Redes de Computadores
Trabalho Prático 3
PeeringDB - REST API

Ingrid Rosselis Sant'ana da Cunha

1 Introdução

O último trabalho da disciplina Redes de Computadores envolveu a criação de um par cliente-servidor que utiliza API com interface REST para consultas no PeeringDB via requisição HTTP. O servidor lê dados de um banco do PeeringDB e atende às requisições via HTTP do cliente em algum endpoint específico. A ideia do trabalho pode ser resumida a carregar a base do PeeringDB e fazer requisições de forma a gerar dados para duas análises distintas.
O trabalho foi feito em Python 3 e utiliza as bibliotecas `json` e `socket` por parte do cliente e `json` e `flask` por parte do servidor.
Os dois programas podem ser executados da seguinte maneira:

```
./server.py port Netfile Ixfile Netixlanfile
```

```
./client.py IP:port Opt
```

Onde `port` é o porto onde o servidor está executando, `Netfile` é o arquivo de redes, `Ixfile` é o arquivo de IXPs, `Netixlanfile` é o arquivo de associações rede-IXP, `IP` é o endereço de IP onde o servidor está e `Opt` é a opção de análise desejada.

2 Modelagem

2.1 Servidor

Para o servidor, foi escolhido utilizar o framework `Flask` por ser mais simples de implementar as requisições e, por causa disso, o servidor inteiro ficou extremamente simples. Ele inicia os processos requiridos pelo `Flask`, lê a entrada, abre os dados do PeeringDB e faz 4 chamadas de callback, a inicial requerida pelo `Flask` e as 3 requiridas para o acesso do cliente:

0. `/` : inicial do `Flask`;
1. `/api/ix` : todos os IXPs;
2. `/api/ixnets/<ix_id>` : identificadores das redes de um IXP;
3. `/api/netname/<net_id>` : nome de uma rede.

2.2 Cliente

O cliente é muito mais elaborado que o servidor. Ele possui um método para cada requisição de endpoint descrita na especificação; um método para reinício do `socket`, um método para saída no formato TSV e um método para cada uma das análises especificadas no trabalho.
As requisições foram feitas todas da mesma forma: monta-se o cabeçalho da requisição desejada, a requisição é enviada ao servidor, espera-se por toda a resposta do servidor (que foi lida em pacotes de 1024 bytes, por escolha de implementação), após isso a reposta é tratada – remove-se o cabeçalho padrão do `Flask` e coloca-se no formato de retorno descrito na especificação, e, por fim, o resultado é retornado.

Quanto às duas análises, elas podem ser resumidas nos seguintes pseudo-códigos:

Algorithm 1 Análise 0

```
resultado = dicionário com chave net_id
resposta1 ← requisição tipo 1
for ix_id em resposta1 do
  resposta2 ← requisição tipo 2
  for net_id em resposta2 do
    if net_id não conhecido then
      name ← requisição tipo 3
      count = 1
      resultado do net_id ← name e count
    else
      aumenta contagem de aparição da rede em 1
    end if
  end for
end for
return resultado
```

Algorithm 2 Análise 1

```
resultado = dicionário com chave ix_id
resposta1 ← requisição tipo 1
for ix_id em resposta1 do
  name ← name no ix_id
  count = 0
  resultado do ix_id ← name e count
end for
for ix_id em resultado do
  net_ids = requisição tipo 2
  count ← quantidade de itens em net_ids
end for
return resultado
```

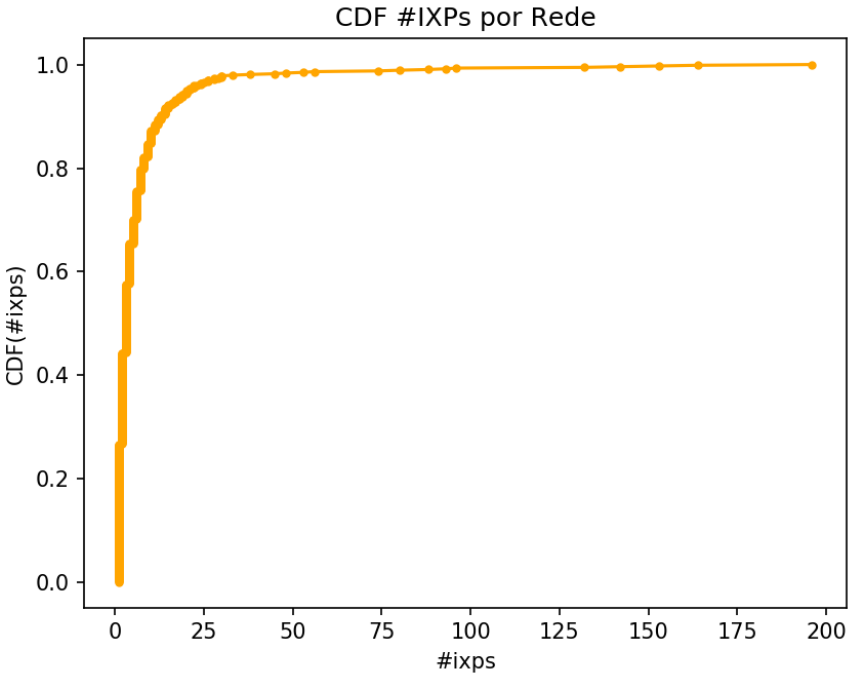
3 Análise de Resultados

Foi requisitado na especificação duas análises feitas no cliente, as relações IXPs-redes e redes-IXPs e, em seguida, deveríamos gerar os gráficos de CDF de ambos os resultados. A função de distribuição cumulativa (*cumulative distribution function*, CDF) fornece a probabilidade de que uma variável aleatória seja menor ou igual a um determinado valor. Ou seja, a CDF calcula a concentração de itens em determinado ponto.

Os dois gráficos mostrados abaixo são bem semelhantes, mostrando que o relacionamento entre redes e IXPs é de muitos para muitos.

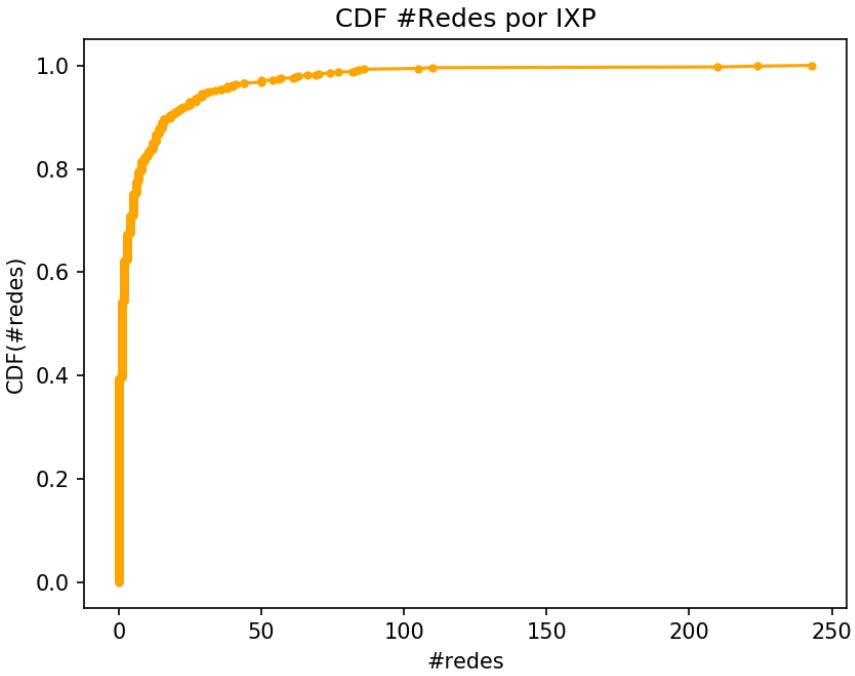
3.1 Redes por IXP

No gráfico abaixo podemos notar como ocorre a concentração de IXPs por redes: aproximadamente 95% das redes se encontram em até 20 IXPs, e esse número aumenta de maneira sútil.



3.2 IXP por Redes

Já para este gráfico a concentração de redes por IXPs é um pouco menos agressiva, 90% dos IXPs se relacionam com aproximadamente 25 redes, e esse número também aumenta de maneira sútil conforme mais redes são contabilizadas.



4 Dificuldades

Este trabalho prático foi o mais tranquilo em quesito de implementação nesta disciplina. Sua dificuldade real foi entender a proposta e como deveria ser inciado. Mesmo depois de ler várias vezes a especificação e vários materiais sobre o assunto, ainda era um pouco confuso se eu havia entendido direito a proposta e por onde o trabalho deveria ser iniciado.
A parte do servidor foi incrivelmente rápida, o `Flask` realmente deixa o trabalho no servidor praticamente inexistente. Já o cliente foi um grande desafio para entender como funcionam as requisições HTTP e tratar alguns problemas como: requisições consecutivas e montagem das análises.
O problema das requisições consecutivas é o seguinte: não é possível fazer duas requisições consecutivas sem ser obrigado a reiniciar o `socket` entre elas. Eu demorei muito até entender o problema e depois, para solucioná-lo, fiz um método que reinicia o `socket` e que era chamado entre as requisições.
A análise 0 foi difícil na parte de montagem: ela requeria um "work around"para obter os `net_ids` (pegar cada `ix_id` e a partir dele pegar cada `net_id`). Mas depois de planejar o código, a solução saiu sem grandes problemas (além da contagem repetidas de redes, que foi simples consertar).
A parte das CDFs foi interessante também, foi difícil entender o que foi pedido e gerar as visualizações, mas com elas ficou bem clara a distribuição das relações IXP-rede.

5 Conclusão

O objetivo deste trabalho prático era implementar um par cliente-servidor utilizando chamadas de consultas remotas usando a interface REST para o PeeringDB, cujos dados são disponibilizados pelo servidor em endpoints utilizando `Flask` para que o cliente possa fazer requisições HTTP e gerar duas análises.
Ambas as análises mostraram que o relacionamento entre IXPs e redes é bastante condensado – muitas redes em um IX,P e ambas as análises resultaram em gráficos aproximados.
Este trabalho pôs em prática o que foi visto em sala de aula, principalmente na âmbito de uso de socket, chamadas RPC e cabeçalhos HTTP. Os resultados aqui obtidos são considerados satisfatórios e o trabalho foi concluído sem grandes problemas.