Implementação de uma Aplicação de Proxy Sever

Eduardo Scartezini
14/0137084
Departamento de Ciência da Computação
Instituto de Ciências Exatas

Andrei Buslik
11/0024702
Departamento de Ciência da Computação
Instituto de Ciências Exatas

***Abstract*—Este relatório é parte do trabalho final do segundo semestre de 2017 da matéria de Teleinformatica e Redes da Universidade de Brasília. Contém alguns conceitos teóricos, a maior parte deles visto em sala de aula, bem como uma explicação mais detalhada de nosso servidor *proxy Web*, que possui filtro de conteúdo e *cache* de páginas *web*.**

***Keywords*—Redes de computador, dados, HTTP, proxy, filtro, cache.**

I. Apresentação Teórica

A. Proxy Web

Um servidor *proxy* é um tipo de servidor, ou seja, um sistema ou uma aplicação que atua como um intermedário entre clientes e servidores. A relação ocorre da seguinte forma: um cliente conecta-se a um servidor proxy e faz uma requisição. O servidor proxy analisa esta requisição, alterando o endereçdo cliente para o seu próprio, e a envia para um determinado servidor. Ao receber a resposta do mesmo, pode alterá-la ou não, e, em seguida, reenvia para o cliente.

Geralmente, essa intermediação ocorre entre um cliente de uma certa rede local e o servidor de uma rede externa, como a Internet, conforme mostra a figura abaixo.

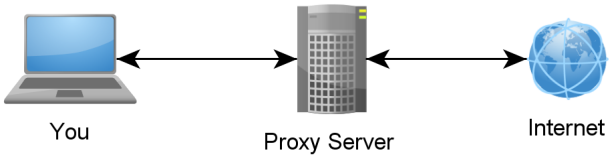

Fig. 1. Atuação de um servidor proxy. Retirada do site: https://okhosting.com/resources/uploads/2016/05/proxy-server.png

É importante salientar que o *proxy* garante anonimato ao cliente, pois, ao invés de seu endereço IP ficar registrado na cache da página de destino, fica registrado o endereço IP do próprio servidor *proxy*.

B. TCP

O Protocolo de Controle de Transmissão (em inglês, *Transmission Control Protocol*) é um dos protocolos mais importantes da Camada de Transporte, quarta camada segundo o Modelo OSI ou terceira camada segundo o modelo TCP/IP. Ela é responsável por receber dados da Camada de Aplicação, verificar sua integridade, separá-los em pacote e, então, transmiti-los para a camada abaixo, a Camada de Rede.

Uma de suas principais características é ser orientado à conexão, ou seja, há uma troca de pacotes entre cliente e servidor, ditos pacotes de controle, antes de serem trocados os pacotes de dados. Dessa forma, são estabelecidos parâmetros para a conexão.Este procedimento ocorre por meio do *handshake* ou *three-way handshake*.

Conforme a figura abaixo, o *handshake* se inicia quando o cliente envia um segmento de sincronização (*SYN*), que contém um número inicial de sequência, ao servidor. O servidor, ao receber o *SYN*, envia uma mensagem de aceite da conexão (*SYN ACK*). Junto dele, é enviado o número inicial de sequência do servidor. Por fim, o cliente confirma o recebimento do aceite enviando um segmento de *acknolodgment* (*ACK*).

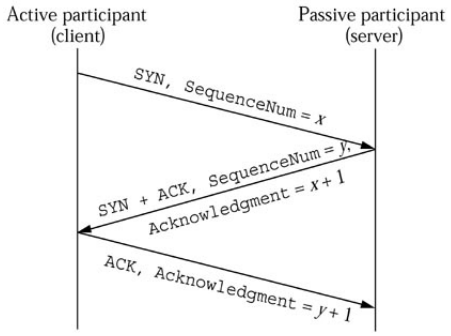

Fig. 2. Como ocorre o *handshake*. Retirada do site: http://ww.wiki.hping.org/122

Já para encerrar a conexão, um dos *hosts*, neste caso, o cliente, envia um segmento *FIN* solicitando o término da conexão. O *host* B, neste caso, o servidor, envia dois segmentos: um *ACK*, indicando que confirma o recebimento do segmento, e, então, também envia um *FIN*. Quando o *host* A enviar um segmento *ACK*, a conexão estará oficialmente terminada, conforme mostra a figura abaixo.

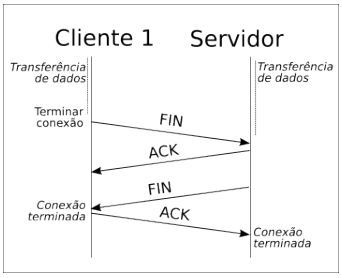

Fig. 3. Finalizando a conexão. Retirada do site: https://upload.wikimedia.org/wikipedia/commons/7/75/TCP_termination.png

C. Protocolo HTTP

O Protocolo de TransferÍncia de Hipertexto (em inglês, *HyperText Transfer Protocol*) é um exemplo de protocolo da Camada de Aplicação, a última camada de ambos os modelos citados acima. É ele quem permite a comunicação de dados entre redes de computadores, principalmente com a World Wide Web, conhecida popularmente pelo acrônimo WWW.

Seu funcionamento ocorre por meio de requisições de clientes, que desejam um determinado recurso, e de respostas de servidores, que contêm solicitações.Em suas requisições, o cliente envia um cabeçalho que o identifica e outras informações, entre elas, o método utilizado na requisição.Podemos entender melhor observando o exemplo abaixo.

```
GET / HTTP/1.1
Host: spesa.com.br
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US)
Gecko/20061201 Firefox/2.0.0.3 (Ubuntu-feisty)
Accept:
text/xml,application/xml,application/xhtml+xml,text/
html;q=0.9,text/plain;q=0.8,image/png
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

Fig. 4. Exemplo de requisição feita por um cliente. Retirada do site: https://nandovieira.com.br/entendendo-um-pouco-mais-sobre-o-protocolo-http

O servidor, então, envia sua resposta, que contém não só o conteúdo HTML da página desejada, mas também um cabeçalho de resposta. Nele, a informaçõ mais importante encontrada é o código de resposta, que vai indicar se a requisição foi concluída com sucesso, se ela não é autorizada, se não foi encontrada, entre outras opções.

```
HTTP/1.x 200 OK
Date: Fri, 04 May 2007 16:05:43 GMT
Server: Apache/2.0.59 (Unix) mod_ssl/2.0.59 OpenSSL/0.9.7a DAV/2
PHP/4.4.4 mod_bwlimited/1.4
Cache-Control: no-cache
Keep-Alive: timeout=3, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1
```

Fig. 5. Exemplo de resposta feita por um servidor. Retirada do site: https://nandovieira.com.br/entendendo-um-pouco-mais-sobre-o-protocolo-http

Uma última característica importante desse protocolo que ele dito *stateless*, isto é, ele não retém as informações quando recebe mais de uma requisição, sendo necessário utilizar-se de outras formas para guardar as informações desejadas.

II. Arquitetura do Sistema

Nosso *proxy* foi desenvolvido para conter em arquivos distintos as funcionalidades que deveriam ser implementadas como requisitos deste trabalho. Dessa forma, ele contém:

- Uma pasta *src* contendo a *main.c* e os arquivos *socket.c*, *decoder.c* e *cache.c*, com o escopo das funções utilizadas para realizar, respectivamente, a comunicação cliente-servidor, a filtragem e o *caching*;
- Uma pasta *include* contendo os arquivos *socket.h*, *decoder.h* e *cache.h*, locais em que se encontram o cabeçalho das funções utilizadas nas funcionalidades acima;
- Uma pasta *resources/filter*, onde se encontram o arquivo com os domínios autorizados (*Whitelist*), o arquivo com os domínios não-autorizados (*Blacklist*) e o arquivo com os termos proibidos (*Deny_terms*), utilizados pela filtragem;
- Uma pasta *www* com as páginas HTML que são retornadas na mensagem de resposta ao cliente após a etapa de filtragem;
- Um makefile, com as regras de compilação;
- Um README, com as intruções necessárias para rodar o arquivo.

As linguagens escolhidas para a implementação deste trabalho foram C e C++.

III. Documentação do Código

A documentação do código se encontra presente no mesmo.

IV. Funcionalidades Implementadas

Como mencionado anteriormente, nosso trabalho foi pensado de forma modularizada conforme as três etapas propostas.

A. Funcionamento Básico

Como atua como um *gateway*, isto , um dispositivo que conecta redes, o *proxy Web* recebe a mensagem de um determinado *host* e lê as informações presentes no cabealho da requisição - como o método, o *path*, a versão do procolo IP, a página desejada- e as guarda. Depois, altera o cabeçalho inicial, de forma que o *host* de origem passe a ser o próprio *proxy*.

Para realizar estas ações, foram criados os arquivos ”socket.h” e ”socket.c”. Eles contêm, respectivamente, o cabeçalho das funções utlizadas para receber, ler e encaminhar os pacotes recebidos por nosso *proxy* e o desenvolvimento das mesmas.

Funcções desenvolvidas são:

- *openSocket*, que define um socket para o servidor;
- *request*, que recebe a requisição feita pelo cliente;
- *stablishConnection*, que estabelece a conexão com o cliente;
- *dnsResolve*, que salva o nome e o endereço do *host*;
- *recv_timeout*, responsável por garantir que as respostas cheguem dentro do tempo necessário;
- *time2string* e *timesystem*, responsàveis por coletar o tempo exato do sistema e das requisições.

Também foram desenvolvidas funções que utilizam *threads*, como

- *in_thread*, relacionada com a criação de *threads* e com a escuta do canal;
- *start_thread*, que inicia a *thread*.

Além disso, também são utilizadas funções relacionadas à filtragem e ao *caching*, uma vez que elas também dependem de informações adquiridas nesta etapa. Portanto, são feitas chamadas a elas pelo arquivo ”socket.c” e, consequentemente, o mesmo arquivo recebe seus retornos.

Exemplos de funções relacionadas aos tópicos abaixo:

- *decodeHTTP*, *decodeHTTP* e *logMessage*, da parte de filtragem;
- *readCache* e *writeCache*, da parte de *caching*.

Estas funções sero explicadas nos tópicos abaixo.

B. Filtragem de Requisições

Em nosso trabalho, o *proxy* checa se a página que o cliente requisitou acessar encontra-se na *Whitelist* ou na *Blacklist*. Caso o domínio não se encontre em nenhuma destas listas, é necessário checar se há, no conteúdo da página, alguma palavra presente em uma terceira lista, a de *Deny_Terms*.

Para isso, foram criados os arquivos ”decoder.h” e ”decoder.c” com as funções:

- *decodeHTTP*, responsável por decodificar o cabeçalho presente na requisição;
- *filterHost*, que busca nas listas de domínios autorizados e não-autorizados a p’agina que o cliente deseja acessar;
- *filterTerms*, que busca se há alguma palavra do conteúdo que esteja na lista de termos proibídos;
- a *makeReqModified*, que modifica a requisição;
- *grepHttpCode*, que realizada o *grep*, um comando de busca do Linux.

C. Caching

A *cache* o local de armazenamento de páginas HTTP frequentemente utilizadas. Elas são armazenadas para que os próximos acessos a elas levem menos tempo para serem realizados.

Em nosso trabalho, as funções ligadas à cache estão localizadas nos arquivos ”cache.c” e ”cache.h”. Neles se encontram algumas funções citadas anteriormente e outras não, como:

- *readCache* e *writeCache*, que, respectivamente, são as funções responsáveis pela leitura e escrita na *cache*;
- *inCache*, que abre o arquivo da cache;
- *fileName*, que guarda as informações presentes nos cabeçalhos das requisições.

V. Conclusão

Este relatório relaciona conceitos teóricos aprendidos nas aulas de Teleinformatica e Redes II do semestre 02/2017, ministradas pelo Professor MSc. João Gondim. Apresenta-se, ainda, o servidor *Proxy Web* desenvolvido pelos alunos Eduardo Scartezini e Andrei Buslik como trabalho final para esta disciplina, incluindo aspectos como a arquitetura do sistema e funcionalidades implementadas.

References

[1] Kurose, James F. Computer networking: A top-down approach featuring the internet, 3/E. Pearson Education India, 2005.

[2] TecMundo. *O que é um Proxy?*. Sítio mundial da rede de computadores: https://www.tecmundo.com.br/navegador/972-o-que-e-proxy-.htm. (Acesso em 18/06/2017)

[3] What Is My IP Address. *What is a Proxy Server?*. Sítio mundial da rede de computadores: http://whatismyipaddress.com/proxy-server. (Acesso em 18/06/2017)

[4] IP Location. *What is a proxy server?*. Sítio mundial da rede de computadores: https://www.iplocation.net/proxy-server. (Acesso em 18/06/2017)

[5] CCM. *O protocolo TCP*. Sítio mundial da rede de computadores: http://br.ccm.net/contents/284-o-protocolo-tcp. (Acesso em 18/06/2017)

[6] TecMundo. *O que TCP/IP?*. Sítio mundial da rede de computadores: https://www.tecmundo.com.br/o-que-e/780-o-que-e-tcp-ip-.htm. (Acesso em 18/06/2017)

[7] Vieira, Nando. *Entendendo um pouco mais sobre o protocolo HTTP*. Sítio mundial da rede de computadores: https://nandovieira.com.br/entendendo-um-pouco-mais-sobre-o-protocolo-http. (Acesso em 18/06/2017)

[8] Podila, Pavan. *HTTP: The Protocol Every Web Developer Must Know - Part 1*. Envato Tuts+. Sítio mundial da rede de computadores: https://code.tutsplus.com/tutorials/http-the-protocol-every-web-developer-must-know-part-1–net-31177. (Acesso em 18/06/2017)

[9] Tecmundo. *O que é Cache?*. Sítio mundial da rede de computadores: https://www.tecmundo.com.br/navegador/201-o-que-e-cache-.htm