UVOD

UVOD

Tema ovog seminara je bila optimizacija Programa za računanje reputacije koji je dio diplomskog rada Dejdar Tomislava. Diplomski rad u pdf formatu je dio distribucije ovog programa i ako netko ima zadatak raditi išta na ovom programu, trebao bi pročitati taj rad.

Program je dosta restrukturiran, optimiziran i dodane su mu opcije za lakše izvođenje, za iscrtavanje grafova i kontrolu nekih parametara.

Neke od glavnih promjena, svaka će biti kasnije detaljnije opisana:

- izbačena 3 različite ulazne točke u program i sve svedeno na jednu
- dodana konfiguracijska datoteka za lakšu kontrolu i izmjenu parametara
- optimizacija nekih dijelova, ubrzanje varira o parametrima ali zna bit i do 50%
- dodani grafovi i to za više ASova odjednom radi lakše usporedbe
- popravljene neke greške
- refaktoriranje
- komentiranje nekih dijelova

U dokumentaciji će se koristiti 2 varijable za opisivanje putanji datoteka:

- $ROOT – glavni direktorij u koji se otpakira zip datoteka (seminar_rep_2_0)
- $PROJECT – poddirektorij gdje se nalazi projekt, $ROOT/project_rep_2_0

KAKO POKRENUTI PROGRAM

Potrebni ulazni podaci:

- konfiguracijska datoteka *config.ini*
- pretparsirana RIB tablica, ime i put do tablice se definiraju u konfiguracijskoj datoteci
- update dump datoteke, trebaju se nalaziti u $PROJECT/Up direktoriju

Svi potrebni ulazni podaci za testiranje dolaze sa programom. RIB tablica *rib.20110113.0000.parsed* se nalazi u $PROJECT/Rib direktoriju. Update dump datoteke se nalaze u $PROJECT /Up direktoriju i obuhvaćaju vremenski interval od 13.01.2012. u 00:00 sati do 14.01.2012. u 15:45 sati. Konfiguracijska datoteka je već podešena za rad sa tim testnim podacima.

Program se pokreće naredbom **>PYTHON START.PY**

U direktoriju $PROJECT/alt se nalaze alternativni podatci za pokretanje programa: Rib tablica, pripadajuće update datoteke i konfiguracijska datoteka sa namještenim vremenima za taj slučaj.

Treba biti oprezan u slučaju da se dodaju još neke update datoteke uz postojeće, recimo kopiraju se datoteke iz $PROJECT/alt direktorija a ne izbrišu se već postojeće iznutra, jer program čita sve datoteke po redu tako da ako se dodaju update datoteke sa ranijim datumom doći će do krivih rezultata, a ako se ne zada vremensko ograničenje u konfiguracijskoj datoteci program će nastaviti parsirati sve datoteke u tom direktoriju.

RIB TABLICE

Za pretparsiranje datoteka s RIB tablicama usmjernika napravljen je parser u programskom jeziku Perl pod nazivom Zebra Dump Parser. On se koristi za pretparsiranje podataka iz RIB tablica usmjernika u oblik pogodan za daljnju obradu programskim jezikom Python. Sam postupak pomaže standardizaciji sustava s obzirom na ulazne podatke. Naime ulazni podaci, pogotovo RIB tablice mogu doći u više formata što može stvoriti određene poteškoće prilikom njihove obrade. Zato RIB pretparser ima zadaću pretparsiranja RIB dump datoteka u jednostavniji format pogodan za daljnju obradu reputacijskom sustavom. RIB dump datoteke sadrže ispise cjelokupne RIB tablice usmjernika. Oni su zapisani u MRT formatu (engl. Multithreaded Routing Toolkit) napravljenom prvenstveno za pohranu usmjerničkih informacija. Datoteke u MRT formatu binarne su datoteke te postoji više podformata MRT-a. Danas je najčešće korišteni format za BGP dump datoteke "IPV4 MRT RIB table dump v2". Takve binarne datoteke u MRT formatu dodatno su komprimirane, najčešće u gzip ili bzip2 sustavu komprimiranja te ih je prvo potrebno dekomprimirati.

UPDATE DATOTEKE

Update datoteke koje se koriste u programu su binarne UPDATE datoteka u koje kolektori pohranjuju sve UPDATE poruke dobivene od svojih susjeda u određenom vremenskom intervalu. One se pohranjuju najčešće u pravilnim vremenskim razmacima od 15 ili 20 minuta. Imena datoteka su tako složena da se iz njih može lagano iščitati vrijeme nastanka, ime je početni trenutak od kojeg se pohranjuju UPDATE poruke. Iz imena sljedeće datoteke saznaje se početni trenutak sljedećeg prozora prikupljanja i pohrane UPDATE poruka što se poklapa sa zaključnim trenutkom prethodnog prozora.

Za potrebe ovog rada korišteni su podaci s kolektora pohranjeni na web stranici održavanoj od strane Oregonskog sveučilišta http://www.routeviews.org/

ANALIZA PROGRAMA

ANALIZA KODA

Program se sastoji od 3 glavne datoteke:

- *start.py*
- *analyzer.py*
- *core.py*

start.py je glavna točka ulaza u program, u njemu se učitavaju parametri iz konfiguracijske datoteke, inicijalizira glavna klasa *Analyzer*, kojoj se predaju konfiguracijski parametri i pozivaju funkcije *analyzeLinkBindings*() i/ili *analyzePrefBindings*(), u ovisnosti u konfiguracijskim parametrima.

analyzer.py je modul u kojem se nalazi definicija klase *Analyzer*, te neke konstante potrebne za njezin rad (imena izlaznih datoteka i definicija direktorija gdje se nalaze update poruke).
Klasa *Analyzer* ima slijedeće metode:

- *init*() metoda koja služi za inicijalizaciju samog objekta i pamćenje parametara koji se pošalju
- *analyzeLinkBindings*() koja pokreće analizu podataka metodom postojanosti veza između autonomnih sustava
- *analyzePrefBindings*() koja pokreće analizu podataka metodom povezanosti prefiks – izvorišni autonomni sustav
- *drawGraph*() koja generira png datoteku koja sadrži graf u kojem su prikazani zadani podaci

core.py je modul u kojem se nalaze klase koje obavljaju svo računanje i analizu podataka, te još neke pomoćne klase koje se koriste za organizaciju podataka.
Dvije glavne klase su:

- *PrefixPath* koja se koristi u *analyzeLinkBindings*()
- *PrefixAS0Binding* koja se koristi u *analyzePrefBindings*()

Pomoćne klase su:

- *Prefix* – služi za pamćenje prefiksa i njegove duljine
- *AS* – služi za pamćenje broja autonomnog sustava
- *AsPath* – služi za baratanje sa path dijelom podataka (dodavanje na kraj, dodavanje na početak, čitanje izvorišnog AS-a, micanje duplikata ...)
- *AsPrefix* – služi za pohranu i manipulaciju podataka o vezi prefiks – autonomni sustav
- *AsPrefixRep* – služi za računanje reputacije
- *Print* – statična klasa koja služi za kontroliranje ispisa u konzolu u ovisnosti o verbose parametru

TIJEK IZVOĐENJA PROGRAMA

Za razliku od prije, sada se pokreće *start.py*. U njemu se učitavaju konfiguracijski parametri iz datoteke *config.ini.* Ti parametri su sljedeći:

- *time_window* - veličina vremenskog prozora u kojem će se računati reputacija (u minutama)
- *time_limit* – maksimalna duljina intervala u kojem će se računat reputacija (u minutama), ako se stavi 0 onda će se računati za sve update datoteke koje postoje u $PROJECT/Up direktoriju
- *time_start* – vrijeme početka računanja
- *rib* – put i ime pretparsirane RIB tablice
- *text_output* – hoće li program ispisivati izlazne tekstualne datoteke (0 ili 1)
- *verbose* – koliko detaljno će program ispisivati u konzolu (0 ili 1)
- *alpha*, *gama*, *delta* – parametri koji se koriste u računanju reputacije
- *selected_as* – autonomni sustavi za koje se želi da budu iscrtani u grafu
- *graph_x*, *graph_y* – veličina slike u kojoj se nalazi graf, u pikselima
- *analysis* – metoda analize, može biti jedna ili obje (pref, link)
- *profiling* – hoće li program obaviti profiliranje brzine (0 ili 1)

Nakon učitavanja se inicijalizira objekt *Analyzer* u kojeg se predaju većina pročitanih parametara. Nakon toga se pokreće funkcija *start*() u kojoj se pozivaju metode koje pokreću samu analizu, a ako je izabrano profiliranje onda se profiliranje vrši nad funkcijom *start*().

Nakon što se pokrene odabrana metoda analize program u principu radi iste stvari, parsira RIB tablicu i kreće računati reputaciju tako da učitava jednu po jednu update datoteku, parsira update poruke iz nje, i kad dođe do granice vremenskog prozora izračuna reputaciju za taj prozor. Nakon toga nastavlja dalje parsirati update poruke dok ne dođe do zadnje update datoteke ili ne dođe do zadanog vremenskog ograničenja (*time_limit*). Na kraju obje metode se poziva metoda *drawGraph* koja generira graf na temelju izračunatih reputacija, koje su se pamtile prilikom računanja pojedinog prozora. Nakon što se obavila jedna analiza program će stati, osim ako nisu bile izabrane obje metode, u kojem slučaju će se pokrenuti druga metoda analize.

U *core.py* postoji konstanta DEBUG koja ako se stavi na vrijednost 1 učitavanje RIB tablice će se prekinuti nakon 100,000 zapisa tako da se ubrza izvođenje tog dijela programa da se može lakše orijentirati na modifikaciju drugih dijelova.

PROMJENE

Dodan je *start.py* modul, koji je sada glavni modul za pokretanje programa. U njemu je dodano proširenje path varijable u runtimeu, tako da se biblioteke iz $PROJECT/lib direktorija mogu normalno korisiti. Na taj način se libovi ne trebaju posebno instaliravati, i ne treba se paziti na njihove verzije, što čini program mobilnijim.

Moduli *pref_bindings.py*, *link_bindings_15.py* i *link_bindings_4.py* su sada objedinjeni u jedan modul *analyzer.py*.

Na kraju programa se ispisuje ukupno vrijeme izvođenja programa (u minutama).

Izbačeno prekomjerno korištenje '\n' znakova kod ispisa u konzolu.

Kod učitavanja RIB tablice (u *core.py*) kod obadvije klase je dodan kod za lakše debugiranje (limit na broj učitanih redaka tako da se ubrza vrijeme izođenja) .

Izbačeni neki nepoznati ispis, jedna od metoda je ispisivala nešto u *'statistics'* datoteku.

Neki višelinijski komentari su formatirani, jer prije su neke linije bile predugačke (preko 1000 znakova), pa ih je bilo teško čitati.

Dodano formatiranje vremena za ljepši ispis na grafovima.

Dodana metoda *drawGraph()* u *analyzer.py* koja iscrtava grafove koristeći *pychart* lib (koji koristi *ghostscript* lib, na linuxu je već preinstaliran a za windowse se treba posebno skinuti)

Program je u svojoj prvoj verziji imao samo tekstualni ispis, koji znao bit i suludo velik (nekad i preko 3Gb!) što je dodatno usporavalo izvođenje programa i punilo hard disk. Sad je dodan parametar da se izbjegne tekstualni ispis i ostane samo grafički, što dodatno ubrzava rad programa.

Parametri *alfa*, *gama* i *delta* prebačeni u *config.ini*, *beta* se nije ni koristio za ništa pa je izbrisan.

Optimizacija

Kod analize koda (sa cProfile metodom) sam ustanovio da se veliki dio vremena kod učitavanja troši na početku na učitavanje dumpa RIB tablice. Pošto sama Rib tablica ima oko 50 Mb teksta, probao sam različite načine učitavanje (prvo učitat cijeli fajl pa ga onda ić parsirat, buffered učitavanje...) međutim ispostavilo se da se nemože uopće ubrzat, to je 50Mb tekst fajla koji se mora učitat i proparsirat liniju po liniju, i to traje uvijek isto. Međutim primjetio sam da ispisivanje trenutnog prefiksa koji se parsira troši oko 40% vremena, pa sam grupirao ispis da se ne ispisuju svi prefiksi nego samo jedna linija za sve koji imaju prvi oktet isti (1.*.*.*) što je ubrzalo učitavanje, a još uvijek ima neki output da se vidi da program nije stao.

Što se tiće parsiranja UPDATE poruke i računanja reputacije u svakom prozoru, tu sam probao napraviti neke optimizacije sa premještanjem for petlji da se netreba prolaziti recimo prvo kroz listu reputacija, pa onda kroz listu inkrementa raputacija pa opet kroz listu reputacija, ali ništa od toga nije rezultiralo ikakvim ubrzanjem. Jedna optimizacija koja je pridonjela ubrzanju što se tiče brzine (u

nekim slučajevima i do 50%), i velikom uštedom što se tiče prostora na disku, je što sam dodao parametar koji određuje hoće li se ispisivati i tekstualni ispis, ili da se samo graf iscrta.

Probao sam prodrijeti dublje u organizaciju podataka i samo računanje, ali se pokazalo kao prekomplicirano, jer kod nije napisan da bude „user friendly“ (pogledaj poglavlje KRITIKA ORIGINALNOG RADA) tako da su sva imena varijabli skraćenice koje su ponekad preslična jedna drugoj, nigdje nema komentara šta se radi u kojem dijelu, i cijeli kod je prepun lošeg formatiranja i na nekim mjestima ima krivog formatiranja (mješanja space i tabova). Tako da je moje mišljenje da bi se taj cijeli kod trebao refaktorirati i detaljno proučiti i komentirati, i mislim da bi to mogla bit cijela tema za jedan seminar.

KRITIKE ORIGINALNOG RADA

PREVIŠE ULAZNIH TOČAKA

Program se sastojao od 3 glavnu ulazne točke (entry pointa):
link_bindings_15.py
link_bindings_4.py
pref_bindings.py

link_bindings_15.py i *link_bindings_4.py* su se razlikovali samo u 2 linije koda, i to u linijama gdje se postavlja veličina vremenskog prozora za računanje i u definiciji imena direktorija gdje su se zapisivali tekstualni ispisi.

pref_bindings.py je bio preko 90% isti kao i gore navedeni moduli i u principu se razlikovao samo u jednom objektu koji se inicijalizirao (u ovom slučaju PrefixAS0Binding, dok je u gore navedenim bio *PrefixPath*) i definiciji izlaznih datoteka

MJENJANJE RADNOG DIREKTORIJA PROGRAMA

Još jedna greška je bila mijenjanje radnog direktorija programa umjesto samo pokazivanje na poddirektorij:

```
dircur = os.getcwd()
os.chdir(dircur + "/UP")            #directory containing UPDATE dumps
file_list = glob.glob("*.*")
```

umjesto

```
file_list = glob.glob( UPDATES_DIR + "/*.*")
```

KRIVO FORMATIRANJE

U metodi AsPath. MakePath() je bilo totalno krivo formatiranje, miješanje razmaka i tabova, tako da su neki editori krivo prikazali identaciju koda. U nekom drugom jeziku to ne bi predstavljao toliko veliki problem ali u Pythonu je identacija zapravo ključan dio sintakse koji određuje redoslijed izvršavanja naredbi tako da se izgubio smisao koda.
Originalni kod:

```
____        for seg in data.segments:
________          if seg.type == bgp.AS_SET:

                        #print "\nSET\n"
______________                  self.path.append('{')
                        for elem in seg.path:
                              self.path.append(As(int(elem)))
______________                self.path.append('}')

________          elif seg.type == bgp.AS_SEQUENCE:

                        #print "SEQ"
                        for elem in seg.path:
                              self.path.append(As(int(elem)))

________          else:
                        #print "\nELSE_SET\n"
______________                  self.path.append('{')
                        for elem in seg.path:
                              self.path.append(As(int(elem)))
______________                self.path.append('}')
```

Ispravljeni kod:

```
            for seg in data.segments:
                  if seg.type == bgp.AS_SET:
                        self.path.append('{')
                        for elem in seg.path:
                              self.path.append(As(int(elem)))
                        self.path.append('}')

                  elif seg.type == bgp.AS_SEQUENCE:
                        for elem in seg.path:
                              self.path.append(As(int(elem)))

                  else:
                        self.path.append('{')
                        for elem in seg.path:
                              self.path.append(As(int(elem)))
                        self.path.append('}')
```

Vidi se da se iz originalnog koda u prvom if bloku može zaključiti da se prvo u self.path doda '{', i onda se prođe kroz sve elemente seg.path i za svakog se doda taj elemntu u self.path i '}', što je krivo.

NEČITKO KODIRANJE

Također u metodi AsPath.MakePath() se vidi jedan primjer lošeg kodiranja:

Prođi kroz sve elemente u dana.segments
    ako je tip = bgp.AS_SET napravi A
    ili ako je tip = bgp.AS_SEQUENCE napravi B
    inače napravi A

Iako ovaj kod nije kriv, u smislu da će ispravno raditi, jednostavnije i čitkije bi bilo da je samo jedna provjera ako je tip = bgp.AS_SEQUENCE napravi B, inače napravi A. Ovaj primjer nažalost nije jedini takav primjer u cijelom kodu.

NEPOTREBNA KLASA

```
class As:                       #base AS class

      #constructor takes integer AS number
      def __init__(self, AS):
            self.AS = AS

      #returns integer AS number
      def GetIntegerAs(self):
            return self.AS

      #returns string AS number
      def GetStringAs(self):
            return str(self.AS)

      #sets AS number, takes integer AS number as argument

      def SetAs(self, AS):
            self.AS = AS
```

Klasa As je totalno nepotrebna , jer ne radi ništa što ne bi bilo brže i čitljivije nego da se umjesto nje koristi najobičniji broj, a kad se treba broj prebacit u string da se samo napravi str(broj). Prilikom profiliranja sam utvrdio da je GetIntegerAS() 8. po redu funkcija koja troši najviše vremena, na njezine pozive se potroši oko 2% ukupnog vremena izvođenja programa. Iako to nije veliki postotak potrošenog vremena, opet pokazuje na to koliko je krivo ovakvo kodiranje, plus što jako umanjuje čitljivost.

KRIVA METODA

Kad se učitava RIB tablica, konkretno AS_PATH dio, postoji metoda koja bi trebala micat duplikate iz te liste međutim ta metoda to ne radi, provjeravaju samo ako su duplikati jedan iza drugoga u listi, a ne ako su igdje drugdje.

Kriva metoda:

```
def RemoveDouble(self):       #removes double ASes in AS Path

      path_temp = []
      as_temp = 0

      for elem in self.path:

            if elem.GetIntegerAs() != as_temp:

                  path_temp.append(elem)
                  as_temp = elem.GetIntegerAs()

      self.path = path_temp
```

Popravljena funkcija:

```
def RemoveDouble(self):
      path_temp = []
      as_temp_list = []

      for elem in self.path:
            if elem.GetIntegerAs() not in as_temp_list:
                  as_temp_list.append( elem.GetIntegerAs() )
                  path_temp.append(elem)

      self.path = path_temp
```

PREDUGAČKE LINIJE KOMENTARA

U nekim komentarima zna biti previše znakova u jednoj liniji, što čini kod jako nečitkim. Ovo je najduža jedna linija komentara u cijelom kodu:

```
Because totalTime counts time the prefix HAD been active, it doesn't
include time from last activation nor currently active prefixes. So, if the
prefix was activated just once and not deactivated it can be 0. It is
increased every time prefix becomes inactive or at the end of the
observation window. If the observation window finishes while the prefix is
still being in active state, totalTime is increased by the time from last
activation and timeOfActivation is set to the time of begining of next
window. In that case the repetition counter is set to 1 to represent the
first activation of the prefix in next window.The listOfRouters contains
list of ip addresses of routers in string format because it's usually very
small (it contains just a few peering routers). I rather use IPes of
peering router instead of neighboring ASes because same AS can have more
peering routers and some peers can also be in our own AS. When the
listOfRouters is empty, the prefix is INACTIVE. When it's NOT empty, the
prefix is ACTIVE. When the last router announcing prefix is removed from
the list, the prefix becomes inactive and totalTime is increased. When a
router is added in empty list, the prefix becomes active, timeOfActivation
is set and repetition counter is increased.
```

1269 znakova!

PREMALO KOMENTARA

Iako nekih komentara ima, u gornjem primjeru vidimo da su neki i surealno dugi, komentara na pravim mjestima nema. Ima dosta kompleksnih metodi koje nemaju nikakve korisne komentirane.

PRESLIČNA IMENA KLASA

Ovo su imena klasa u classes.py (sada *core.py*):

- *Prefix*
- *As*
- *AsPath*
- *PrefixPath*
- *PrefixAS0Binding*
- *ASPrefix*
- *AsPrefixRep*

Iz toga se ne može ništa korisno zaključiti, recimo *AsPath* je pomoćna klasa koja u sebi drži podatke o path dijelu podataka iz RIB tablice, a *PrefixPath* je glavna klasa koja računa reputaciju metodom postojanosti veza između autonomnih sustava (i nema nikakve veze sa *Prefix* klasom)

NEUKLONJEN DEBUG ISPIS

Postoje i ovakve stvari, definicija na razini modula varijable *f9Name*:

```
f9Name = dircur + '/statistics'
```

koja se koristi samo prilikom ispisivanja reputacija i to na način da se dodaju (datoteka se otvara u append modu) brojevi na kraj datoteke.

```
f2 = open(f9Name, 'a')
if pos>100.0:
      f2.write(str(100.0) + "\n" )
else:
      f2.write(str(pos) + "\n" )
f2.close()
```

Beskoristan ispis koji je uklonjen.

BIBLIOTEKE

Potrebne biblioteke:

- *dpkt-1.7* sa zakrpom koja je već instalirana, dpkt se nalazi u $PROJECT/lib direktoriju tako da ga nije potrebno dodatno instaliravat
- *pybgpdump.py* – samo jedna datoteka koja olakšava korištenje dpkt biblioteke koja se nalazi u $PROJECT/lib direktoriju
- *ipaddr.py* – za manipuliranje sa IPv4 i IPv6 adresama, nalazi se u $PROJECT/lib direktoriju
- *pychart* – za generiranje grafova, nalazi se u $PROJECT/lib direktoriju
- *ghostscript* – interpreter za PostScript i PDF datoteke, treba ga *pychart*, na linux sustavima preinstaliran, na windowsima se treba posebno skinuti sa http://pages.cs.wisc.edu/~ghost/

MEMORIJSKO PROFILIRANJE

Memorijsko profiliranje nisam obavio uspješno. Probao sam slijedeće alate:

- Guppy
  - http://guppy-pe.sourceforge.net/
  - Na windows platformi neuspješno pokretanje, ne može učitat dll-ove potrebne za rad
- PySizer
  - http://pysizer.8325.org/
  - Nekompatibilan jer je rađen za python verziju 2.3 i 2.4
- Python Memory Validator
  - http://www.softwareverify.com/python-memory.php
  - Samo za windows platformu
  - Nisam našao podatke o količini zauzete memorije, nego samo o broju alociranih objekata, što se pokazalo kao nepotrebna informacija

VREMENSKO PROFILIRANJE

Ovo su rezultati profiliranja stare verzije (originalnog diplomskog) i verzije 2.0.
Svi testovi su rađeni na istom ulaznom skupu podataka, RIB tablica *rib.20110113.0000.parsed*, i update datoteke od *updates.20110113.0000.bz2* do *updates.20110114.1545.bz2*, sa vremenskim prozorom od **4** sata.
Platforma za testove je Windows XP u Eclipse okruženju.

STARA VERZIJA

Ukupno vrijeme izvođenja: **293** sekunde

Funkcija	Ukupno vrijeme (sekunde)
ReadRIB	64.7
FileWritePrefInf	48.9
Write nad file objektima	36.8
WinCalculation	36
Postpend	7.6
RemoveDouble	6.8
dpkt:unpack	6.8
GetIntegerAS	4.6

```
Fri May 11 05:01:45 2012    prof

         138287513 function calls (135010405 primitive calls) in 293.121
seconds

   Ordered by: internal time
   List reduced from 211 to 10 due to restriction <10>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1   64.737   64.737   90.964   90.964
D:\fax\mreze\mreze\old\classes.py:623(ReadRIB)
       11   48.943    4.449   89.634    8.149
D:\fax\mreze\mreze\old\classes.py:799(FileWritePrefInf)
 47159686   36.828    0.000   36.828    0.000 {method 'write' of 'file'
objects}
       10   36.072    3.607   57.075    5.707
D:\fax\mreze\mreze\old\classes.py:873(WinCalculation)
  2818344    7.676    0.000    8.941    0.000
D:\fax\mreze\mreze\old\classes.py:112(Postpend)
   805871    6.863    0.000   10.596    0.000
D:\fax\mreze\mreze\old\classes.py:157(RemoveDouble)
  1694162    6.805    0.000    9.393    0.000
D:\fax\mreze\mreze\old\lib\dpkt\dpkt.py:124(unpack)
 11798641    4.655    0.000    4.655    0.000
D:\fax\mreze\mreze\old\classes.py:74(GetIntegerAs)
9183016/7402637    4.601    0.000    8.910    0.000 {len}
  3428029    3.449    0.000    3.449    0.000
D:\fax\mreze\mreze\old\classes.py:1066(IncreaseSum)
```

VERZIJA 2.0 SA TEKSTOM

Uključen tesktualni ispis
Uključen verbose ispis na konzolu

Ukupno vrijeme izvođenja: **256** sekundi

Funkcija	Ukupno vrijeme (sekunde)
FileWritePrefInf	52.3
WinCalculation	39.1
Write nad file objektima	37.8
ReadRIB	18.3
Postpend	7.6
RemoveDouble	8.5
dpkt:unpack	6.6
PostPend	6.4
GetIntegerAS	4.7

```
Fri May 11 05:31:40 2012    prof

         150249373 function calls (146972270 primitive calls) in 256.838
seconds

   Ordered by: internal time
   List reduced from 446 to 10 due to restriction <10>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
       12   52.387    4.366   94.536    7.878
D:\fax\mreze\mreze\rep_2_0\core.py:794(FileWritePrefInf)
       11   39.124    3.557   62.060    5.642
D:\fax\mreze\mreze\rep_2_0\core.py:883(WinCalculation)
 51487751   37.826    0.000   37.826    0.000 {method 'write' of 'file'
objects}
        1   18.380   18.380   44.390   44.390
D:\fax\mreze\mreze\rep_2_0\core.py:616(ReadRIB)
   805871    8.505    0.000   13.164    0.000
D:\fax\mreze\mreze\rep_2_0\core.py:144(RemoveDouble)
  1694162    6.642    0.000    9.228    0.000
./lib\dpkt\dpkt.py:124(unpack)
  2818344    6.480    0.000    7.663    0.000
D:\fax\mreze\mreze\rep_2_0\core.py:101(Postpend)
 12144368    4.727    0.000    4.727    0.000
D:\fax\mreze\mreze\rep_2_0\core.py:62(GetIntegerAs)
9532765/7752386    4.606    0.000    8.842    0.000 {len}
  3773940    3.802    0.000    3.802    0.000
D:\fax\mreze\mreze\rep_2_0\core.py:1105(TimePercentageRep)
```

VERZIJA 2.0 BEZ TEKSTA

Isključen tesktualni ispis
Isključen verbose ispis na konzolu

Ukupno vrijeme izvođenja: **154** sekunde

Funkcija	Ukupno vrijeme (sekunde)
WinCalculation	38.8
ReadRIB	18.3
RemoveDouble	8.5
PostPend	6.6
GetIntegerAS	4.7

```
Fri May 11 05:38:31 2012    prof

         90155066 function calls (86877963 primitive calls) in 154.722
seconds

   Ordered by: internal time
   List reduced from 440 to 10 due to restriction <10>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
       11   38.884    3.535   61.695    5.609
D:\fax\mreze\mreze\rep_2_0\core.py:883(WinCalculation)
        1   18.259   18.259   44.078   44.078
D:\fax\mreze\mreze\rep_2_0\core.py:616(ReadRIB)
   805871    8.435    0.000   13.054    0.000
D:\fax\mreze\mreze\rep_2_0\core.py:144(RemoveDouble)
  1694162    6.631    0.000    9.203    0.000
./lib\dpkt\dpkt.py:124(unpack)
  2818344    6.446    0.000    7.617    0.000
D:\fax\mreze\mreze\rep_2_0\core.py:101(Postpend)
 12144368    4.705    0.000    4.705    0.000
D:\fax\mreze\mreze\rep_2_0\core.py:62(GetIntegerAs)
9532765/7752386    4.538    0.000    8.743    0.000 {len}
```

KOMENTAR REZULTATA

Stara verzija:	293
Verzija 2.0 sa text izlazima:	256
Verzija 2.0 bez text izlaza:	154

Vidimo da je verzija 2.0 sa tekstualni izlazima brža za otprilike 13%. Tu dolazi do izražaja samo optimizacija prilikom učitavanja RIB tablice, na koju se u staroj verziji troši 64.7 sekundi, a u verziji 2.0 18.3 sekunde. To je ubrzanje od 42%. Sve ostalo ostaje otprilike isto.

Pravo ubrzanje se primjeti kada se isključe tekstualni ispisi, i na konzoli i u tekstualne datoteke. Onda ubrzanje iznosi oko 50%.

DALJNJI RAD

Slijedi prijedlog tema koje bi se trebale napraviti i popravit kako bi ovaj program postao još korisniji, bolji i lakši za upotrebu. Uz ime svake teme je dana i procjena koliko bi vremena otprilike trebalo utrošiti da se implementira ista i to u slučaju da tek dobije seminar (znači uključeno i vrijeme upoznavanja sa samim projektom).

REFAKTORIRANJE I KOMENTIRANJE CORE MODULA (50 SATI)

Nakon što se pročita poglavlje KRITIKE ORIGINALNOG RADA, javljaju se sumnje da li stvarno ovaj program radi i računa ono što bi trebao. Prvo i osnovno je detaljno proučiti i shvatiti što se dešava u klasama *PrefixPath* i *PrefixAS0Binding*. Slijedeći korak je refaktoriranje cijelog *core.py* modula na taj način da se postave logična imena klasa i varijabli unutar istih. Uz sve to treba i dobro komentirati sve metode i klase tako da kod bude čitljiviji svima koji će ga čitati kasnije. Na kraju treba provjeriti sve matematičke formule jesu li zaista točno napisane i dobijaju li se točni rezultati.

PAMETNIJE UČITAVANJE UPDATE DATOTEKA (16 SATI)

Neki update dumpovi imaju pomaknuto vrijeme, vjerojatno zbog vremenske zone, pa bi se to moglo malo detaljnije obraditi.
Program trenutno radi tako da uzme listu update dumpova, sortira ju po datumu i krene ju parsirati. Ako postoji više različitih dumpova (recimo od oba primjera iz diplomskog), program će prvo proparsirati one koji imaju raniji datum, bez obzira koji je datum zapravo zadan kao početak, i tako dati krive rezultate. Trebalo bi uskladiti vrijeme početka (*time_start*, koje se definira u *config.ini*) sa vremenom update poruke, da se stvarno počnu parsirati od tog trenutka, a ne od datoteke sa najmanjim vremenom kao što je dosad.

IZDVAJANJE UČITAVANJA RIB TABLICE + IZABIR AS-ova NAKON PARSIRANJA ISTE (24 SATA)

Trenutno se RIB tablica učitava posebno u svakoj metodi analize zato što se odmah pri učitavanju i parsira. Samo učitavanje bi se moglo izdvojiti i premjestiti u Analyzer klasu, pa se kao parametar slati svakoj metodi posebno. Na taj način bi se ubrzalo izvođenje ako se žele upotrijebiti obje metode analize. I moglo bi se napraviti pretparsiranje AS-ova. Jer trenutno se autonomni sustavi, za koje se želi generirati graf, navode u konfiguracijskoj datoteci. To zna rezultirat time da ako traženi AS ne postoji u RIB tablici niti se za njega ne dobijaju update poruke, da će graf biti prazan za taj AS. Taj problem bi se mogao riješiti tako što bi se prvo učitala RIB tablica pa korisniku ponudio izbor svih AS-ova koji su pronađeni unutra.

MOGUĆNOST GENERIRANJA GRAFOVA NAKON SVAKOG PROZORA, ILI NAKON SVAKIH N PROZORA (24 SATA)

Trenutno se generira jedan graf za cijeli period vremena računanja. Mogla bi se uvest opcija generiranja grafova za svaki prozor ili za svakih n prozora. Pychart ima podršku da može generirati i više grafova po jednoj datoteci, tako da bi se moglo napraviti i generiranje svakog grafa posebno u svoju datoteku, ili svih (ili nekih) grafova u nekoliko datoteka (recimo po danima).

PROUČITI I INKORPORIRATI ZEBRA PARSER U OVAJ PROGRAM (24 SATA)

Zebra dump parser je uključen u distribuciju ovog programa, nalazi se u direktoriju RIB_Parser. Trebalo bi proučiti njegov rad i inkorporirati ga u ovaj program tako da se može zadati neka unaprijed skinuta RIB tablica pa program prvo pokrene Zebru kao eksterni proces i onda iskoristi njezin izlaz kao RIB tablicu za svoju analizu.

PRERAĐIVANJE ZEBRA DUMP PARSERA U PYTHON (80+ SATI)

Idealno bi bilo kad bi se Zebra mogla preraditi u python pa da se lakše uključi u ovaj program. To bi rezultirao boljom integracijom nego da se Zebra pokreće kao vanjski proces, međutim to je prilično kompleksan parser i to bi zahtijevalo jako puno vremena.

MOGUĆNOST DOHVAĆANJA ULAZNIH PODATAKA SA WEB/FTP POSLUŽITELJA (50 SATI)

Nakon što se napravi integracija sa Zebra dump parserom, slijedeći logični korak je samostalno dohvaćanje ulaznih datoteka sa online poslužitelja. Treba prvo locirati mjesta otkud se tražene datoteke mogu skinuti, jesu li u pravom formatu i kolika arhiva podataka postoji. Nakon toga bi se trebalo napraviti da se program sam spoji na zadanog poslužitelja i skine zadane datoteke.

ONLINE RAD (50 SATI)

Nakon što se napravi mogućnost samostalnog dohvaćanja ulaznih podataka mogla bi se napraviti opcija da program kontinuirano skida update dumpove pa da na taj način radi realnom vremenu i svako malo generira neki graf.

LISTA AS-ova

Popis AS-ova koji se nalaze u podacima iz primjera:

109	3292	6083	9121	13768	18779	24205	28310	34714	38520	43802	46063
145	3320	6128	9146	14868	18881	24206	28554	34758	38524	43833	47481
174	3339	6316	9228	15095	20485	24207	28555	34779	38542	44285	47814
209	3356	6453	9316	15201	20804	24209	28590	34984	38548	44356	48011
227	3475	6503	9341	15412	20892	24210	28624	35002	38613	44615	48195
286	3491	6663	9498	15445	20960	24212	28645	35297	38755	44632	48609
376	3538	6684	9503	15742	20965	24339	28666	35491	38757	44898	48774
637	3549	6713	9505	15818	21107	24518	28716	35561	38758	44942	48881
638	3561	6739	9768	16016	21127	24520	29015	35567	38759	44957	49111
680	3786	6747	9785	16058	21189	24521	29024	35753	38770	45123	49630
688	3816	6762	9794	16131	21246	24525	29032	35819	38776	45147	50288
701	4021	6854	9875	16161	21275	24526	29072	35916	38783	45282	50356
702	4134	6939	9989	16284	22080	24532	29073	35931	38789	45287	50592
703	4230	7018	10026	16323	22205	24535	29255	36917	38896	45288	50886
721	4323	7162	10113	16338	22284	24536	29311	36991	39156	45289	51143
1103	4434	7303	10217	16345	22388	24538	29518	36992	39200	45292	51193
1221	4436	7473	10429	16397	22431	24560	29544	37074	39386	45298	51750
1239	4454	7474	10445	17379	22773	24785	29614	37105	39737	45302	51793
1273	4635	7500	11032	17408	23352	25019	29761	37216	39912	45305	52229
1280	4637	7660	11259	17477	23651	25229	30678	37246	40285	45317	53105
1299	4648	7713	11381	17557	23666	25395	30895	38040	41313	45322	55448
1600	4713	7738	11537	17658	23671	26592	30988	38144	41864	45324	55661
1680	4739	7922	11915	17670	23679	26615	31133	38147	41916	45325	55664
1733	4755	8151	12182	17671	23691	26616	31499	38149	42220	45474	55665
1853	4761	8400	12302	17769	23698	26769	31618	38150	42258	45609	55666
1959	4766	8402	12389	17803	23756	26876	31641	38154	42337	45698	55670
2497	4788	8452	12445	17826	23946	27064	32098	38155	42396	45706	55671
2516	4800	8495	12479	17874	23947	27065	32289	38156	42432	45707	55675
2518	4802	8551	12486	17885	23949	27066	32327	38163	42613	45711	55825
2588	5391	8591	12552	17996	23950	27137	32528	38164	42757	45712	55926
2764	5400	8657	12644	18004	23951	27138	32787	38165	42791	45714	262738
2828	5486	8738	12654	18059	23953	27216	32869	38496	42896	45722	262741
2907	5511	8744	12715	18101	24077	27724	33770	38500	42957	45725	262788
2914	5541	8866	12738	18106	24130	27750	34072	38503	42979	45731	
3212	5588	8928	12880	18347	24194	27768	34109	38504	43061	45786	
3215	5603	8966	12956	18351	24196	27817	34224	38506	43351	46023	
3216	5606	9002	13145	18365	24197	27932	34279	38509	43376	46046	
3255	5617	9050	13178	18379	24201	28161	34419	38511	43438	46054	
3257	5803	9116	13237	18479	24202	28202	34444	38515	43462	46056	
3269	6045	9119	13331	18709	24203	28286	34665	38518	43663	46062	