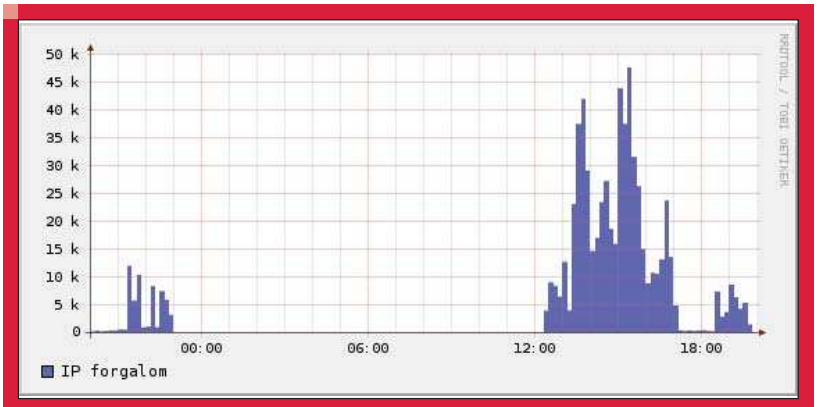
A `--end`, a `--start` és a `--step` a grafikon léptékét határozza meg. Ezek közül a `--end` a grafikon végének időpontját határozza meg, amely általában a now (most) szó. A `--start` ebből következően grafikon kezdetének időpontja lesz, s a végponthoz képest szoktuk meghatározni. Ez gyakorlatilag egy `end±n` forma lesz, ahol az n egy úgynevezett időeltolás (***time offset***). Ezt simán angol nyelven tudjuk megfogalmazni, például az

```
end-2years3months14days14hours
➥56minutes23seconds
```

jelenti, hogy a grafikon kezdőpontja végpontjához képest ***két*** évvel, ***három*** hónappal, ***14*** nappal, ***14*** órával, ***56*** perccel és ***23*** másodperccel korábbi időpontban van. A `--step` paraméter a grafikon felbontását határozza meg, amelyet nem szükséges megadnunk, mivel a program saját maga meghatározza a grafikon felbontásától és a megadott kezdő- és végponttól függően.
Az időtengelyt már rendbe tettük, azonban nem foglalkoztunk még a függőleges iránnyal, amelyen az értékeket ábrázoljuk. A program alapértelmezésben automatikusan meghatározza a szükséges határokat, azonban vannak esetek, amikor jobb belenyúlni ebbe a folyamatba. A legalapvetőbb beállítási lehetőség a minimális és a maximális ábrázolt értékek közvetlen megadása az `--upper-limit` illetve a `--lower-limit` paraméterrel. Ezen túl a program saját automatizmusát egy alternatív módszerre tudjuk cserélni a `--alt-autoscale` paraméterrel, amely akkor hatásos igazán, ha nagy érték körül van kis változás (a program kézikönyve a `260*0.001sin(x)` függvényre hivatkozik).
Ha a két tengelyt megfelelően beállítottuk, akkor beállíthatjuk a grafikon rácsozatát, amely megvezeti szemünket az értékek követésénél. A dokumentáció szerint *„**quite complex**”*, de szerencsére csak viccel a program írója, bár el kell ismerni kicsit ködösen fogalmaz. Nos, a lényeg, hogy az `--x-grid` paraméter után meg kell adni a rácsozatot `MINUTE:30` formában, amely azt takarja, hogy félóránként lesz egy vonal a grafikonon. Ezt egy kettőspont után a fő rácsozat követi, amely például `HOUR:4` esetén minden negyedik órában lesz egy vastagabb vonal a grafikonon.
Ezt követi (szintén kettőspont után) a feliratok pozíciója, amely például lehet `DAY:1`, s ekkor minden eltelt nap alatt lesz egy-egy felirat. A felirat pozícióját és egy formátumszöveget



3. ábra Hálózati IP forgalom



4. ábra 10 perces lépésköz – nem túl szép, de pontos

kell végül megadni; így a példánk a következőképpen néz ki:

```
--x-grid MINUTE:30:HOUR:4:DAY:
➥1:0:%X
```

A függőleges tengely paraméterezése más sokkal egyszerűbb, hiszen csak meg kell adni azt a két számot, amelyek a rácsvonalak és a fővonalak sűrűségét meghatározzák:

```
--y-grid 40:10
```

Itt a program 40 egységenként tesz egy vonalat, s 400 egységenként egy fő rácsvonalat. Ezzel a függőleges tengely paraméterezése nincs teljesen kész, hiszen lehetőségünk van logaritmikus ábrázolásra is, amelyet különösen nagy változások esetén érdemes használni:

```
--logarithmic
```

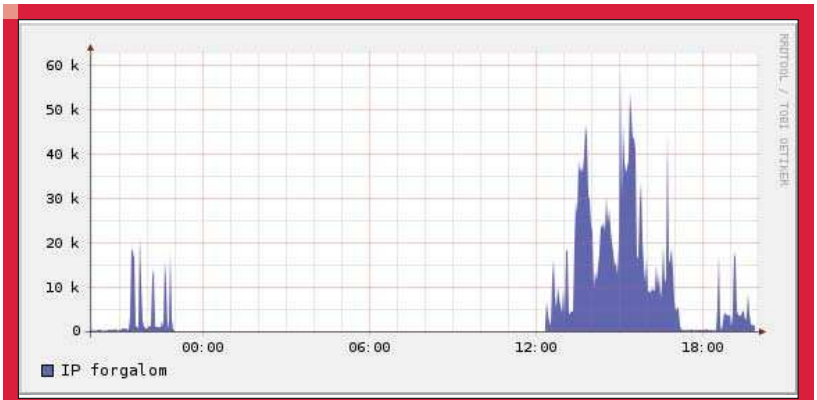
A grafikonra tehetünk címkéket, amelyeket a maradék két szabad helyre tudunk tenni: a grafikon fölé vagy bal oldalára. A jobb oldalt ugyanis a program készítője lefoglalta magának, a grafikon alatt pedig a jelmagyarázat kapott helyet. A főcímet a `--title` paraméterrel tudjuk beállítani, a bal oldali szöveget a `--vertical-label` paraméter segítségével. Mindkét esetben idézőjelek között érdemes megadni a kívánt szöveget.
Módosíthatjuk a grafikon színeit, ha erre támad kedvünk, így például zökkenőmentesen beilleszthetjük egy weboldal arculatába a kész képeket. Ehhez a

```
--color KOMPONENS#RRGGBB[AA]
```

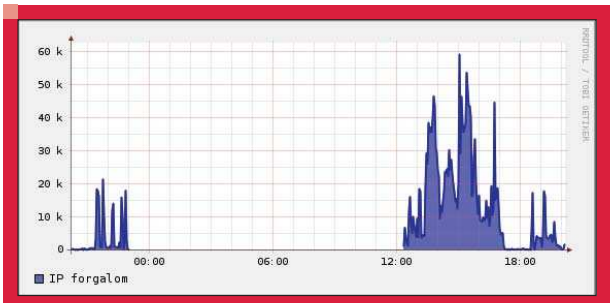
használható, ahol a kettőskereszt után megadott hexadecimális számok a szín vörös, zöld és kék összevevőjét jelölik, utána megadhatjuk az adott komponens átlátszóságát is, ha szeretnénk ilyet. A komponens lehet a `BACK`, mint háttér; `CANVAS`, mint a grafikon háttere; `SHADEA` és `SHADEB`, mint a grafikon kerete; `GRID` és `MGRID`, mint a rácsozat; `FONT`, mint a betűk; `AXIS`, mint a koordináta rendszer; `FRAME`, mint a keret; s végül `ARROW`, mint a koordináta tengelyek lezáró nyilai. Ezzel meg is volnánk a közös paraméterekkel, így jöhet a rázósabb része a grafikon definíciónak, amellyel meghatározzuk, hogy milyen adatsorokból milyen grafikont szeretnénk rajzolni. Ehhez `DEF` kezdetű sorokat kell megadnunk, amelyet minimálisan három megadandó paraméter követ: az érték elnevezése, az ***rrd*** állomány neve, végül az adatforrás neve és adattárolás a módja zárja a sort (természetesen minden egyes paramétert kettőspont választ el). Ezeken a paramétereken túl megadhatunk más léptéket, kezdő- illetve záróidőt; ezen paramétereket ritkán szoktuk használni, a program automatikusan beállítja helyettünk.
Ha több ***rrd*** állományunk van, és ezekben több adatforrásunk, akkor is tudunk egy grafikonra adatokat gyűjteni, ugyanis minden DEF sornak egyedi nevet kell adnunk, s meg kell neveznünk az adatbázist az adatforrással együtt. Az eddigi sok munkával még nem rajzoltunk semmit, egyszerűen csak meghatároztuk azon adatsorokat, amelyeket ábrázolni szeretnénk... a rajzolás még teljes egészében hátravan.
A rajzoláshoz érdemes keresni olyan ***rrd*** állományokat, amelyek sok összefüggő adatot tartalmaznak. Igazi állatorvosi ló lehet például az ***Ntop rrd*** modulja által készített hálózati adatbázis, amely rengeteg információt rögzít a hálózati forgalmunkról.
Nézzünk egy egyszerű példát, ahol is szeretnénk tudni a teljes ***IP*** forgalom alakulását egy napra visszamenőleg, mégpedig kitöltött területtel ábrázolva (*3. ábra*):

```
DEF:ip0=ipBytes.rrd:counter:
➥AVERAGE AREA:ip0'#6666FF':'
➥IP forgalom'
```

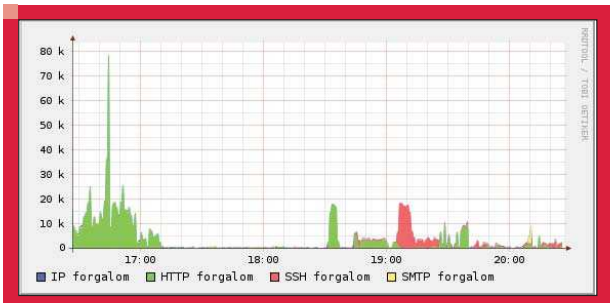
A kitöltött területet az `AREA` vezeti be, amelyet az adatsor neve, a színkódja, s végül a megnevezése követ. A színkód az őt megelőző kettőskereszt miatt van idézőjelben, mivel a # jel utáni szöveget a ***UNIX*** parancsértelmezők egy része figyelmen kívül hagyja, mint megjegyzést (a ***BASH*** csak akkor, ha a # jel előtt egy szóköz is áll).



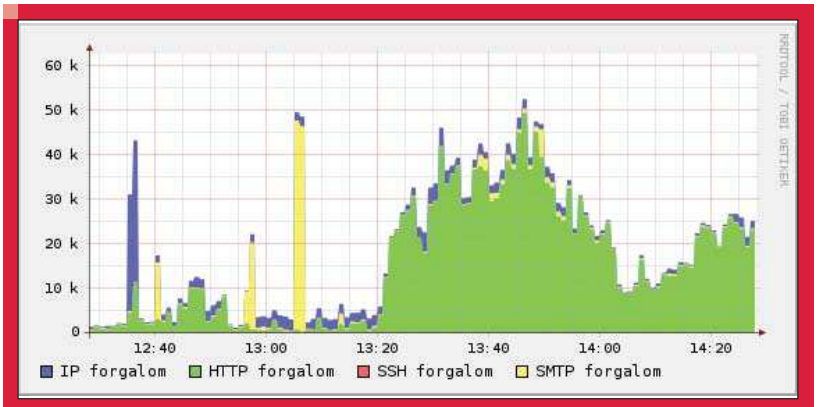
5. ábra Lejtős átlagolás – szép, de nem pontos



6. ábra Beburkolva



7. ábra Halmozott területek I.



8. ábra Halmozott területek II.

Látszik, hogy nem valami szép a grafikon, ha az értékek nagyon ingadoznak, érdemes egy picit simítani a képen, amelyhez két lehetőségünk van: vagy a `--step` paramétert hívjuk segítségül (*4. ábra*) egy nagyobb (például tíz perces lépésközzel) vagy bekapcsoljuk az úgynevezett lejtő-módot (`--slope-mode`) (*5. ábra*). Még szebb eredményt érhetünk el, ha egy vastagabb vonallal beburkoljuk a görbét (*6. ábra*):

```
DEF:ip0=ipBytes.rrd:counter:
➥AVERAGE AREA:ip0'#6666FF':'
➥IP forgalom' LINE2:ip0'
➥#0000AA':''
```

Érdemes néhány másik jellemzőt is felrajzolni a grafikonra, például az *IP* forgalom tételes megoszlását: szeretnénk látni, hogy mennyi ebből a *HTTP*, az *SMTP* és például az *SSH*. Ehhez csak annyit kell tennünk, hogy más-más *rrd* állományokból gyűjtünk össze adatforrásokat:

```
DEF:ip0=ipBytes.rrd:counter:
➥AVERAGE DEF:http0=
➥IP_HTTPBytes.rrd:counter:
➥AVERAGE
DEF:ssh0=IP_SSHBytes.rrd:
➥counter:AVERAGE DEF:mail0=
➥IP_MailBytes.rrd:counter:
➥AVERAGE
```

Majd vonalakkal ábrázoljuk az egyes mennyiségeket:

```
AREA:ip0'#6666FF':'IP forgalom'
➥LINE:http0'#66FF66':'HTTP
➥forgalom' LINE:ssh0'#FF6666'
➥:'SSH forgalom' LINE:mail0'
➥#FFFF66':'SMTP forgalom'
```

Ez olyannyira nem lett szép, hogy nem is érdemes megmutatnom, a vonalak ugyanis egymást keresztezik, átfedik... sokkal szebb eredményt tudunk elérni, ha nem vonalakat, hanem területet használunk, s ezek nem a függőleges tengely legalsó pontjáról indulnak, hanem egymásra halmozzuk őket, s így a legfelső pontjuk pont az összes halmozott érték összegénél fog járni:

```
AREA:ip0'#6666FF':'IP forgalom'
AREA:http0'#66FF66':'HTTP
➥forgalom'
AREA:ssh0'#FF6666':'SSH
➥forgalom':STACK
AREA:mail0'#FFFF66':'
➥SMTP forgalom':STACK
```

Az összegzés mindig az előző értékhez képest működik, így a teljes IP forgalomhoz nem adjuk hozzá az összetevőit. Egy picit belenéztem a grafikonba, s találtam olyan pontot, ahol a különféle adatforgalmak szépen látszanak (*7. ábra; 8. ábra*).
A második (befejező) rész az ***RRDTool*** olyan extra tulajdonságait taglalja, mint az értékek és szövegek írása a grafikon alá, illetve az adatsorokkal való matematikai és logikai műveletek.

Auth Gábor
(auth.gabor@enaplo.hu)

Egy pécsi középiskolában informatikát és programozást oktat. Tíz éve botlott először a UNIX rendszerekbe, 7 év Linux használat után kapta el a FreeBSD lázat, amiből máig nem tudott kigyógyulni.

KAPCSOLÓDÓ CÍMEK

Az RRDTool honlapja
➲ http://www.rrdtool.org

A cikkben említett példák forrása
➲ http://user.enaplo.hu/~auth.gabor/rrd/

Való életből vett példa
➲ http://www.enaplo.hu/index.jsp?page=visitor.loadStat