Available online at www.sciencedirect.com

ScienceDirect

Procedia Engineering 69 (2014) 128 – 132

Procedia Engineering

www.elsevier.com/locate/procedia

24th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2013

An Educational HTTP Proxy Server

Martin Sysel*, Ondřej Doležal

Department of Computer and Communication Systems,Faculty of Applied Informatics, Tomas Bata University in Zlín, nám. T. G. Masaryka 5555, 760 01 Zlín, Czech Republic

Abstract

The efficiency and safety of Web access can be enhanced by the deployment of an http proxy server in many cases. The first part of this paper provides an introduction to the issue of an HTTP proxy server. The second part of the paper describes used technologies and an implementation of a multithreaded HTTP proxy server with an embedded WWW server used for the graphics user interface. In its current state, the developed proxy server can be used to monitor the WWW traffic of a local area network and, with further development of its functionalities, can include such areas as content filtering or access control.

Keywords: Proxy server; Http; Socket; Thread

1. Introduction

The global computer network Internet, from its beginnings, expanding rapidly in the 70th and 80th of the 20th century. There are connected to a network more than 1.67 billion users now. The most common use of end users access to the Internet is WWW - World Wide Web. This is so dominant that even in normal communication can be traced merging concepts of Internet and WWW. The reasons for such popularity are more - architecture enabling seamless collaboration between completely heterogeneous systems, intuitive user interface and also the ability to create complex applications.

The actual World Wide Web can be characterized as a distributed client-server information system with thin client, transmitting information in the form of HTML pages and other objects using HTTP application protocol [3, 4]. It is a typical protocol of request-response, which controls the data transfer between server and client (such as a

* Corresponding author. Tel.: +420 57 603 5180; fax: +420 57 603 5279.
E-mail address: Sysel@fai.utb.cz

doi:10.1016/j.proeng.2014.02.212

web browser). HTTP traffic that is point-to-point communication but there are a lot of use cases where this communication can benefit from inclusion of additional active element - proxy server [6].

Proxy server is the middle element in communication between the server and the client. This element may only redirects communication or checks the application protocol. HTTP proxy server may accelerate access to resources and perform inspection or monitoring operations. Protect the privacy of users can be provided too [9].

The aim is to analyze the requirements for Internet HTTP proxy server, specify the ways of their solution and create application program design that will implement HTTP proxy in GNU Linux including necessary documentation. The program is created using free software and product itself is published under an open source license GNU GPL.

2. Proxy Server

A proxy server is usually a computer system - a combination of hardware platforms and software applications - which serves as an intermediary in the network communication between the parties. The client-server systems provide an intermediary in the communication between the client (usually sending requests) and the server (sending the response), see Fig. 1. Generally, to the proxy is not limited for client-server systems, suitable proxy can provide the exchange of messages in networks such as peer-to-peer.

The basic principle of operation of the proxy server is to receive client requests (against which looks like a server), these requirements are analyzed and then send the target servers (to whom they are acting as a client), and then the answers to pass the original client - in original or modified form. The proxy server operates on the 7th layer ISO / OSI model (application) to analyze incoming requests [1, 2]. Therefore, it's also called this proxy as an application proxy. Proxy server works with the same application protocol such as serviced clients. Operation with various protocols can be achieved by different proxy servers, or multi-protocol servers.

In addition to this application proxy, there are also application-independent ones, providing only transport packets without the knowledge of application layer protocols. Their using however requires the use of specific communication protocol to communicate with the proxy server. A typical representative of the universal proxy protocol is SOCKS protocol working on the 5th layer of ISO/OSI model of the session. SOCKS routes network packets between a client and server through a proxy server. SOCKS5 additionally provides authentication so only authorized users may access a server [1]. Deployment SOCKS proxy client application requires adaptation - modification of the network code. But there are client implementations that after running redirect network traffic to the client SOCKS proxy in the protocol, eliminating the need to modify the client code.

2.1. Reasons for using the application proxy

The primary reason for deploying of the first proxy server: allow access to external sources of computer facilities inside the firewall-protected network or otherwise directly inaccessible. Proxy server in this case is installed on the computer with a firewall, and serves as a gateway for intermediating network traffic of the application protocol. A similar effect - serving resources - can be achieved in the presence of a suitable firewall with a rule opened for outbound traffic; then communication between the client and the server is routed normally [5, 6].

An application proxy offers next functionality [8, 9]:

- In most cases, more clients can access to proxy server. All responses to requests pass through proxy server, and if the proxy server stores the contents of the answers can improve response times and reduce bandwidth to repeat. Requests use the stored response from the previous identical requests. Such a proxy server is called a caching proxy. Most of the HTTP proxy servers implement this functionality. The validity of stored responses is very important; this issue is described by HTTP protocol specification [3, 4].
- Proxy server allows processing of the finer requirements. Generally, the organization can restrict access to individual client computers by destination address, protocol or type of resource. Specialized HTTP proxy servers also support a time limit within a day or rate control, which can reduce inefficient using of bandwidth during working hours or using for improper purposes.

- Filtering proxy can be used to detect and block malicious content. Again thanks processing at the application level proxy server can perform scanning incoming content and block access to the infected sources. Similarly, the inspection of outgoing data for viruses and generally known malware can be done.
- Modification of the content of other platforms, such as access to WWW resources from mobile device. Application proxy can perform dynamic recompression to reduce data flow transcription content to skip unsupported components, etc. It is possible to combine functionality with caching and save the results to avoid their recurrence.
- Increased security can be achieved by using an application proxy server for logging and audit client requests, as opposed to logging on lower layers enables logging at the application layer easy access to all of the client / server transaction property.
- Application proxy server may in appropriate cases, make transfers between different protocols on the client side and the server side. In practice, it works as a translator at the level of application protocols. Clients can access the resources or client software programmer saves many lines of code.
- Last but not least, the proxy server can be used for anonymous access to the target server. This effect can be used to bypass website restrictions applicable to the relevant source address of the client, the appropriately configured proxy also mask the client system attributes such as the type and version of software, etc.

3. An Implementation of HTTP proxy

The practical part of the paper is an implementation of a simple educational HTTP proxy server processing incoming requests in separate threads. The implementation is realized in the programming language C++. The program was created and is working in a Linux environment.

Server implements full support for standard HTTP 1.0, ie methods GET, POST, and HEAD, and supports a subset of the HTTP 1.1 standard to allow seamless communication of an existing implementations of client and server (ie, Web browsers and Web servers) [3, 4].

Implementation of proxy server also contains the HTTP server, which implements the graphics user interface of proxy server. This interface allows the surveillance of communication proxy via WWW browser. It contains overviews of the overall server status, status of individual threads and overview of recent requests to HTTP requests, including the result of their execution. In addition to this overview, proxy server also records complete record of all requests to the log files, separately for proxy subsystem and the HTTP server subsystem.

3.1. System architecture

The application is programmed as a multi-threaded application. Threads are implemented by producer-consumer model [7]. One thread creates (producer) jobs (incoming requests) and inserts them into the queue, Fig. 1. The new jobs in the queue are processed by own thread (consumer). This processing is performed in parallel for as many requests as there are currently running consumer threads.

After starting function main() it is opened socket accepting new connections and its launched the loop containing blocked waiting for the new arrival communication. New incoming communication is checked if it is not exceed the maximum limit of connections, and if not, new thread is created with an entry point handleClient() and the new socket is created. Otherwise, the main thread waits to releasing a free thread. It is also updated counter of free threads.

Fig. 1. Proxy Server communication.

Working thread first receives a client request, it tries to decode using function parseRequest() and, if it is successful (it is correct and supported request), handle response from the remote server by calling the GetResponse() function. This response then returns to the client, sync updated information for using of thread and then thread is finished.

3.2. Memory requirements

Server uses a minimum of global memory allocated on the heap. Individual threads use a stack (in the current version of glibc fixed-size 2MB per thread), and a dynamically allocates memory from the heap as needed. To reduce memory consumption, HTTP communications between the client and the target server is solved by interleaving. It does not wait for retrieving of the whole client's request (containing requirements for HTTP entities such as POST) or whole responses of proxy server, but this communication is processed and forwarded immediately after incoming communication. The proxy server avoids the need to allocate memory for a potentially big transmitted communications and makes it with a small buffer (about 2 kilobytes) in real time (depending on the specifics of the used TCP/IP subsystem and network). This mechanism also has a positive effect on the speed of response, because the client receives a reply before the proxy server receives a whole message.

Total memory requirement of an HTTP proxy process is by default about 23 megabytes (for 10 threads) after starting the application.

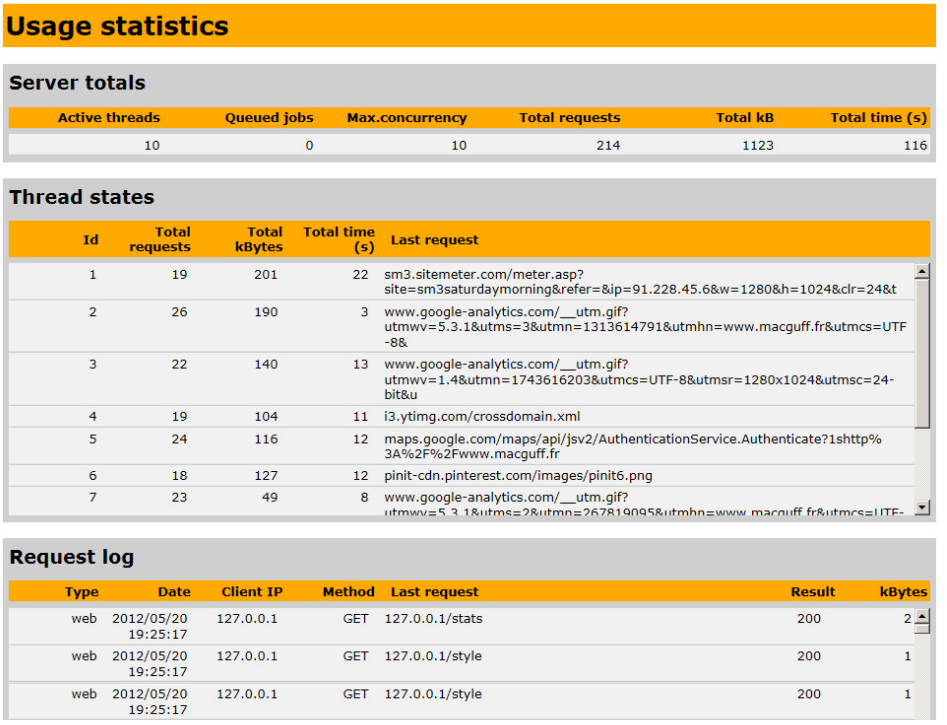
3.3. CPU requirements

Server is effective in terms of processor time consumption. This effectiveness is achieved in particular by eliminating the active waiting. Proxy server realizes all operation of the network input and output as blocking state. Receiving new connections in the main thread in the process, reading requests and outgoing responses of connected sockets in each thread – everything uses blocking state.

As with most network applications, even when proxy server is the main bottleneck of bandwidth network connection – typical current processors are capable process data much faster than the network can provide data. The application spends waiting for the arrival of additional data or further connections most of the time. When testing an application which consisted of processing 10,000 requests, it was detected by tool gprof total CPU load only 2.5%.

3.4. Logging and statistics

Proxy server collects statistical information of operations and stores a record of completed requests in log files and memory buffer that is used to generate web pages with traffic reports, see Fig. 2. All records are written to log files - separately for the Web part, and separately for proxy requests. Statistical information includes the sum of the total number of requests, the sum of transferred data and the lasting time of the operation, and further details of the final processing of this request. The information is recorded for each thread and also for the entire application.



Usage statistics

Server totals

Active threads	Queued jobs	Max.concurrency	Total requests	Total kB	Total time (s)
10	0	10	214	1123	116

Thread states

Id	Total requests	Total kBytes	Total time (s)	Last request
1	19	201	22	sm3.sitemeter.com/meter.asp?site=sm3saturdaymorning&refer=&ip=91.228.45.6&w=1280&h=1024&clr=24&t
2	26	190	3	www.google-analytics.com/__utm.gif?utmwv=5.3.1&utms=3&utmn=1313614791&utmhn=www.macguff.fr&utmcs=UTF-8&
3	22	140	13	www.google-analytics.com/__utm.gif?utmwv=1.4&utmn=1743616203&utmcs=UTF-8&utmsr=1280x1024&utmsc=24-bit&u
4	19	104	11	i3.ytimg.com/crossdomain.xml
5	24	116	12	maps.google.com/maps/api/jsv2/AuthenticationService.Authenticate?1shttp%3A%2F%2Fwww.macguff.fr
6	18	127	12	pinit-cdn.pinterest.com/images/pinit6.png
7	23	49	8	www.google-analytics.com/__utm.gif?utmwv=5.3.1&utms=2&utmn=267819095&utmhn=www.macguff.fr&utmcs=UTF-

Request log

Type	Date	Client IP	Method	Last request	Result	kBytes
web	2012/05/20 19:25:17	127.0.0.1	GET	127.0.0.1/stats	200	2
web	2012/05/20 19:25:17	127.0.0.1	GET	127.0.0.1/style	200	1
web	2012/05/20 19:25:17	127.0.0.1	GET	127.0.0.1/style	200	1

Fig. 2. Proxy Server GUI – statistic.

4. Conclusion

This work proved that this developed educational HTTP proxy server is a useful piece of software. It can be used to accelerate access to resources, perform access control or traffic logging. It can be also used to enhance anonymity of its users. Nowadays, one of the most used HTTP proxy servers is the Squid proxy cache. Further discussion was targeted towards technologies necessary for building current HTTP proxies - sockets and threads. A multi-threaded HTTP proxy server in C language was created using discussed technologies which incorporates an embedded HTTP server used to access the runtime information and usage statistics. Target platform was GNU/Linux. Further testing of this server proved that a multi-threaded design is very useful for server applications, and that the created proxy server neither uses significant amounts of system resources nor it degrades the WWW performance in an important way. In its current state, the proxy server can be used to monitor the WWW traffic of a local area network and is suitable for student education. Further work on this project will be given to extending the functionality, as content filtering or access control.

References:

[1] RFC 791. Internet Protocol. [s.l.] : IETF, 1981. 45 s. on-line : http://tools.ietf.org/html/rfc791.
[2] RFC 793. Transmission Control Protocol. [s.l.] : IETF, 1981. 85 p. on-line: http://tools.ietf.org/html/rfc793.
[3] RFC 1945. Hypertext Transfer Protocol - HTTP/1.0. [s.l.] : IETF, 1996. 60 s. on-line: http://tools.ietf.org/html/rfc1945.
[4] RFC 2616. Hypertext Transfer Protocol - HTTP/1.1. [s.l.] : IETF, 1999. 176 s. on-line: http://tools.ietf.org/html/rfc2616.
[5] RFC 2617. HTTP Authentication: Basic and Digest Access Authentication. [s.l.] : IETF, 1999. 34 s. on-line: http://tools.ietf.org/html/rfc2617.
[6] RFC 3143. Known HTTP Proxy/Caching Problems. [s.l.] : IETF, 2001. 32 s. on-line: http://tools.ietf.org/html/rfc3143.
[7] GAY, Warren. Linux Socket Programming by Example. [s.l.] : Que, 2000. 557 p. ISBN 0789722410.
[8] Dolezal, Ondrej. An HTTP Proxy Server. [s.l.] : UTB in Zlin, 2012.
[9] Books, LLC, General Books LLC. Proxy Servers: Proxy Server, Wingate, Tor, Proxomitron, Proxy Auto-Config, Java Anon Proxy, Sun Java System Web Proxy Server, Web Cache, General Books LLC, 2010, ISBN 1156716780.
[10] Ubuntu, Squid - Proxy Server - Official Ubuntu Documentation, 2012, on-line: https://help.ubuntu.com/lts/serverguide/squid.html.