Universidade Federal de São Carlos
Departamento de Computação

BGP Statistics

Professor Dr. Cesar Augusto Marcondes

Bruna Zamith Santos (RA: 628093)
Lucas Queiroz (RA: 587087)
Vitor Gallera (RA: 726595)

07/2018
São Carlos - SP, Brasil

Conteúdo

1 Introdução

O presente relatório visa detalhar as atividades realizadas no desenvolvimento do projeto entitulado *BGP Statistics*, na disciplina "Engenharia de Segurança Cibernética". Esta foi ministrada pelo professor Dr. Cesar Augusto Marcondes e orientada pelo aluno de doutorado Emerson Barea, ao longo do primeiro semestre de 2018, no Departamento de Computação da Universidade Federal de São Carlos.

O documento está organizado como segue: Seção 2 faz uma breve introdução sobre Tabelas BGP; a Seção 3 expõe o objetivo do trabalho desenvolvido; a Seção 4 detalha os passos seguidos para o desenvolvimento do projeto; a Seção 5 expõe os resultados obtidos; e, por fim, a Seção 6 contém as conclusões obtidas a partir destes resultados.

Todos os algoritmos desenvolvidos e recursos utilizados neste projeto encontram-se no repositório `https://github.com/bzamith/seguranca-cibernetica`.

2 Tabelas BGP

Sistemas Autônomos (do inglês, Autonomous Systems - ASes) são um conjunto de prefixos de IPs sob gestão de um grupo ou instituição comum e que, portanto, possuem características e políticas de roteamento comuns. Do próprio nome, "Sistema"pois trata-se de uma estrutura com complexidade mínima de rede; e "Autônomo"pois a entidade possui autonomia, ou poder de decisão para as diferentes opções de caminhos externos (conexão com outros AS) [1].

O *Border Gateway Protocol* (BGP), por sua vez, é um protocolo de roteamento entre ASes, realizando troca de informações de roteamento entre os roteadores afim de determinar os caminhos ideais para o fluxo de tráfego [2]. Estabelece conexões TCP e, ao invés de manter somente o custo para cada destino, cada roteador BGP tem controle de qual caminho está sendo usado [3].

Os roteadores fazem uso de 3 tipos de tabelas de roteamento, cada uma com um diferente propósito:

- *BGP Neighbor*: Contém informações sobre os BGPs vizinhos;
- *BGP Table*: Também conhecida como BGP RIB, contém as informações de acessibilidade da camada de rede. Além disso, contém todas as rotas de todos os vizinhos, centenas de rotas para uma mesma rede, com diferentes atributos;
- Tabela de Roteamento BGP: Contém apenas as melhoras rotas da *BGP Table*. Após a BGP ter selecionado o melhor caminho para uma rede, esse caminho é adicionado à Tabela de Roteamento BGP [4].

É possível a obtenção de Tabelas de Roteamento BGP através dos chamados *Looking Glass* dos provedores.

[1] `ftp://ftp.registro.br/pub/gter/gter28/07-Asbr.pdf`
[2] `http://www.noction.com.br/blog/bgp_bestpath_selection_algorithm`
[3] `http://www.higuita.com.br/sr/2-Roteamento-BGP.pdf`
[4] `http://bgp.us/routing-table/`

3 Objetivo

Este projeto objetiva a identificação de redundância de rotas entre IXs (do inglês, *Internet exchange point*) brasileiros. Os IXs funcionam como hubs em que provedores podem conectar seus servidores e, deste modo, facilitar o tráfego das informações.

> *No Brasil, o IX.br é um projeto de IXs locais gerido pelo Núcleo de Informação e Coordenação do Ponto Br (Nic.br) e pelo Comitê Gestor da Internet no Brasil (CGI.br), que facilita o fluxo de informações entre provedores de internet e conteúdo online no país. Com pico de tráfego total de mais de 3 Tbit/s, o Brasil tem IXs em mais de 20 localidades* [5].

Para atingir o objetivo, as seguintes etapas foram seguidas: Extração das Tabelas de Roteamento BGP dos IXs brasileiros, *parsing* das tabelas a fim de inserir as informações num banco de dados, e por fim a análise dos dados extraídos. Estas etapas estão detalhadamente descritas na Seção 4.

[5] `nic.br/noticia/na-midia/fortaleza-tem-novo-ix/`

4 Desenvolvimento

4.1 Extração dos Dados

Os dados utilizados no projeto foram extraídos a partir de um protocolo telnet via expect script, executado de hora em hora, durante 7 dias. No total, extraímos tabelas de roteamento BGP IPv4 e IPv6 de 29 IXs, sendo estes:

Tabela 1: Relação dos IXs cujas tabelas foram extraídas

Salvador, BA (ba)	Belém, PA (bel)	Campinas, SP (cas)	Fortaleza, CE (ce)
Cuiabá, MT (cgb)	Campina Grande, PB (cpv)	Caxias do Sul, RS (cxj)	Brasília, DF (df)
Goiânia, GO (gyn)	Foz do Iguaçu, PR (igu)	João Pessoa, PB (jpa)	Lajeado, RS (laj)
Londrina, PR (lda)	Manaus, AM (mao)	Maringá, PR (mgf)	Belo Horizonte, MG (mg)
Natal, RN (nat)	Recife, PE (pe)	Curitiba, PR (pr)	Santa Maria, RS (ria)
Rio de Janeiro, RJ (rj)	Porto Alegre, RS (rs)	Florianópolis, SC (sc)	Aracajú, SE (se)
São José dos Campos, SP (sjc)	São José do Rio Preto, SP (sjp)	São Paulo, SP (sp)	Teresina, PI (the)
Vitória, ES (vix)			

Entretanto, para permitir uma análise mais rápida e direta, apenas uma tabela de cada IX foi levada em consideração, e apenas para dados IPv4. No total, foram coletados 812.513 diferentes endereços IP.

As tabelas possuem o seguinte formato (exemplo reduzido extraído de uma Tabela de Roteamento BGP):

```
spawn telnet lg.ba.ptt.br
terminal length 0
show ip bgp
Trying 200.128.0.83...
Connected to lg.ba.ptt.br.
Escape character is '^]'.
==============================================
=                PTTMetro BA                 =
=             Contact: eng@ptt.br            =
=               +55 11 5509-3550             =
=             inoc-dba: 22548*100            =
=                                            =
=          Looking Glass Server              =
=   All connections and keystrokes logged    =
==============================================

lg.ba.ptt.br> terminal length 0
lg.ba.ptt.br>
lg.ba.ptt.br> show ip bgp
BGP table version is 0, local router ID is 200.219.145.252
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale, R Removed
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 0.0.0.0          200.219.145.53                         0 53236 61568 i
*> 1.0.4.0/22       200.219.145.23                         0 53242 6939 4826 38803 56203 i
*> 1.0.4.0/24       200.219.145.23                         0 53242 6939 4826 38803 56203 i
*> 1.0.5.0/24       200.219.145.23                         0 53242 6939 4826 38803 56203 i
*> 1.0.6.0/24       200.219.145.23                         0 53242 6939 4826 38803 56203 i
*> 1.0.7.0/24       200.219.145.23                         0 53242 6939 4826 38803 56203 i
*> 1.0.16.0/24      200.219.145.23                         0 53242 7738 2497 2519 i
*> 1.0.64.0/18      200.219.145.23                         0 53242 6939 10026 2519 7670 18144 i
*> 1.0.128.0/17     200.219.145.23                         0 53242 7738 6762 38040 9737 i
*> 1.0.128.0/18     200.219.145.23                         0 53242 7738 6762 38040 9737 i
*> 1.0.128.0/19     200.219.145.23                         0 53242 7738 6762 38040 9737 i
*> 1.0.128.0/24     200.219.145.23                         0 53242 7738 6762 38040 23969 ?
*> 1.0.129.0/24     200.219.145.23                         0 53242 6939 4651 23969 i
*> 1.0.131.0/24     200.219.145.23                         0 53242 7738 6762 38040 23969 ?
```

4.2 Banco de Dados

Para armazenamento das informações das tabelas de roteamento BGP, foram adicionadas tabelas a um banco de dados SQL já criado pelo orientador do projeto, Emerson Barea. Tal banco de dados é responsável por armazenar diversas informações de determinado *Looking Glass*. A figura 1 mostra um modelo esquemático do banco de dados completo.

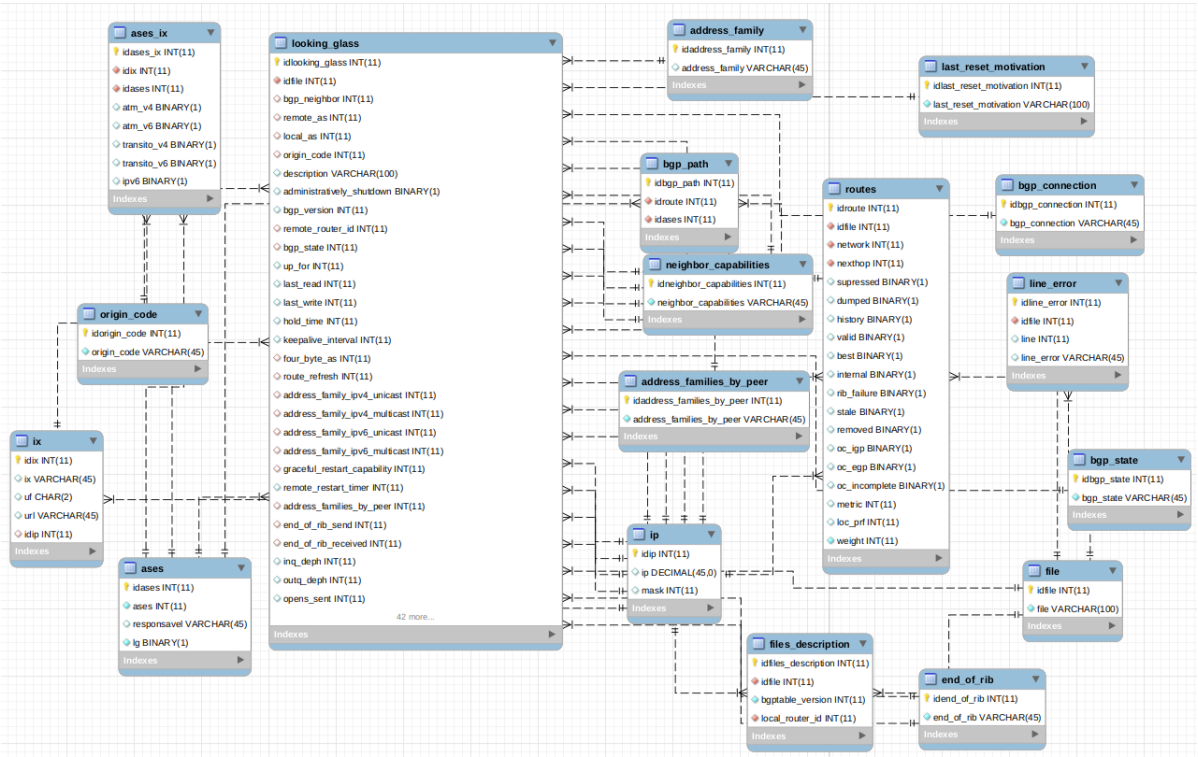

Figura 1: Modelo esquemático do banco de dados

As tabelas criadas para armazenamento de informações das Tabelas de Roteamento BGP foram a tabela "Routes"e a tabela "BGP_Path", sendo que as tabelas utilizadas no nosso projeto são:

- File: Armazena o nome do arquivo;
- Files_Description: Chave estrangeira para a tabela "File", amarzena o "*local router id*"e a versão da tabela BGP;
- IP: Armazena o número IP em decimal e sua respectiva máscara;
- Routes: Armazena as informações de cada rota da Tabela de Roteamento BGP;
- BGP_Path: Armazena os *paths* de cada rota (uma rota pode ter um ou mais *paths*).

4.3 Parsing

O algoritmo de *parsing* da tabela BGP foi feito em linguagem Python. Após o *parsing*, já é feita a correta inserção dos valores no banco de dados. Esses valores são:

- *Local Router Id*;
- Versão da tabela;
- Para cada rota:
  - *Status codes*;
  - *Network*;
  - *Next hop*;
  - *Metric*;

- *Local preference*;
- *Weight*;
- *Path*;
- *Origin codes*.

Existiram algumas dificuldades no desenvolvimento do algoritmo de *parsing*. A principal delas se deve ao fato de que a Tabela de Roteamento BGP não possui um divisor comum entre as colunas de informação, sendo que o espaçamento é variável; além disso, nem todos os valores de todas as colunas estão presentes, podendo alguns serem vazios. Para contornar esse problema, tivemos que considerar os espaços entre as colunas do *header*, e então usá-los como indices para as informações seguintes.

Não obstante, o algoritmo é programado para lidar com casos onde a tabela está corrompida, ou vazia. Ainda, a inserção no banco de dados não é feita se ela já foi inserida anteriormente. Assim, antes de cada inserção, é primeiro feita uma consulta ao banco. Devido ao grande volume de dados, otimizamos o código o máximo possivel, usando conceitos de tabela *hash*, dicionário e acesso a memória. Por fim, foi preciso criar um código que pudesse funcionar tanto para o IPv4 quanto para o IPv6, tendo em vista que a formatação e a quebra de linhas dos dois tipos de tabela são diferentes.

4.4 Análise

A análise dos resultados foi feita da seguinte forma: Primeiramente, extraímos do banco de dados as informações da tabela "IP"(o próprio IP em decimal, mais a máscara). Entâo, fizemos a leitura por meio de programação em linguagem R. Um conceito que buscamos entender foi como redes iguais podem ser representadas de maneira diferente. Isso porque uma mesma rede pode possuir máscaras diferentes e, portanto, níveis de representação diferentes. Para isso, fizemos primeiro a conversão do IP para decimal durante o parsing e, posteriormente, a reconversão para string (em R). Então, criamos uma função (Algoritmo 1) que retorna a *subnet* com base no valor do endereço IP e sua máscara. Por exemplo, 255.0.0.0/8, 255.255.0.0/16 e 255.255.255.0/24 depois da conversão, reconversão e processamento, pertenceriam todos ao mesmo bloco, o 255.0.0.0/8. Com isso, podemos fazer uma anãlise da redundância de IPs em rotas.

```
library(binaryLogic)
library(iptools)
library(plyr)

subnet = function(ip,mask){
  octetos = strsplit(ip,"\\.")
  ip_binario=c()
  for(i in 1:4){
    binarios = as.binary(octetos[[1]][i],n=0)
    binarios = c(rep(0,8-length(binarios)),binarios)
    ip_binario = c(ip_binario,binarios)
  }
  ip_binario = c(ip_binario[1:mask],rep(0,32-mask))
  saida=""
  for(i in 1:4){
    atual = ip_binario[(8*(i-1)+1):(8*i)]
    saida=paste(saida,Reduce(function(x,y) x*2+y, atual),sep="")
    if(i!=4){
      saida = paste(saida,".",sep="")
    }
  }
  return(saida)
}
```

Listing 1: Algoritmo que retorna a *subnet* de um dado IP e sua máscara

Detalhamos na seção 5 os resultados das seguintes análises:

- Qual a porcentagem de redundância de IPs em rotas da Tabela de Roteamento BGP?
- Em quais classes elas mais ocorrem (Classes A, B, C e D)?
- Qual a frequência desses blocos de endereços IPs?
- Esses blocos com maior frequência estão bem distribuídos ou concentrados em uma determinada região e/ou estado?

Para a classificação dos endereços IP, convertemos o primeiro octeto do endereço em binário e verificamos: Se começar com ”0”, é classe A; caso comece com ”10”, é classe B; se começar com ”110”, é classe C; se começar com ”1110”é classe D; por fim, se nenhuma das anteriores, é classe E [6].

[6] https://www.tutorialspoint.com/ipv4/ipv4_address_classes.htm

5 Resultados

Dos 812.513 endereços de IP, pudemos obter 752.148 blocos de IP. Assim, temos 60.365 endereços redundantes, o que caracteriza 7.43% de redundância. A Tabela 2 mostra o tamanho dos blocos (quantidade de IPs contidos) e a quantidade de blocos que possuem esse tamanho. Por exemplo, o bloco do endereço 87.68.0.0 possui 6 IPs contidos dentro dele, e existem mais 29 blocos que também possuem tamanho 6.

Tabela 2: Tamanho dos blocos por Quantidade

Tamanho do Bloco	**Quantidade de Blocos**
1	700.478
2	44.449
3	6.018
4	970
5	199
6	30
7	4

Quanto às classes de endereços, elas estão distribuídas segundo as frequências expostas na Tabela 3.

Tabela 3: Frequência das classes de endereço IP

Classe	**Frequência**
A	4.234
B	406.307
C	27.6393
D	32.210
E	33.004

A Tabela 4 associa os tamanhos dos blocos com a classes. Assim, podemos avaliar se determinadas classes estão associadas com blocos maiores e, portanto, com mais recorrência.

Tabela 4: Tamanho dos blocos por Classes

Tamanho do Bloco	**A (%)**	**B (%)**	**C (%)**	**D (%)**	**E (%)**
1	0.57	53.77	37.14	4.17	4.34
2	0.51	56.32	32.42	5.73	5.00
3	0.33	63.40	25.75	5.61	4.88
4	0.51	65.46	22.37	4.94	6.70
5	69.34	21.10	6.03	3.51	0
6	46.66	30.00	3.33	20.00	0
7	0	50	50	0	0

Visando uma análise visual, plotamos alguns gráficos em busca de informações relevantes e respostas para as perguntas expostas na seção anterior (Seção 4.4). As Figuras 2 e 3 buscam uma relação entre os endereços de IP e os tamanhos dos blocos, a fim de determinar se alguma região de endereços de IP possuia ou não maior recorrência de redundância. A Figura 2 trata-se de um gráfico de barras e a Figura 3, um gráfico de dispersão.

Outra informação importante era se alguma classe de endereçamento IP podia ser associada com maior recorrência de redundância. Para isso, foi plotado o gráfico da Figura 4.

Por fim, o gráfico da Figura 5 permite uma visualização da redundância de IPs por regiões. Para a construção de tal gráfico, separamos os blocos de tamanho 3, 4 e 5. Então, contamos quantas vezes eles aparecem em cada tabela, de cada cidade.

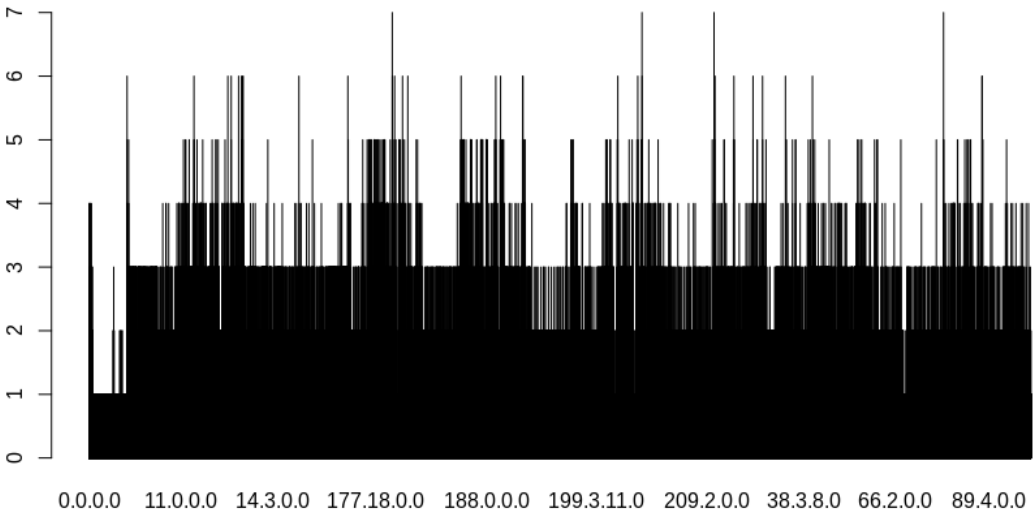

Figura 2: Gráfico de barras: Endereços x Tamanho dos blocos

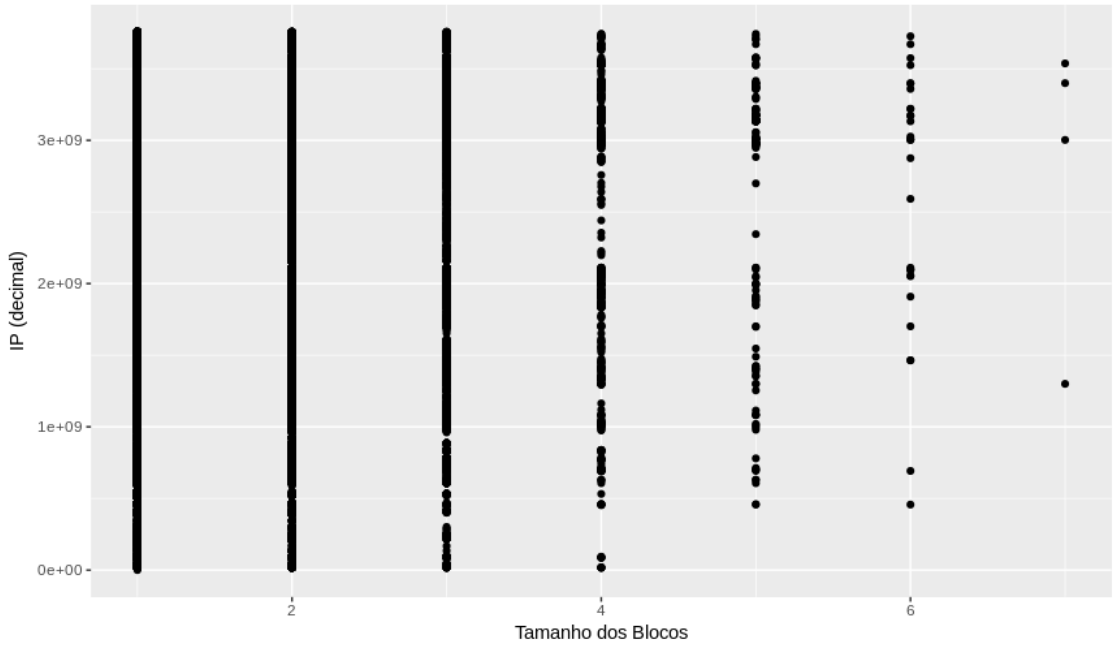

Figura 3: Gráfico de dispersão: Endereços x Tamanho dos blocos

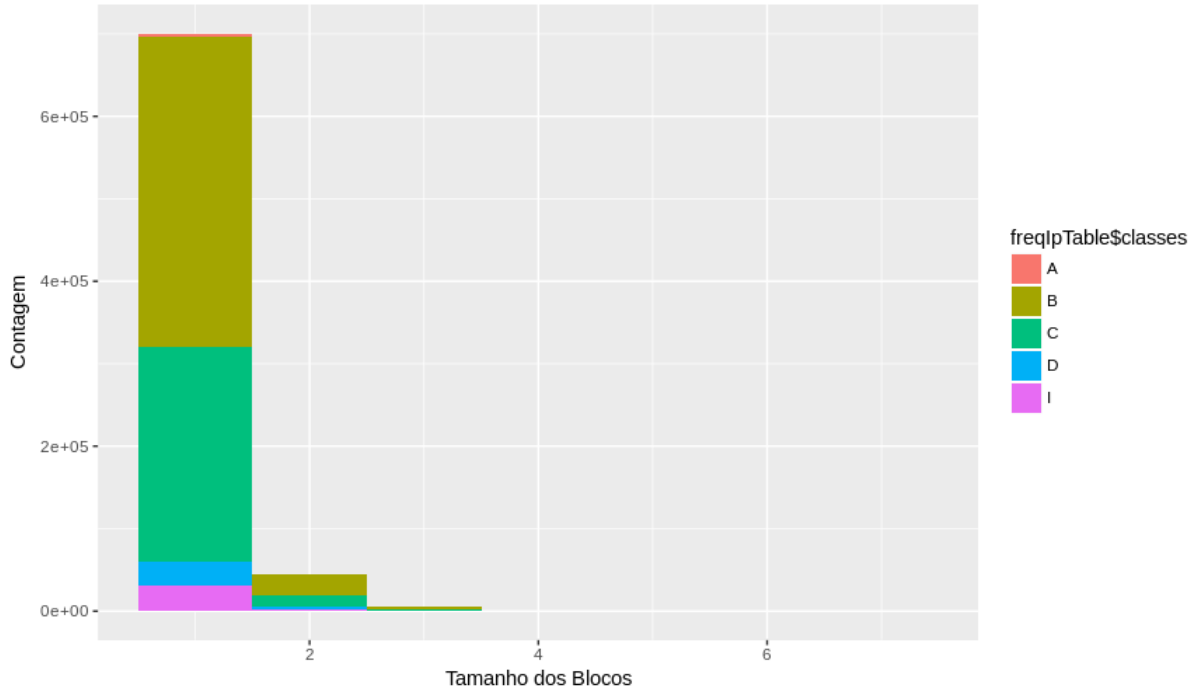

Figura 4: Histograma: Tamanho de bloco x Classe

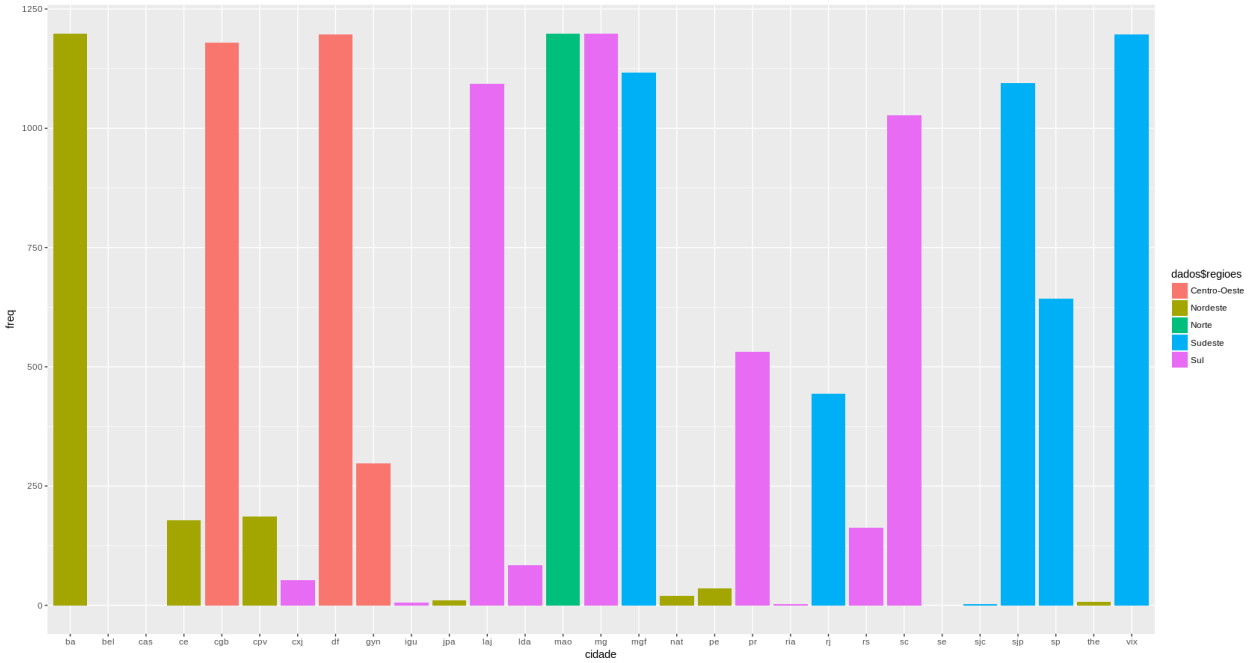

Figura 5: Histograma: Redundância x Regiões

6 Conclusões

A partir do estudo detalhado nesse relatório, conseguimos chegar a uma porcentagem de redundância em endereços de IP das rotas BGP, sendo igual a 7.43%. Essa informação é relevante em casos de busca por padronização ou generalização das tabelas, como no caso da criação de uma tabela única, ao invés da distribuição das mesmas entre os diferentes IXs. Pela análise das tabelas e dos gráficos plotados, expostos na seção anterior (Seção 5), fica clara uma maior frequência de redundância em IPs das Classes A e B.

Gostaríamos de ter expandido a pesquisa para tabelas de roteamento BGP obtidas dos mais diversos Looking Glass, todavia, para tanto seria necessário obter acesso à tabela inteira. Buscamos contato com as empresas via e-mail mas não obtemos nenhuma resposta. Entretanto, apesar da quantidade mais restrita de dados, acreditamos que os resultados mostram-se valiosos para futuras análises, principalmente com as contribuições do *parsing* da tabela de roteamento BGP e do banco de dados.

Assim, podemos responder as perguntas feitas na Seção 4.4:

- Qual a porcentagem de redundância de IPs em rotas da Tabela de Roteamento BGP? 7.43%.
- Em quais classes elas mais ocorrem (Classes A, B, C e D)? Classes A e B (vide Tabela 4).
- Qual a frequência desses blocos de endereços IPs? Exposto na Tabela 2.
- Esses blocos com maior frequência estão bem distribuídos ou concentrados em uma determinada região e/ou estado? Não existe nenhuma relação óbvia entre os tamanhos dos blocos e determinada região e/ou estado, vide Figura 5.

7 Árvore de redes

Como solicitado ao fim do projeto pelo Emerson, criamos uma árvore para os diferentes endereços IPs. Fizemos isso utilizando um script em Python, fazendo uso das bibliotecas *ipaddress*, *netaddr* e *anytree*. O algoritmo primeiramente obtém as *subnets* dos endereços IPs e então cria os níveis da hierarquia. Por fim, identifica as relações entre os IPs para construir a árvore final.

```
import csv
from netaddr import *
import ipaddress
import re
from anytree import Node, RenderTree

def numtoip(ip):
    return ipaddress.ip_address(int(re.sub('[^0-9]', '', str(ip)))).__str__()

def getstrcidr(nets):
  for i in range(0,len(nets)-1):
    nets[i] = str(nets[i])
  return(nets)

def diff(l1, l2):
  #return([x for x in l1 if x not in l2])
  #return(list(set(l1) - set(l2)))
  l1 = set(l1)
  l2 = set(l2)
  l3 = l1.difference(l2)
  return(list(l3))


with open('iptable.csv', 'rb') as csvfile:
  leitor = csv.reader(csvfile, delimiter=',')
  leitor.next()
  nets = list()
  for row in leitor:
    nets.append(str(numtoip(row[1])) + "/" + str(row[2]))


nos_nivel1 = list()
root = Node("root")

nivel1 = getstrcidr(cidr_merge(nets))
for i in range(0,len(nivel1)):
  nos_nivel1.append(Node(nivel1[i],parent=root))

resto = diff(nets,nivel1)
nivel2 = getstrcidr(cidr_merge(resto))
nivel2 = diff(nivel2,nivel1)
nos_nivel2 = list()

for i in range(0,len(nivel2)):
  for j in range(0,len(nivel1)):
    if(ipaddress.ip_network(unicode(nivel2[i]),False).subnet_of(ipaddress.ip_network(unicode(nivel1[j
        ]),False))):
      nos_nivel2.append(Node(nivel2[i],parent=nos_nivel1[j]))
      break

# encoding=utf8
import sys
reload(sys)
sys.setdefaultencoding('utf8')
log = open("arvore.txt","w")
for pre, fill, node in RenderTree(root):
  print("%s%s" % (pre, node.name),file=log)
```

Listing 2: Algoritmo para construção da árvore de IPs

A Figura 6 exibe uma amostra da árvore construída.

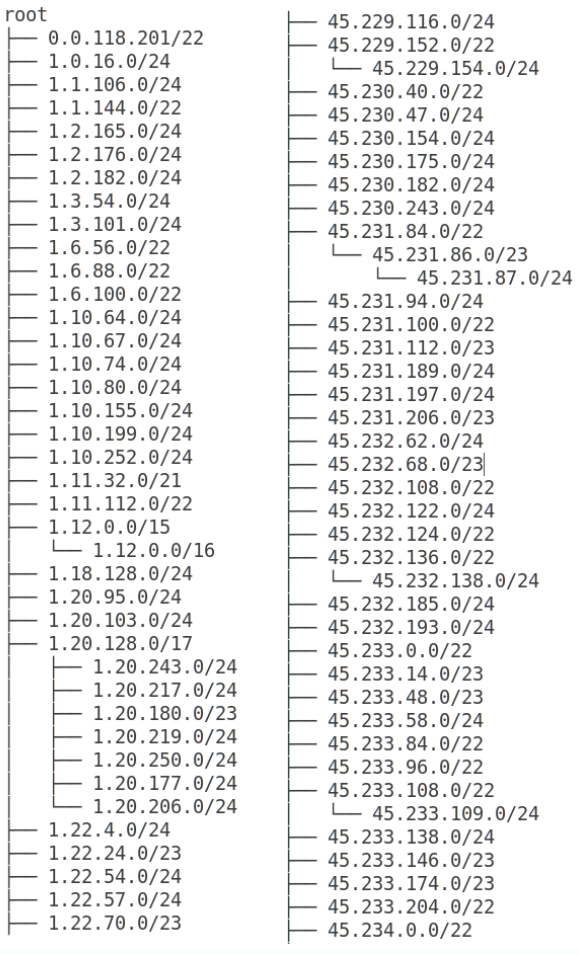

Figura 6: Amostra da árvore de IPs