Universidade Federal de Minas Gerais
DCC023: Redes de Computadores
Trabalho Prático
PeeringDB REST API

Introdução

Neste trabalho iremos implementar um par cliente-servidor utilizando chamadas de procedimentos remotos (RPC) em REST. Para fazer algo mais prático e ligado a outros assuntos discutidos na disciplina, focaremos em um sistema de consulta para o PeeringDB, um banco de dados de interconexões (*peerings*) entre redes membro de IXPs (Internet Exchange Points). Como de costume, o sistema possuirá um (1) programa cliente, que faz requisições HTTP para um servidor através de uma API REST; e (2) um programa servidor, que atende as requisições de clientes.

Utilizando uma API REST

Uma interface REST permite que sistemas na web troquem mensagens textuais independente de implementação e plataforma. De forma prática, um servidor REST oferece um conjunto de requisições HTTP específicas que o cliente usa para consultar e obter um conjunto de dados (ou parte dele). Geralmente, o formato do dado disponibilizado por essas APIs é o JSON. Por exemplo, execute o seguinte comando em bash (tendo acesso à internet):

```
wget -q https://www.peeringdb.com/api/ix/1 -O-
```

O comando `wget` realiza uma requisição HTTP para o endereço passado como parâmetro (detalhes: `man wget`). Você deve observar uma saída como esta (a saída está truncada):

```
{"meta": {}, "data": [{"id": 1, "org_id": 2, "org": {"id": 2, "name":
"Equinix", "website": "http://www.equinix.com", ...
```

O texto retornado está em formato JSON e é o resultado de uma consulta à API oficial do PeeringDB. Note que a URL da requisição contém todas as informações da pesquisa:

```
https://www.peeringdb.com     → endereço do servidor
/api                          → acesso à API desse servidor REST
/ix                           → informação sobre IXPs
/1                            → informação sobre o IXP com o identificador 1
```

O formato das URLs de uma API como essa é especificado pelo servidor. Damos o nome de *endpoint* a cada uma dessas especificações de URL.

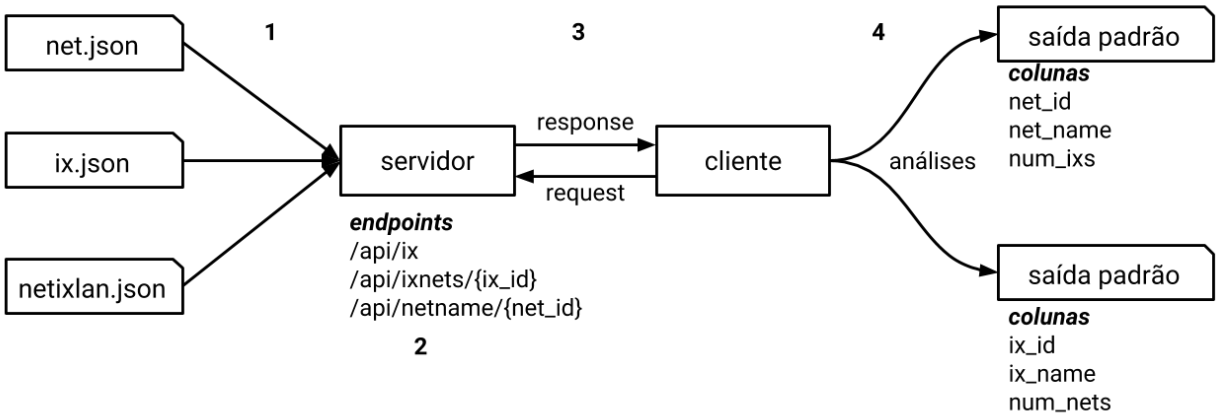
Objetivo do trabalho

Você deve implementar os lados cliente e servidor de uma API como a descrita acima para os dados do PeeringDB, **mas sem utilizar a biblioteca do PeeringDB para acesso à API** (isto é, você deve construir as os cabeçalhos HTTP manualmente).

Programas Cliente e Servidor

A figura abaixo descreve o trabalho que deve ser feito:

1. O servidor inicia, carrega os dados do PeeringDB
2. O servidor exporta um subconjunto dos dados via uma API REST (HTTP)
3. O cliente consulta os dados do servidor através dos *endpoints* pré-definidos
4. O cliente usa os dados recuperados do servidor para gerar algumas análises



Programa Servidor

O servidor é responsável por carregar as informações do PeeringDB e responder requisições de clientes através dos *endpoints.* Ao iniciar, o servidor lê três arquivos que representam os dados do PeeringDB: `net.json` contendo informações sobre redes, `ix.json` contendo informações sobre IXPs e `netixlan.json` contendo associações entre redes, IXPs e LANs. Amostras desses arquivos especificando o formato serão disponibilizadas para vocês testarem o sistema.

Três *endpoints* usando HTTP GET precisam ser implementados pelo servidor:

1. `/api/ix:` todos os IXPs

    - Requisição ao endpoint `/api/ix`
    - Resposta:
      `{"data": <lista dos objetos IXPs>}`

2. `/api/ixnets/{ix_id}:` identificadores das redes de um IXP

- Requisição ao endpoint `/api/ixnets/{ix_id}`
- Resposta:
  `{"data": <lista dos identificadores das redes do IXP identificado por 'ix_id'>}`

3. `/api/netname/{net_id}`: nome de uma rede

   - Requisição ao endpoint `/api/netname/{net_id}`
   - Resposta:
     `{"data": <nome da rede identificada por 'net_id'>}`

Detalhes de execução e implementação

O servidor deve ser executado com a seguinte linha de comando:

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

Onde [port] é o porto no qual o servidor receberá mensagens, [Netfile] é o caminho do arquivo de redes, [Ixfile] é o caminho do arquivo de IXPs e [Netixlanfile] é o caminho do arquivo de associações.

Para o servidor (apenas) será permitido o uso de algum web framework que facilite a implementação dos endpoints. Por exemplo, para implementações em Python, sugerimos o uso do Flask. Adicionalmente, você pode utilizar uma biblioteca para manipulação de dados em formato JSON. Por exemplo, para implementações em Python, utilize a biblioteca json.

Programa Cliente

O cliente é responsável por realizar duas análises sobre os dados do PeeringDB. Diversas redes podem participar do PeeringDB a partir de vários IXPs, isso significa que a relação entre redes e IXPs é muitos-para-muitos. Essas relações estão identificadas no arquivo de associações (netixlan.json)[1].

Análise 0 (IXPs por rede)

Para essa análise o cliente deve produzir na saída padrão um **TSV UTF-8** (tabelas com células separadas por tabulações `'\t'`) com uma rede por linha, contendo as seguintes colunas (nessa ordem):

[1] Note que o arquivo de netixlan.json contém associações entre redes, IXPs e LANs, e não apenas entre redes e IXPs. Faz parte do seu trabalho contabilizar essas relações adequadamente.

1. Identificador da rede: esse dado pode ser encontrado no campo `id` do arquivo `net.json`; ou no campo `net_id` do arquivo `netixlan.json`
2. Nome da rede: esse dado pode ser encontrado no campo `name` do arquivo `net.json`
3. Número de IXPs associados à rede: deve ser gerado pelo seu cliente

Exemplo de saída:

```
2    Akamai Technologies    153
3    DALnet IRC Network    17
4    Limelight Networks Global    80
...
```

Análise 1 (redes por IXP)

Para essa análise o cliente deve produzir na saída padrão um **TSV UTF-8** (tabelas com células separadas por tabulações `'\t'`) com uma rede por linha, contendo as seguintes colunas (nessa ordem):

4. Identificador do IXP: esse dado pode ser encontrado no campo `id` do arquivo `ix.json`; ou no campo `ix_id` do arquivo `netixlan.json`
5. Nome do IXP: esse dado pode ser encontrado no campo `name` do arquivo `ix.json`
6. Número de redes associadas ao IXP: deve ser gerado pelo seu cliente

Exemplo de saída:

```
1    Equinix Ashburn    110
2    Equinix Chicago    82
3    Equinix Dallas    50
...
```

Detalhes de execução e implementação

Note que você **não deve incluir** o cabeçalho TSV com o nome das colunas na saída das análises e o cliente deve ser executado com a seguinte linha de comando:

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

Onde [IP] é o IP do servidor, [port] é o porto que o servidor está utilizando e [Opt] especifica o número da análise: "0" se for a análise de IXPs por rede ou "1" se for a análise redes por IXP.

Para o cliente **não será permitida a utilização de nenhum framework.** As únicas bibliotecas permitidas são: (1) sockets e (2) manipulação de formato JSON. Isso quer dizer que seu cliente precisa entender o protocolo HTTP para fazer requisições ao servidor.

Implementação e Interface com Usuário

Seus programas cliente e servidor devem interoperar com outros programas (teste com implementações de colegas) e inclusive com as versões do programa implementados pelo professor. Este trabalho pode ser implementado em Python, C, C++, Java, ou Rust, mas deve interoperar com emuladores escritos em outras linguagens.

- **Inicialização.** Os programas devem ser executados exatamente como descrito acima (com os mesmos parâmetros e na mesma ordem), para facilitar correção semiautomática. Note que o servidor e o cliente podem se comunicar através do IP 127.0.0.1 se o programa estiverem executando na mesma máquina, mas pode também ser o endereço de uma outra máquina acessível pela rede.[2]
- **Saída.** O servidor não precisa gerar nenhuma saída. O cliente gera apenas a saída no formato TSV como descrito acima. Isso quer dizer que mensagens de estado ("Cliente iniciou", "Conexão estabelecida", etc.) devem ser omitidas ou redirecionadas para a saída de erro (stderr) ao invés da saída padrão (stdin).
- **Terminação.** O servidor só termina com comando explícito do usuário (CTRL+C). O cliente deve terminar imediatamente após gerar a saída da opção de análise.

Avaliação

- **Esse trabalho é individual.**
- **Erros na especificação e implementação.** Qualquer incoerência ou ambiguidade na especificação deve ser apontada para o professor como comentário neste documento; se confirmada a incoerência ou ambiguidade, o aluno receberá pontos extras. A mesma política se aplica a erros na implementação do professor.
- **Entrega.** Seu programa deve ser entregue de forma modular e de forma que a compilação (se necessário) e execução sejam descomplicadas. Descreva na documentação o processo de compilação e execução do seu programa.

Testes

Pelo menos os seguintes aspectos serão testados/conferidos:

- Todos os endpoints do servidor funcionam adequadamente
- O cliente implementa as duas análises e gera as saídas corretamente

[2] Para determinar o endereço da máquina onde o servidor vai executar você pode usar o comando [ifconfig], ou fazer o seu servidor escrever o endereço IP da máquina onde ele está executando.

- O cliente não utiliza nenhum framework para manipulação de mensagens HTTP, apenas a API de sockets tradicional.

Documentação

Cada aluno deverá entregar documentação em PDF de até 6 páginas (3 folhas), sem capa, utilizando fonte tamanho 10, e figuras de tamanho adequado ao tamanho da fonte. A documentação deve discutir desafios, dificuldades e imprevistos de projeto, bem como as soluções adotadas para os problemas. Em particular, sua documentação deve:

- Apresentar instruções de compilação (se necessário) e execução do seu programa. Os executáveis devem ter o nome de *client* e *server*, exatamente, para fins de correção semiautomática.
- Discutir o funcionamento do cliente e do servidor, incluindo os detalhes de implementação dos endpoints e das mensagens trocadas.
- Apresentar uma breve caracterização das redes e dos IXPs PeeringDB:
    - Uma CDF da distribuição da quantidade de IXP por rede (análise 0)
    - Uma CDF da distribuição da quantidade de redes por IXP (análise 1)
    - Uma breve discussão desses resultados encontrados, isto é, discutindo como é a distribuição de associações entre redes e IXPs.