Nomes: Acquila Santos Rocha 201600709
Carlos Henrique Rorato 201600718
Juliana de Melo Teixeira 201600745

Documentação Laboratório 6

Redes de Computadores 1

1. Decisões de projeto

Este projeto visa a construção de um servidor Proxy Web, com funcionalidade de caching. O cache possui a capacidade de verificar se o objeto armazenado está **atualizado** ou **desatualizado**. Nesse servidor, é tratado a possibilidade de fazer ou não fazer cache. Caso a memória do cache esteja cheia, é utilizado a **Política LRU (Least Recently Used),** que verifica o objeto menos acessado e o retira do cache até que tenha espaço suficiente para o armazenamento.
A conexão do proxy possui "duas vias". Uma via se conecta, através de um socket TCP na porta **54321** ao cliente que faz a requisição (navegador). Outra via se conecta através de um socket TCP na porta **80**, com o servidor requisitado pelo cliente.
Como forma de atualização e registro de operações, o projeto possui um log, que informa todas os eventos ocorridos no servidor proxy. **Ex.:** `18:20:44,148 proxy INFO Requisicao de ('127.0.0.1', 33344) para http://inf.ufg.br/`

1.1 Estrutura

Todos os códigos-fonte estão estruturados e divididos em quatro arquivos, **proxy.py**, **cache.py**, **util.py**, **main.py.** Cada um deles será descrito minuciosamente nos tópicos seguintes.

1.2 Detalhes

1.2.1 main.py

Cria um objeto da classe ServidorProxy e executa ServidorProxy.start()

```
from proxy import ServidorProxy
server = ServidorProxy()
server.start()
```

1.2.2 util.py

Descrição: Este arquivo contém funções essenciais para o funcionamento do

cache. Possui métodos para o parsing das informações vindas do cliente e do servidor.

Métodos

parseHeader(data) - Recebe como argumento os dados da requisição do cliente e retorna um dicionário de dados, separado por chave e valor, todas as informações da requisição.

Ex .: `recv_data['If-Modified-Since'] = 'Thu, 09 Feb 2012 18:09:34 GMT'`

parseServerHeader(data) - Recebe como argumento todos os dados da resposta do servidor e retorna um dicionário de dados, separado por chave e valor todas as informações da requisição.

Ex .: `recv_data['status_code'] = 304`

getHttp(url, response, buffer_size) - Recebe como argumento a url do servidor, a requisição do cliente e o tamanho do buffer da string que receberá os dados do servidor. Retorna toda a resposta obtida em uma string.

notImplementedMethod(conn) - Recebe como argumento o objeto socket conn. Este método deve retornar uma resposta ao cliente dizendo que o método da requisição não é suportado pelo proxy. Caso a requisição não tenha método 'GET', ignorar a requisição.

1.2.3 cache.py

Descrição: Arquivo detentor da classe Cache, responsável por controlar as funções básicas de cache..

Classe:

Cache(size)

Métodos:

__init__(self, size) - Instância as variáveis privadas da classe.

```
self.max = size # Tamanho máximo do cache
self.actual_size = 0 # Tamanho atual do cache
self.free_space = self.max # Espaço livre do
cache
self.cache = {}
```

free(self, n) - Atualiza as variáveis de controle de tamanho de cache na situação de liberação de memória.
use(self, n) - Atualiza as variáveis de controle de tamanho de cache na situação de uso da memória de cache.
popCache(self, url) - Retira um objeto do cache baseado na chave (url), no dicionário de dados.
update(self, url, response) - Atualiza um objeto no cache (com a url passada no argumento do método), quando necessário.
getData(self, url) - Recupera um objeto do cache através da chave 'url'.
freeSpaceByLRUPolicy(self, size) - Utiliza a **Política LRU (Least Recently Used)**, para lidar com falta de espaço no cache. Irá retirar da memória de cache todos os objetos menos acessados (por **Cache.update** ou **Cache.getData**), até que tenha espaço para armazenamento do último objeto

1.2.3 proxy.py

Descrição: Arquivo principal do projeto, onde é definida a classe ServidorProxy, o método de início do sistema (**ServidorProxy.start**), e o método
ServidorProxy.requestHandler.

Classe:

ServidorProxy()

Métodos:

__init__(self) - Instancia as variáveis privadas da classe

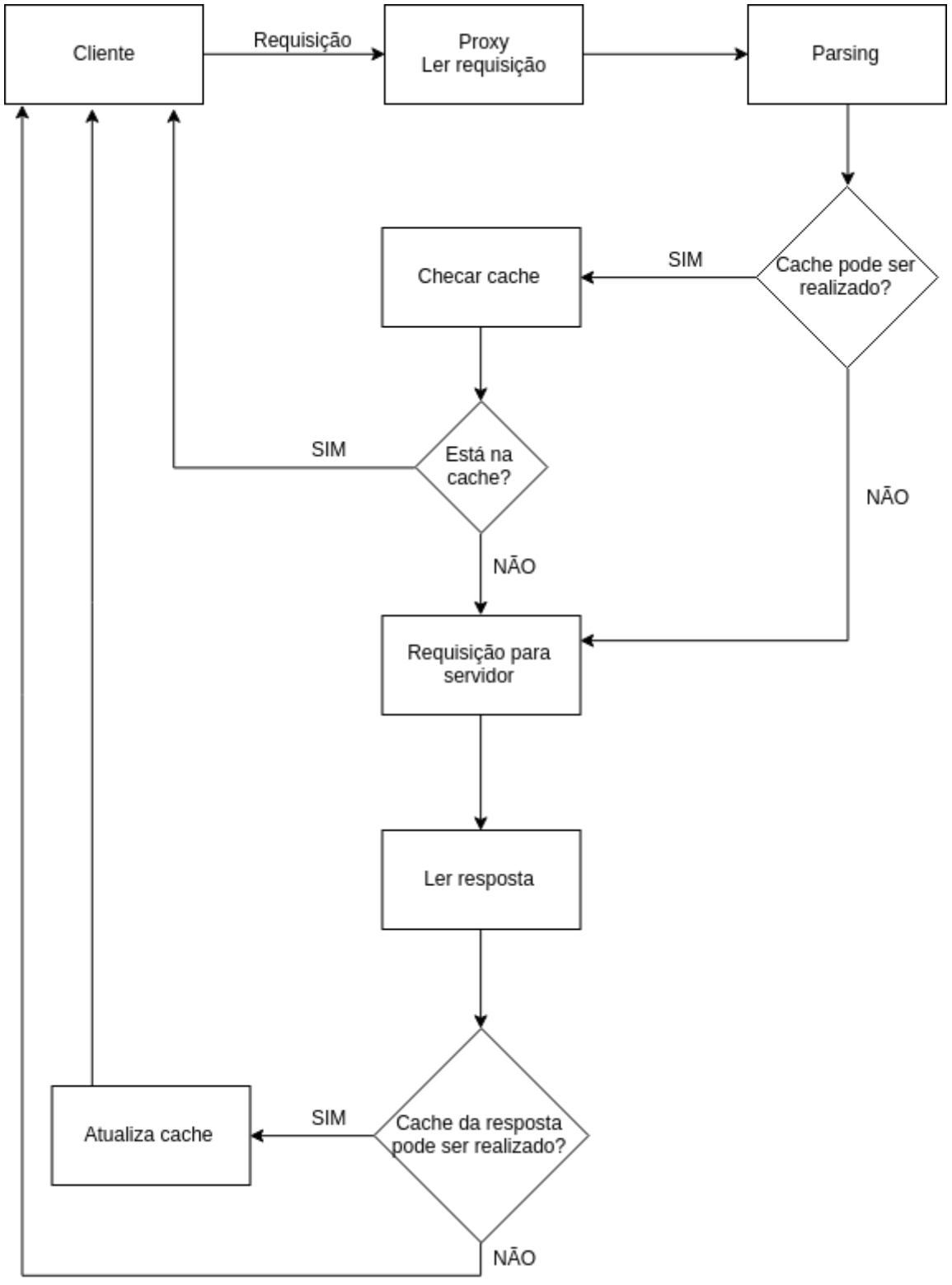
```
self.port = 54321 # Configura porta de execução do
servidor proxy
self.buffer_size = 8192 # Tamanho em bytes do buffer de
requisição
```

```
self.conexoes = 10 # Quantidade máxima de conexões
suportadas.
self.cache = Cache(999999999) # Criação do objeto de
cache
```

start(self) - Cria as threads e inicia o servidor.
requestHandler(self, data, conn, addr) - Este método funciona em complemento ao cache. Quando recebido uma requisição vinda do cliente, verifica se o objeto requisitado está no cache, caso não esteja, verifica se **Cache-Control** está configurado para 'no-cache'. Em caso positivo, recupera o objeto do servidor e envia para o cliente sem armazenar, caso contrário o objeto é armazenado.

```
if 'Cache-Control' in recv_data.keys() and recv_data['Cache-Control']
== 'no-cache':
    log.info("Cache-Control - 'no-cache'")
    response = getHttp(host, data, self.buffer_size)
else:
    response = getHttp(host, data, self.buffer_size)
    self.cache.cachePush(url, response)
```

Caso o objeto esteja em cache, é verificado se está atualizado, em caso positivo, redireciona uma resposta com status_code = 304. Caso esteja desatualizado, envia resposta com status_code = 200, o cache deve ser atualizado através de `self.cache.update`.



2. Descrição da interface de monitoramento

A interface de monitoramento consiste no registro de cada ação tomada pelo servidor no **proxy.log**. Caso dados sejam inseridos na cache, há a apresentação da quantidade de bytes que foram inseridos, juntamente com informações como tempo, nome do nível e mensagem.

3. Testes realizados

Os testes foram realizados com a finalidade de verificar o funcionamento correto do proxy com cache, com algumas possíveis situações que puderam ser utilizadas para garantir a funcionalidade. Sendo elas:

- Comportamento do servidor proxy com espaço de armazenamento de cache reduzido;
- Comportamento do servidor proxy com campo if-modified-since desatualizado, onde deve-se realizar a requisição ao servidor;
- Comportamento do servidor proxy com campo if-modified-since atualizado, onde deve-se pegar os dados diretamente da cache;
- Comportamento do servidor proxy com campo Cache-Control: no-cache, onde deve-se realizar requisição ao servidor.

Os testes apresentaram os resultados esperados na versão final do código do proxy com cache, confirmando sua corretude em relação a seu funcionamento.