Universidade de Brasília
Transmissão de Dados

Proxy Web

Mariana Pimentel Martins da Silva 14/0085360
Rafaela Sinhoroto Lima 14/0091696

26 de junho de 2017

1. Introdução Teórica

- **Proxy Web Server**

O *proxy* é um intermediário entre clientes requisitando serviços de servidores [1]. Essa relação entre o cliente, o servidor e o *Proxy* está ilustrada na figura 1. Um cliente pode conectar-se ao *proxy* para solicitar diversos serviços de vários servidores. A maioria dos *proxys* atuais são *web proxys*, ou seja, servem como um intermediário na requisição de conteúdos da rede.

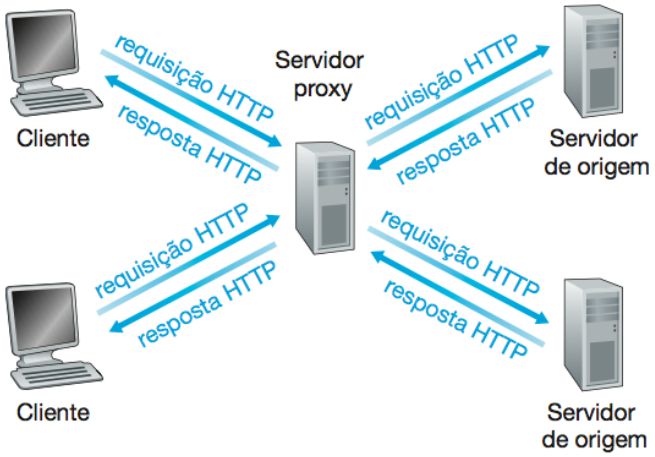

Figura 1 - Clientes requisitando serviços dos servidores por meio de um servidor *Proxy.*
(Figura retirada de: [2])

Sendo assim, o *proxy* pode até modificar os pedidos dos clientes antes de enviar ao servidor e modificar a resposta do servidor antes de enviá-la ao cliente. Com isso, torna-se possível implementar um sistema de filtragem de requisições. Além disso, outros adicionais podem ser implementados em um servidor *Proxy* como, por exemplo, um sistema de *Caching* que salvaria as respostas recebidas de alguns servidores, mandando-as diretamente para o usuário quando ele requisitar o acesso ao mesmo site. O *Proxy* também fornece anonimato para o cliente, visto que ele altera o cabeçalho da mensagem para que seu endereço esteja como endereço de origem antes de encaminhar uma mensagem para o servidor. Essas são somente algumas das diversas motivações para o uso de um servidor *proxy*.

- **TCP**

O TCP é um protocolo da Camada de Transporte que é orientado para conexão, confiável e possui controle de congestionamento. Para poder fornecer transferência confiável de dados, o TCP conta com mecanismos de detecção de erro, retransmissões, reconhecimentos cumulativos,

temporizadores e campos de cabeçalho para números de sequência e de reconhecimento. Com isso, a camada de aplicação pode contar que receberá os dados ordenados e sem erros.

Por ser um serviço orientado à conexão, antes que mensagens da camada de aplicação possam ser repassadas, o cliente e o servidor trocam informações de controle. Isso faz com que eles se preparem para se comunicar, estabelecendo os parâmetros da transferência de dados. A conexão estabelecida é full-duplex (simultânea), podendo enviar e receber mensagens ao mesmo tempo. Além disso, a aplicação deve interromper a conexão ao ser encerrada.

Por ser um serviço de entrega confiável de dados, o TCP também emprega controle de congestionamento, onde há uma limitação da capacidade de transmissão de um processo quando a rede está congestionada, e controle de fluxo, um serviço que compatibiliza a taxa à qual o remetente está enviando com aquela à qual a aplicação de destino está lendo.

- **Protocolo HTTP**

O HTTP (*HyperText Transfer Protocol* - Protocolo de Transferência de Hipertexto) é um protocolo da camada de aplicação da Web.

De maneira geral, o HTTP é executado em dois programas, um cliente e outro servidor, em sistemas finais diferentes. Os programas conversam entre si por meio da troca de mensagens HTTP. O próprio protocolo define a estrutura dessas mensagens e o modo como o cliente e o servidor as trocam.

Por exemplo, quando um usuário requisita uma página Web, o navegador (cliente) envia ao servidor mensagens de requisição HTTP para os objetos da página. O servidor recebe as requisições e responde com mensagens de resposta HTTP que contém os objetos. Esse processo está ilustrado na Figura 2.

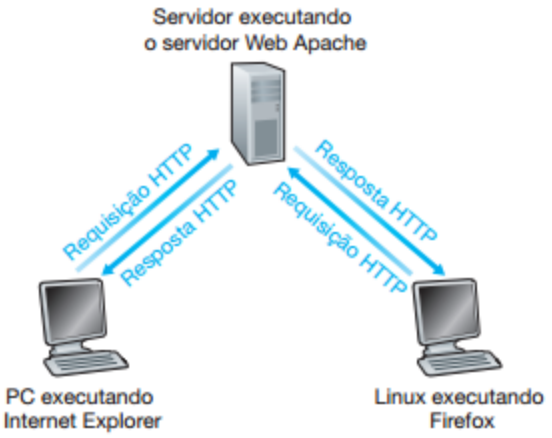

Figura 2 - Clientes enviando requisições HTTP a um servidor Web, que envia as respostas.
(Figura retirada de: [2])

O HTTP usa o TCP como seu protocolo de transporte subjacente. O cliente HTTP primeiro inicia uma conexão TCP com o servidor. Uma vez estabelecida, os processos do navegador e do servidor acessam o TCP por meio de suas interfaces de socket.

2. Arquitetura do Sistema

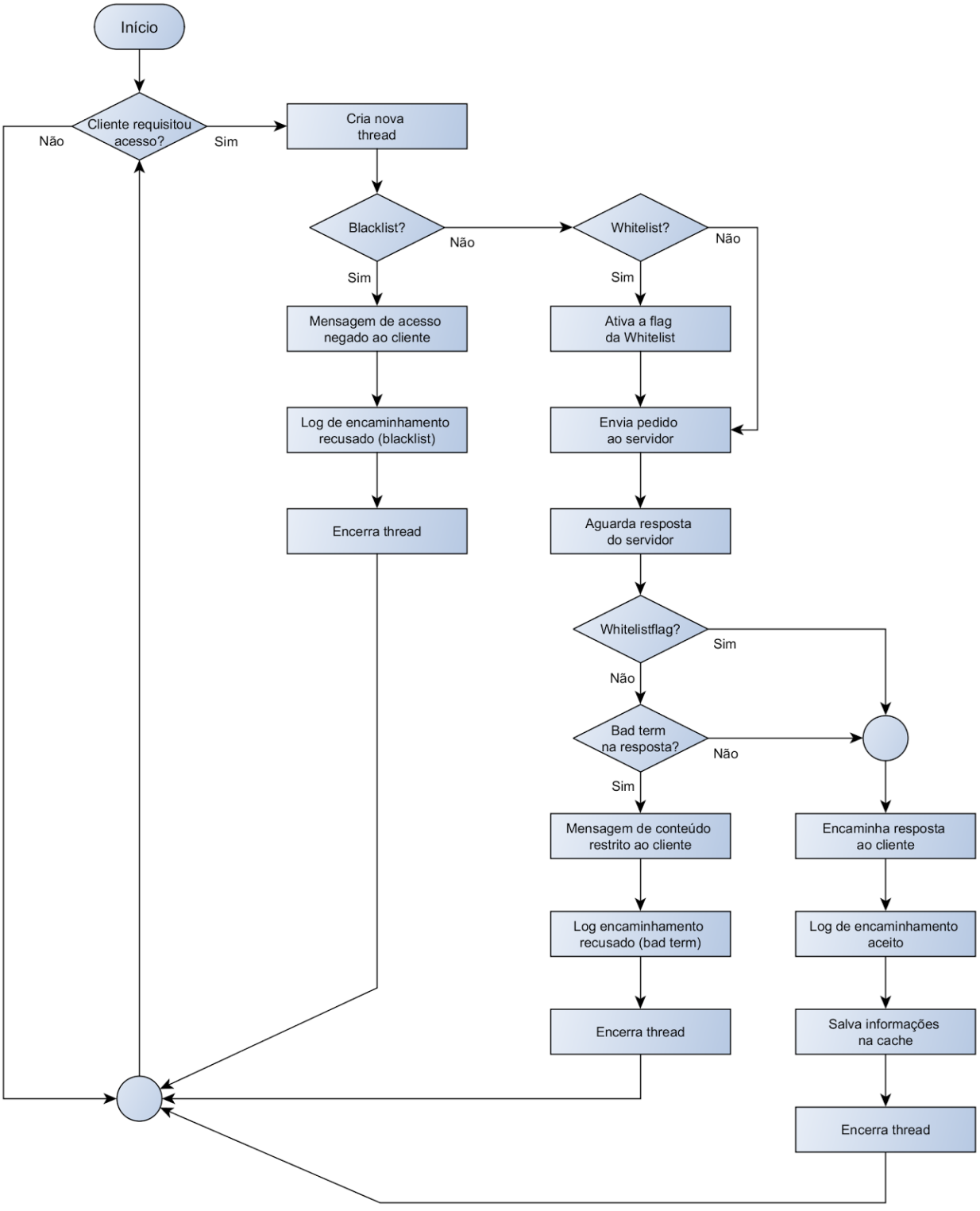

Figura 3 - Fluxograma representando a arquitetura do sistema desenvolvido.

3. Código

O código desenvolvido importa seis bibliotecas, como pode ser visto na figura 4: a *socket*, a *thread*, a *sys*, a *time*, a *string* e a *binascii*. Suas funções no código estão descritas mais detalhadamente no item 5 do relatório.

```
1   import socket
2   import thread
3   import sys
4   import time
5   import string
6   import binascii
```

Figura 4 - Importes de bibliotecas

A função *proxy* feita recebe três parâmetros que contém as informações sobre a conexão, o cliente e o pedido do cliente. São retiradas do pedido informações como a *url* completa e o domínio, utilizando a função de manipulação de strings *split()* e a função de busca *find()*.

Nessa parte inicial da função também é feita a verificação da existência da *url* requisitada nos vetores da *cache*.

```
8   def proxy (conexao, cliente, pedido):
9           try:
10                  linha = pedido.split('\n')[0] #Separa a primeira linha do pedido
11                  url = linha.split(' ')[1] #Pega a url depois do GET
12                  posicao = url.find('://') #Procura informação de protocolo HTTP ou HTTPS
13                  url_completa = url[(posicao+3):] #Pega a url sem essainformação
14                  posicao = url_completa.find('/') #Encontra a primeira barra para extrair o domínio
15                  if posicao == -1:
16                          posicao = len(url_completa)
17                  porta = 80
18                  url_raiz = url_completa[:posicao]
19                  urlCacheadaFlag = False
20                  for j in range (0, len(urls)): #Verifica se o objeto já foi rquisitado alguma vez e
21                          if url_completa == urls[j]:
22                                  urlCacheadaFlag = True
23                                  break
24          except Exception, e:
25                  pass
```

Figura 5 - Início da função proxy (primeiro *try*)

Sabendo a *url* e o domínio que o cliente deseja acessar, o código compara este último com os *sites* presentes na *blacklist*. Caso o mesmo esteja na lista, o servidor envia uma mensagem de “Acesso Negado” para o *browser*, encerra a conexão e gera uma entrada no arquivo *log* de encaminhamento recusado, com a informações de data, hora, *url* requisitada e tipo de bloqueio (*blacklist*).

```
27      for i in range(0, len(blacklist)): #Percorre o vetor da blacklist e procura a url rquisitada nele
28              if blacklist[i] in url: #Se a url está na blacklist, envia mensagem de acesso negado ao navegador e encerra a thread
29                      conexao.send(str.encode("HTTP/1.1 400 Bad Request\nContent-Type:text/html\n\n<html><body> Acesso Negado! </b
30                      if conexao:
31                              conexao.close()
32                      arq=open('log.txt', 'a') #Salva no log como "encaminhamento recusado" com a data, a hora e o motivo
33                      arq.write(time.strftime('%d/%m/%Y %H:%M:%S')+" - Encaminhamento Recusado: "+url+" presente na blacklist\n")
34                      arq.close()
35                      sys.exit(1)
```

Figura 6 - Loop da blacklist

Em seguida, é feito o percorrimento do *loop* da *whitelist.* Procura-se a *url* requisitada no vetor de *sites* da *whitelist* e, caso encontre, ativa-se uma *flag* que indica que o *site* está autorizado a ser acessado.

```
37          WhiteListFlag = False
38          for i in range(0, len(whitelist)): #Percorre o vetor da whitelist e procura
39                  if whitelist[i] in url: #Se a url está na whitelist, aciona a flag c
40                          WhiteListFlag = True
```

Figura 7- Loop da whitelist

Após o processamento da *whitelist*, o programa tenta iniciar uma conexão com o servidor *Web*, enviando o pedido do cliente e aguardando a resposta. Em caso de mais de um pacote de resposta (quando a resposta do servidor *Web* exceder 1024 bits), os pacotes recebidos são concatenados para formar um pacote maior com a resposta total.

Enquanto são recebidos os pacotes dos sites não marcados na *whitelist* (ou seja, se a *flag* da *whitelist* não estiver ativada), analisa-se o conteúdo deles para verificar se está presente algum dos termos restritos. Caso o programa encontre, a conexão é finalizada, é enviada uma mensagem de "Conteúdo Restrito" ao navegador e é gerada uma entrada no *log* de encaminhamento recusado, com a informações de data, hora, *url* requisitada e tipo de bloqueio (*bad terms*).

```
internet = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #Inicia o socket que vai comunicar com o servidor web
internet.connect((url_raiz, porta)) #Conecta com o domínio na porta 80
internet.send(pedido) #Envia o pedido para o servidor e aguarda resposta

resposta_final = ""
while True:
        if urlCacheadaFlag: #Se esse site já foi acessado, pega informações da cache e segue em frente
                resposta_final = respostas[j]
                break
        resposta = internet.recv(maximo_dados) #Pega a resposta do servidor ao pedido
        if resposta: #Se existe uma resposta:
                resposta_final += resposta #Concatena à uma parte anterior da resposta (resposta_final começa vazia)
                resposta_analise = resposta.upper() #Coloca tudo em maiúsculo para deixar insensível ao caso (bloqueia term
                if (resposta_analise.find("CONTENT-TYPE: TEXT/HTML") != -1): #Analisa apenas se for um pacote com conteúdo
                        print "Resposta: ", resposta_analise, "\n"
                        if not WhiteListFlag: #Se estiver na whitelist, pular a verificação de termos proibidos
                                for i in range(0, len(termo)): #Para cada termo do vetor de termos bloqueados, procurar na
                                        posicao = str.find(resposta_analise, termo[i])
                                        print "\nTermo: ", termo[i], "\n"
                                        print "Posicao: ", posicao, "\n\n\n"
                                        if posicao!=-1: #Se encontrou um termo proibido, envia mensagem de conteúdo restrit
                                                conexao.send(str.encode("HTTP/1.1 400 Bad Request\nContent-Type:text/html\n
                                                if internet:
                                                        internet.close()
                                                if conexao:
                                                        conexao.close()
                                                arq=open('log.txt', 'a') #Salva no log como "encaminhamento recusado" com a
                                                arq.write(time.strftime('%d/%m/%Y %H:%M:%S')+" - Encaminhamento Recusado: "
                                                arq.close()
                                                sys.exit(1)
```

Figura 8 - *Loop* da conexão com o servidor *web* requisitado

Se não foi encontrado nenhum dos termos restritos, a resposta do servidor é adicionada na *cache* e enviada ao navegador. Finaliza-se a conexão e é gerada uma entrada no *log* de encaminhamento autorizado, com a informações de data, hora e *url* requisitada.

```
if not urlCacheadaFlag: #Se o objeto não estiver na cache ainda, aidicona-lo ao final do v
        urls.append(url_completa) #Sua url
        respostas.append(resposta_final) #Seu conteúdo
conexao.send(resposta_final) #Se não foi barrado por blacklist ou bad terms, envia respost
internet.close()
conexao.close()
arq=open('log.txt', 'a') #Salva no log como "encaminhamento aceito" com a data e a hora
arq.write(time.strftime('%d/%m/%Y %H:%M:%S')+" - Encaminhamento Autorizado: "+url+"\n")
arq.close()
sys.exit(1) #Encerra a thread
```

Figura 9 - Fim da thread de conexão

Na figura 10, pode-se ver algumas das variáveis utilizadas. O IP está definido como o *localhost*, a porta configurada para o *Proxy* é a posta 2045 e a porta para conexão com o servidor é a porta 80. A quantidade máxima de dados recebida por pacote é configurada como 1024 e permite até 100 conexões pendentes. Além disso, nessa figura também pode ver a abertura e leitura dos arquivos *blacklist.txt*, *whitelist.txt* e *badterms.txt.*

```
98   IP = '127.0.0.1'
99   porta_proxy = 2045
100  porta_destino = 80
101  maximo_dados = 1024
102  conexao_pendente = 100
103  arq=open('blacklist.txt', 'r') #Abre, lê e separa em li
104  blacklist_string = arq.read()
105  blacklist = blacklist_string.split('\n')
106  arq.close()
107  arq=open('whitelist.txt', 'r') #Abre, lê e separa em li
108  whitelist_string = arq.read()
109  whitelist = whitelist_string.split('\n')
110  arq.close()
111  arq=open('badterms.txt', 'r') #Abre, lê e separa em lin
112  termo_string = arq.read()
113  termo_string = termo_string.upper()
114  termo = termo_string.split('\n')
115  arq.close()
116  urls = [] #Cria dois vetores globais para o caching
117  respostas = []
118
```

Figura 10 - Seção de declaração de variáveis globais, abertura de arquivos, etc.

Na Figura 11 está apresentada a parte final do código. Após as declarações e inicializações das variáveis necessárias, tenta-se estabelecer o socket que irá realizar a comunicação entre o *proxy* e o cliente (navegador). Esse servidor será inicializado na porta número 2045 e no IP *localhost* para uso em único computador, ou no IP do computador que servirá o papel de servidor em uma LAN.

Após estabelecido o servidor, é iniciada uma conexão ao aceitar o pedido de conexão do navegador. Com essa conexão, o programa recebe o primeiro pacote de 1024 bits do pedido. Se esse pacote não está vazio, inicia-se a nova thread que irá realizar a filtragem do acesso ao servidor *Web* requisitado.

```
try:
        servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #Inicia o servidor Proxy
        servidor.bind((IP, porta_proxy)) #Funcionando no IP 'localhost' e na porta 2045
        servidor.listen(conexao_pendente)
except socket.error, (valor, mensagem): #Mensagem de erro caso não consiga iniciar a conexão
        if servidor:
                servidor.close()
        print "Socket indisponivel: ", mensagem
        sys.exit(1)

while True:
        (conexao, cliente) = servidor.accept() #Aceita o pedido de conexão do navegador
        print 'Conectado por: ', cliente[0], ":", porta_proxy
        try:
                pedido = conexao.recv(maximo_dados) #Recebe pacote de dados de até 1024 bits
                if not pedido: #Se não obteve o pedido encerra
                        break
                thread.start_new_thread( proxy, (conexao, cliente, pedido, ) ) #Começa uma nova
        except KeyboardInterrupt: #Ao encerrar com o ctrl+C, fecha o servidor e a conexão
                if servidor:
                        servidor.close()
                if conexao:
                        conexao.close()
                print '\nFinalizando conexao do cliente ', cliente
                sys.exit(1)
```

Figura 11 - Fim do programa principal

4. Instruções de Compilação/Execução

Para compilar e executar o código é necessário alterar as configurações de rede do computador ou do *browser.* O *Web Proxy (HTTP)* deve ser ativado com o servidor 127.0.0.1 (*localhost*) e com a porta 2045.

Após finalizar essas configurações, é possível compilar e executar o código normalmente utilizando o comando: python proxy.py.

Se desejar fazer o teste para múltiplos usuários deve-se alterar a variável IP do código para que possua o IP do computador em que o código será executado. As configurações de todos os computadores deverão estar com o *Web Proxy (HTTP)* ativado com o servidor configurado como o IP do computador em que o código será executado e com a porta 2045.

Então, o código poderá ser compilado e executado com o mesmo comando citado anteriormente: python proxy.py.

5. Relação das Bibliotecas

A tabela 1 mostrada abaixo mostra as bibliotecas utilizadas e a contribuição de cada uma delas para o funcionamento do código.

Tabela 1 - Relação das bibliotecas utilizadas

Biblioteca Utilizada	Motivo da inclusão
socket	Para realizar a conexão entre o servidor proxy, os clientes (navegadores) e os servidores web requisitados, utilizando portas e sockets.
thread	Para poder realizar a conexão simultânea de vários usuários (mais de um dispositivo final) e clientes (mais de um navegador por dispositivo)
sys	Para incluir a chamada *sys.exit()*, que encerra uma thread quando a conexão é terminada ou quando há alguma interrupção (programada ou devido a erros)
time	Para aquisição das informações de data e hora atuais para gerar o arquivo de *log* dos encaminhamentos aceitos ou recusados
binascii	Para utilizar a função *str.encode()* para enviar a mensagem de “acesso negado” ou “conteúdo restrito” para o cliente que teve sua requisição recusada
string	Para utilizar as funções de manipulação e busca nas strings do programa

6. Referências Bibliográficas

[1] CAETANO, Marcos F.; VALDY JÚNIOR, José. **Transmissão de Dados – Turma A – Trabalho 1:** Implementação de uma aplicação de *Proxy Server*. Brasília: Unb, 2017.

[2] KUROSE, James F.; ROSS, Keith W. **Redes de Computadores e a Internet.** São Paulo: Pearson, 2006.