APD Load Balancer

Mihaela Ancuta Bornea

June 2004

Abstract

In the last decades, the boom of the information and communication industry is due to increasing need of people to access data accurately and fast. The latter led to the increasing development of media such as newspapers and TV. Another source of info that received huge popularity is the Internet. Internet provides huge amount of data hosted on web servers. However, since time is a critical issue, the speed to access the data is as vital as the data itself. For example during a life transmission from a sport event, or in case of a popular web page, the web server that hosts this data can not face the amount of requests and the users will not receive the information they need. Therefore, the web model must provide organized information in such a way that people need of speed is met. The current application aims to satisfy that need and increase the responsiveness of a web server. This work has as result providing a user friendly and inexpensive implementation to balance the load between servers and increase user satisfaction while reducing response time. Contrary to the simplicity of of the method the achieved goals are more than satisfactory.

This paper contains two parts: one about general aspects related to web model and possible solutions to increase the responsiveness of an web server and the second part will present the solution that was implemented.

Contents

Chapter 1

www and load balancing

1.1 General information

The Internet is a network of networks, linking computers to computers sharing the TCP/IP protocols. Each runs software to provide or "serve" information and/or to access and view information. The Internet is the transport vehicle for the information stored in files or documents on another computer. It can be compared to an international communications utility servicing computers. It is sometimes compared to a giant international plumbing system. The Internet itself does not contain information. It is a slight misstatement to say a "document was found on the Internet." It would be more correct to say it was found through or using the Internet. What it was found in (or on) is one of the computers linked to the Internet.

Computers on the Internet may use one or more of the following Internet services:

1. Electronic mail: Permits us to send and receive mail.
2. Telnet, remote login: Permits logging onto another computer and use it as if one was there .
3. FTP (File Transfer Protocol) Allows a computer to rapidly retrieve complex files intact from a remote computer and view or save them on the disk.
4. The World Wide Web (WWW or "the Web"). The largest, fastest growing activity on the Internet.

The WWW incorporates all of the Internet services above and much more. One can retrieve documents, view images, animation, and video, listen to sound files, speak and hear voice, and view programs that run on practically any software in the world, providing the computer has the hardware and software to do these things.

As more and more businesses have turned to using Web-based applications for everything from generating customer sales and communications to sales applications, the spotlight has increased on IP networks to provide reliable and timely communication links to support mission-critical applications.

With the resultant boom in TCP/IP traffic generated by these applications, new network traffic management devices have emerged, supplying smart content switching capabilities that monitor systems and distribute incoming traffic for optimal response.

1.2 Browser

When one logs onto the Internet a browser is used in order to view documents on the World Wide Web. A browser is a program that resides on a computer enabling people to use the computer to view WWW documents and access the Internet, taking advantage of text formatting, hypertext links, images, sounds, motion, and other features. Netscape and Internet Explorer are currently the leading "graphical browsers" in the world (meaning they facilitate the viewing of graphics such as images and video and more). There are other browsers (e.g., Macweb, Opera). Most offer many of the same features and can be successfully used to retrieve documents and activate many kinds of programs. Browsers all rely on "plug-ins" to handle the fancier files one finds on the Web. Plug-ins are sub-programs stored within a browser or elsewhere in one's computer especially to support special types of files one may click on. If a link is clicked, and the computer does not currently have the plug-in needed for the file that was clicked, the user is usually given the opportunity to get the plug-in. Most plug-ins are free, and easy and safe to install on a computer; one has to follow the given instructions.

The main way in which browsers differ is in the convenience features they offer for navigating and managing the Web and all the URLs someone may want to keep track of. Netscape and Internet Explorer both offer the ability to e-mail documents, download them to diskette, print them, and keep track of where someone has been and sites someone wants to "bookmark."

The current foundation on which the WWW functions is the programming language called HTML. It is HTML and other programming embedded within HTML that make possible Hypertext. Hypertext is the ability to have web pages containing links, which are areas in a page or buttons or graphics on which one can click the mouse button to retrieve another document into the computer. This "click-ability" using Hypertext links is the feature which is unique and revolutionary about the Web.

1.3 HTTP Protocol

HTTP is the network protocol of the Web. It is both simple and powerful. HTTP stands for Hypertext Transfer Protocol. It's the network protocol used to deliver virtually all files and other data (collectively called resources) on the World Wide Web, whether they are HTML files, image files, query results, or anything else. Usually, HTTP takes place through TCP/IP sockets. A browser is an HTTP client because it sends requests to an HTTP server (Web server), which then sends responses back to the client. The standard (and default) port for HTTP servers to listen on is 80, though they can use any port.

The Internet's communication protocol, HTTP, has allowed distributed client/server computing at unprecedented levels. Powerful applications for the exchange of information and transaction processing have emerged and been successfully implemented by manufacturing and services companies alike.

In this world of Internet computing, Web browsers provide the on-ramp to access applications distributed across multiple sites and servers. The ubiquity of Web browsers on enterprise desktops combined with industry-wide standardization around the HTTP communications protocol has encouraged enterprises to invest in developing and/or porting applications that utilize data stored in databases

onto the Web using a wide range of Web languages such as HTML, DHTML, Java, CGI and Active Server Pages. Running on top of TCP, HTTP is a connectionless protocol allowing it to thrive over the Internet; HTTP is used to transmit resources, not just files. A resource is some piece of information that can be identified by a URL (it's the R in URL). The most common kind of resource is a file, but a resource may also be a dynamically-generated query result, the output of a CGI script, a document that is available in several languages, or something else.

1.3.1 Structure of HTTP transactions

Like most network protocols, HTTP uses the client-server model: An HTTP client opens a connection and sends a request message to an HTTP server; the server then returns a response message, usually containing the resource that was requested. After delivering the response, the server closes the connection (making HTTP a stateless protocol, i.e. not maintaining any connect on information between transactions). The format of the request and response messages are similar, and English-oriented. Both kinds of messages consist of:

1. an initial line
2. zero or more header lines,
3. a blank line (i.e. a CRLF by itself), and
4. an optional message body (e.g. a file, or query data, or query output).

Initial request line The initial line is different for the request than for the response. A request line has three parts, separated by spaces:

- a method name
- the local path of the requested resource
- the version of HTTP being used.

A typical request line is:

GET /path/to/file/index.html HTTP/1.0

GET is the most common HTTP method; it says "give me this resource". Other methods include POST and HEAD. Method names are always uppercase.

The path is the part of the URL after the host name, also called the request URI (a URI is like a URL, but more general).

The HTTP version always takes the form "HTTP/x.x", uppercase

Initial response line The initial response line, called the status line, also has three parts separated by spaces:

- the HTTP version,
- a response status code that gives the result of the request, and
- an English reason phrase describing the status code.

Typical response lines are:

HTTP/1.0 200 OK

or

HTTP/1.0 404 Not Found

The HTTP version is in the same format as in the request line, "HTTP/x.x".

The status code is meant to be computer-readable. The status code is a three-digit integer, and the first digit identifies the general category of response:

The reason phrase is meant to be human-readable, and may vary.

- 1xx indicates an informational message only
- 2xx indicates success of some kind
- 3xx redirects the client to another URL
- 4xx indicates an error on the client's part
- 5xx indicates an error on the server's part

The most common status codes are:

- 200 OK The request succeeded, and the resulting resource (e.g. file or script output) is returned in the message body.
- 404 Not Found The requested resource doesn't exist.
- 301 Moved Permanently
- 302 Moved Temporarily
- 303 See Other (HTTP 1.1 only) The resource has moved to another URL (given by the Location: response header), and should be automatically retrieved by the client. This is often used by a CGI script to redirect the browser to an existing file.
- 500 Server Error An unexpected server error. The most common cause is a server-side script that has bad syntax, fails, or otherwise can't run correctly.

Header lines Header lines provide information about the request or response, or about the object sent in the message body. The header lines are in the usual text header format, which is: one line per header, of the form "Header-Name:value", ending with CRLF.

It's the same format used for email and news postings defined in RFC822

As noted above, they should end in CRLF. The header name is not case-sensitive (though the value may be). Any number of spaces or tabs may be between the ":" and the value. Header lines beginning with space or tab are actually part of the previous header line, folded into multiple lines for easy reading. Thus, the following two headers are equivalent:

Header1: some-long-value-1a, some-long-value-1b

HEADER1: some-long-value-1a,
some-long-value-1b

HTTP1.0 defines 16 headers, though none are required.

HTTP1.1 defines 46 headers, and one (Host:) is required in requests.

For Net-politeness, the following headers should be included in the requests:

- The From: header gives the email address of whoever's making the request, or running the program doing so. (This must be user-configurable, for privacy concerns.)
- The User-Agent: header identifies the program that's making the request, in the form "Program-name/x.xx", where x.xx is the (mostly) alphanumeric version of the program. For example, Netscape 3.0 sends the header "User-agent: Mozilla/3.0Gold".

These headers help webmasters troubleshoot problems. They also reveal information about the user. When someone decides which headers to include, he must balance the webmasters' logging needs against the users' needs for privacy.

When writing servers, the following headers should be included in the responses:

- The Server: header is analogous to the User-Agent: header: it identifies the server software in the form "Program-name/x.xx". For example, one beta version of Apache's server returns "Server:Apache/1.2b3-dev".
- The Last-Modified: header gives the modification date of the resource that's being returned. It's used in caching and other bandwidth-saving activities. The Greenwich Mean Time, in the format Last-Modified: Fri, 31 Dec 1999 23:59:59 GMT should be used.

The message body An HTTP message may have a body of data sent after the header lines. In a response, this is where the requested resource is returned to the client (the most common use of the message body), or perhaps explanatory text if there's an error.

In a request, this is where user-entered data or uploaded files are sent to the server. If an HTTP message includes a body, there are usually header lines in the message that describe the body.

In particular:

- The Content-Type: header gives the MIME-type of the data in the body, such as text/html or image/gif.

- The Content-Length: header gives the number of bytes in the body.

1.3.2 HTTP Exchange—Example

To retrieve the file at the URL http://www.somehost.com/path/file.html

First a socket is opened to the host www.somehost.com, port 80 (the default port 80 is used because none is specified in the URL).Then, something like the following is sent through the socket:

GET /path/file.html HTTP/1.0
From: someuser@jmarshall.com
User-Agent: HTTPTool/1.0
[blank line here]

The server should respond with something like the following, sent back through the same socket:

HTTP/1.0 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/html
Content-Length: 1354
<html>
<body>
<h1>Happy New Millennium!</h1>
(more file contents)
.
.
.
.
</body>
</html>

After sending the response, the server closes the socket.

1.4 HTTP Proxies

An HTTP proxy is a program that acts as an intermediary between a client and a server. It receives requests from clients, and forward those requests to the intended servers. The responses pass back through it in the same way.

Thus, a proxy has functions of both a client and a server.

Proxies are commonly used in firewalls, for LAN-wide caches, or in other situations.

When a client uses a proxy, it typically sends all requests to that proxy, instead of to the servers in the URLs. Requests to a proxy differ from normal requests in one way: in the first line, they use the complete URL of the resource being requested, instead of just the path.

For example, GET http://www.somehost.com/path/file.html HTTP/1.0

That way, the proxy knows to which server to forward the request to (though the proxy itself may use another proxy).

1.5 Secure protocols for WWW

SSL, short for Secure Socket Layer, is a protocol developed by Netscape for transmitting private documents via Internet. SSL works by using a private key to encrypt data that's transferred over the SSL connection. Both Netscape Navigator and Internet Explorer support SSL, and many Web sites use the protocol to obtain confidential user information, such as credit card numbers. By convention, URLs that require an SSL connection start with https: in stead of http:.

Another protocol for transmitting data securely over the Word Wide Web is Secure HTTP (S-HTTP). Whereas SSL creates a secure connection between a client and a server, over which any amount of data can be sent securely, S-HTTP is designed to transmit individual messages securely. SSL and S-HTTP, therefor, can be seen as complementary rather than competing technologies. Both protocols have been approved by the Internet Engineering Task Force (IETF) as a standard.

1.6 Load balancing in Web Applications

1.6.1 Overview

Once Internet evolved, browsers have emerged as the interface of choice to access services and information. The ubiquity of the Internet has made it the backbone of choice to connect businesses with their suppliers, partners, and customers. The Internet is experiencing explosive growth. It is estimated that the number of Internet users will grow.

The majority of these users will be using browsers to access some sort of information or service provided through a Web site. It is estimated that Web-related traffic accounts for over 80% of all Internet usage.

The popularity of the Web model is causing corporations to Web-enable their core business processes such as sales, customer service, and marketing, making Web sites an integral part of the their mission-critical infrastructures.

These business processes are accessed by customers, suppliers and partners around the world through the Internet 24 hours a day. The ease of access made possible by the Internet has significantly increased the number of simultaneous client requests that servers at these Web sites must support. Furthermore, the number of requests a server needs to support at any particular time is unpredictable. Sometimes a piece of information suddenly becomes popular or "hot" and results in large spikes in the number of requests that need to be serviced.

If a site does not have the capacity to handle the requests, in a real-word environment, when the number of users accessing the web site increases, the users would experience slow response times or connections being refused. The site may even fail under critical load conditions. Negative experiences such as these can result in loss of customers, revenue, and opportunity.

When it comes to handling lots of visitors, high-volume sites have learned that the actual quality of service a Web server provides to end users typically depends on two parameters

- network-transfer speed

- server-response time.

Network-transfer speed is mainly a matter of Internet-link bandwidth. Server-response time depends upon resources: fast CPU, RAM, and good I/O performance (especially for disk and network traffic).The question is what should someone do when these resources are exhausted and the Web server is struggling against heavy traffic? More RAM could be installed on existing machines, or perhaps the CPU should be replaced with a faster one. Faster or dedicated SCSI controllers and disks with shorter access times (perhaps a RAID system with a huge cache) could also be used. Simply increasing the processing power and other resources on the server may not be cost effective anymore. The administrator must provide scalability for the environment and ensure that it has the required availability and provides better performance.

Operating-system parameters and Web-server software can be configured to achieve better performance.

A different approach in addressing this problem is to improve performance by increasing the number of Web servers.This involves an attempt to distribute the traffic onto a cluster of back-end Web servers. This is an interesting approach, because the back-end servers don't need to be large-scale machines –medium-scale hardware works just fine.

Server load balancers can be instrumental in ensuring that the e-business customers have a satisfactory experience when visiting a site.

A popular way to implement a web site is to combine a server load balancer with several servers running identical applications with access to identical content.

The server load balancer distributes client requests across the servers, improving server efficiency. The multiple redundant servers scale the capacity of the site and ensure its availability.

Load Balancing is distributing processing and communications activity evenly across a computer network so that no single device is overwhelmed. Load balancing is especially important for networks where it's difficult to predict the number of requests that will be issued to a server. Busy Web sites typically employ two or more Web-servers in a load balancing scheme. If one server starts to get swamped, requests are forwarded to another server with more capacity.

Load balancing can also refer to the communications channels themselves.

Assume there are N back-end servers available, named "www.loadbalancedsiteX.com" (where X is the number of server), all offering the same service, and the cluster approach will be used to solve the resource problem. The goal then is to balance the traffic (addressed to www.loadbalanced.com) onto these available servers so that the technical distribution is totally transparent to the end user. The Web-site visitors are not directly confronted with the fact that their requests are being served by more than one machine. They never see the underlying distribution. This is important both for backward compatibility and to avoid problems (for instance, bookmarking pages, or a back-end server crash). The new Web cluster should behave identically to the old single-machine approach.

1.6.2 Clustering

The use of multiple servers to implement a service is referred to as clustering. Clustering improves a service's responsiveness by improving its overall capacity and by improving its scalability. Each server contributes with its capacity to the overall capacity of the service. As user demand increases, servers can be added to transparently keep up with it.

Another important characteristic of clustering is redundancy. If a server fails or is taken out of service for maintenance, the other servers can continue to handle user requests, improving the availability of the service. Load balancers must continuously monitor the health of the servers which make up a cluster. If one of the servers goes out of service, the load balancer must stop forwarding requests to it until it is back in service.

To balance server load, the system distributes requests to different nodes within the server cluster, with the goal of optimizing system performance. This results in higher availability and scalability necessities in an enterprise Web-based application.

Server farms today contain a variety of machines with different performance characteristics and applications. Maximizing utilization of each server requires information about both the server and the incoming client request for services.

SLB (Server Load Balancers) is the general term assigned to techniques intended to provide:

- maximum server utilization,
- high overall availability to applications,
- transparent load distribution across networked servers to make them seem as one to user,
- manageability for servers that need to be removed from service and returned to operations following maintenance.

1.7 Characteristics of the load balancing system

1.7.1 Availability

As enterprises rely on Web-enabled applications to communicate and transact with their customers, these applications by definition become mission critical. Obstacles to accessing applications due to server latency, downtime or errors are therefore unacceptable. A widely cited statistic concerning e-commerce transactions shows why: response times greater than 7 seconds result in loss of 30% of prospective customers. For other applications such as on-line banking, an manufacturing systems, slow response times leads to significant business disruption that is not acceptable. The ability for Web switches to monitor server health and performance and respond by redirecting requests to better performing servers is a crucial component to Web-enabled applications.

High availability can be defined as redundancy. If one server cannot handle a request, can other servers in the cluster handle it? In a highly available system, if a single Web server fails, then another server takes over, as transparently as possible, to process the request.

Multiple machine configuration eliminates a given application server process as a single point of failure.

The idea of having multiple servers (potentially on multiple machines) naturally leads to the potential for the system to provide fail-over. That is, if any one machine or server on the system were to fail for any reason, the system should continue to operate with the remaining servers. The load balancing property should ensure that the client load gets redistributed to the remaining servers, each of witch will take on a proportionately slightly higher percentage of the total load. Of course, such an arrangement assumes that the system is designed with some degree of overcapacity, so that the remaining servers are indeed sufficient to process the total expected client load.

Ideally, the fail-over aspect should be totally transparent to clients of the system. When a server fails, any client that is interacting with that server should be automatically redirected to one of the remaining servers, without any interruption of service and without requiring any special action on the part of that client. In practice, however, most fail-over solutions may not be completely transparent. For example, a client that is currently in the middle of an operation when a server fails, may receive an error from that operation, and may require to retry (at which point the client will be connected to another still-available server). Or the client may observe a delay in processing, before the processing of his request resumes automatically with a different server.

The important point in fail-over is that each client, and the set of clients as a whole, is able to eventually continue to take advantage of the system and receive useful service, even if some servers fail and become unavailable.

When a previously-failed server is repaired and again becomes available, the system may, transparently, start using that server again to process a portion of the total client load.

The fail-over aspect is sometimes called fault tolerance, in that it allows the system to survive a variety of failures or faults. It should be noted, however, that fail-over is only one technique in the much broader field of fault tolerance, and no such technique can make a system 100 percent safe against every possible failure. The goal is to greatly minimize the probability of system failure, but keep in mind that the possibility of system failure can not be completely eliminated.

1.7.2 Scalability

Scalability is an application's ability to support a growing number of users. If it takes an application 10 milliseconds(ms) to respond to one request, how long does it take to respond to 10,000 concurrent requests?

Infinite scalability would allow it to respond to those in 10 ms; in the real world, it's somewhere between 10 ms and a logjam.

Scalability is a measure of a range of factors, including the number of simultaneous users a cluster can support and the time it takes to process a request. The proposed configurations should allow the overall system to service a higher client load than that provided by the simple basic configuration. Ideally, it should be possible to service any given load simply by adding the appropriate number of servers or machines.

In addition to new application development, organizations have invested in upgrading their servers,

installing additional servers and implementing redundant systems, hoping to provide their customers with faster response times, leading to better overall customer satisfaction. But introducing multiple servers brings forth additional requirements such as quickly propagating address changes when bringing servers offline for maintenance and doing so transparently to users. Effective server management requires control of IP addressing where multiple virtual IP (VIP) addresses are being used for server farms. VIP addresses are translated to registered IP addresses before traffic is routed to the public network, thereby providing virtually unlimited availability to non-registered (private) addresses for adding servers to the server farm. The ability to implement VIPs also provides quicker network updates by eliminating the need to re-address a network that requires public network interaction.

1.7.3 Maintainability

While maintainability is somewhat related to availability, there are specific issues that need to be considered when deploying a topology that is maintainable. In fact, some maintainability factors are at cross purposes for availability. For instance, ease of maintainability would dictate that one should minimize the number of application servers instances in order to facilitate on-line software upgrades. Taken to extreme, this would result in a single application server instance, which, of course would not provide a high availability solution. In many cases, it is also possible that a single application server instance would not provide the required throughput or performance.

Some of the maintainability aspects that are considered:

- Dynamic changes of configuration
- Mixed configuration
- Fault isolation

Dynamic changes of configuration In certain configurations, it may be possible to modify the configuration on the fly without interrupting the operation of the system and its service to clients. For example, it may be possible to add or remove servers to adjust to variations in the total client load. Or it may be possible to temporary stop one server to change some operational or tuning parameters, then restart it and continue to service client requests. Such characteristics, when possible, are highly desirable, since they enhance the overall manageability and flexibility of the system.

Mixed configurations In some configurations, it may be possible to mix multiple versions of a server or application, so as to provide for stage deployment and a smooth upgrade of the overall system from one software or hardware version to another. Coupled with the ability to make dynamic changes to the configuration, this property may be used to effectuate upgrades without any interruption of service.

Fault isolation In the simplest applications fail-over means that someone is concerned with clean failures of individual server, in which a server simply ceases to function completely , but this failure

has no effect on the other servers. However, there are sometimes situations where one malfunctioning server may in turn create problems for the operation of other, otherwise healthy servers. For example, one malfunctioning server may hoard system resources, or database resources, or hold critical shared objects for extended periods of time, and prevent other servers from getting their fair share of access to this resources or objects. In this context, some configurations may provide a degree of fault isolation, in that they reduce the potential for the failure of one server to affect other servers.

1.7.4 Session state

Unless there is only one application server or the application is completely stateless, maintaining state between HTTP clients will also pay a factor in determining the configuration. Two methods for sharing of sessions between multiple application server processes (cluster members) will be presented.

- One method is to persist the session to a database.
- An alternate approach is to use memory-to-memory session replication functionality .

The memory-to-memory replication (sometimes also referred to as "in-memory replication") is faster than the database persistence, and eliminates a single point of failure found in the session database. The memory-to-memory replication of persistent sessions will perform better on average compared to database persistence since there is no dependency on disk I/O associated with the reads and writes of the database persistence solution.

1.7.5 Security

The potential to distribute the processing responsibilities between multiple servers and, in particular, multiple machines, also introduces a number of opportunities for meeting special security constraints in the system. Different servers or types of servers may be assigned to manipulate different classes of data or perform different types of operations. The interactions between the various servers may be controlled, for example through the use of firewalls, to prevent undesired accesses to data.

1.7.6 Workload management

For multinational organizations that have data centers in multiple locations, managing traffic load between sites takes on additional complexity. Directing users requests to sites must not limit the number of sites available to users.

Worldwide data center traffic management requires intelligent information about site load, server health, and even requesting client location.

Workload management is the process of spreading multiple requests for work over the resources that can do the work. It optimizes the distribution of processing tasks.

Workload management is also a procedure for improving performance, scalability and reliability of an application. It provides fail-over when some servers are not available. It is most effective when used in systems that contain servers on multiple machines. It can also be used in systems that contain

servers on a single, high-capacity machine. In either case, it enables the system to make the most effective use of the available computing resources.

The proposed configuration should ensure that each machine or server in the configuration processes a fair share of the overall client load that is being processed by the system as a whole. In other words, it would not do for one machine to be overloaded while another machine is mostly idle.

If all machines are all roughly the same power, each should process a portion of the load. If various machines are on different power, each should process a portion of the load in proportion to its relative processing power. Furthermore, if the total load changes over time, the system should naturally adapt itself to maintain this load balancing property, regardless of the actual total magnitude of this load. In other words, all machines may be used at 100% of their capacity.

Workload management can provide the following benefits to an web balancing application:

- It balances client processing requests , allowing incoming work requests to be distributed according to a configured selection policy
- It provides fail-over capability by redirecting client requests to a running server when one or more servers are unavailable. This improves the availability of the application and administrative services.
- It enables systems to be scaled up to serve a higher client load than provided by the basic configuration. With clusters and cluster members, additional instances of servers can easily be added to the configuration
- It enables servers to be transparently maintained and upgraded while applications remain available for users.
- It centralizes administration of application servers.

The HTTP requests can be distributed efficiently across multiple Web servers using workload management policies.

1.8 Approaches to WEB load balancing applications

There are several methods to load balancing in web application server clusters. Of the many methods available to balance a server load, the main two are:

- DNS round robin
- Hardware balancers.

1.8.1 DNS round robin

The Domain Name Server (DNS) database maps host names to their IP address. Here it is exploited the fact that the first step a browser has to take to retrieve the URL www.loadbalancedsite.com is

to resolve the corresponding IP address for www.loadbalancedsite.com. This is accomplished with a passive resolver library that calls a nearby DNS server, which then actively iterates over the distributed DNS server hierarchy on the Internet until it reaches the last DNS server, which finally gives the IP address. Instead of giving a static address for "www.loadbalancedsite.com", the DNS server gives the address of one of the back-end Web servers. Which one depends on the scheme used for balancing the traffic and the technical possibilities available for this decision.

When an URL is typed in a browser (say, "www.loadbalancedsite.com"), the browser sends a request to the DNS asking it to return the IP address of the site. After the Web browser gets the IP address for that site, it contacts the site using the IP address, and displays the page for the client.

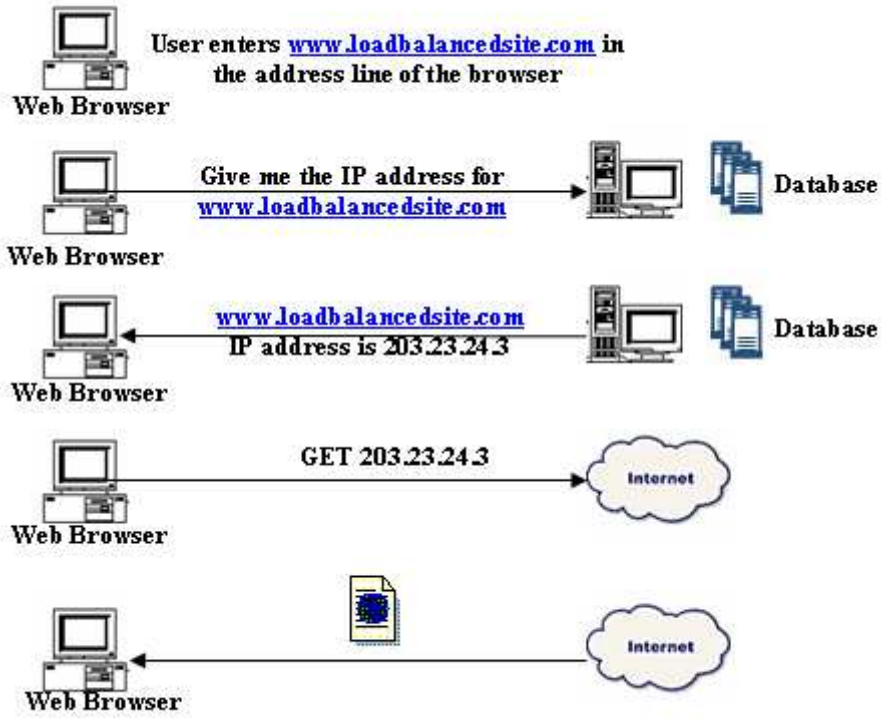

Figure 1

DNS server generally contains a single IP address mapped to a particular site name. In the example, the site www.loadbalancedsite.com maps to the IP address 203.23.24.3. To balance server loads using DNS, the DNS server maintains several different IP addresses for a site name. The multiple IP addresses represent the machines in the cluster, all of which map to the same single logical site name. Using the example, "www.loadbalancedsite.com" could be hosted on three machines in a cluster with the following IP addresses:

203.34.23.3

203.34.23.4

203.34.23.5.

In this case, the DNS server contains the following mappings:

www.loadbalancedsite.com 203.34.23.3

www.loadbalancedsite.com 203.34.23.4

www.loadbalancedsite.com 203.34.23.5.

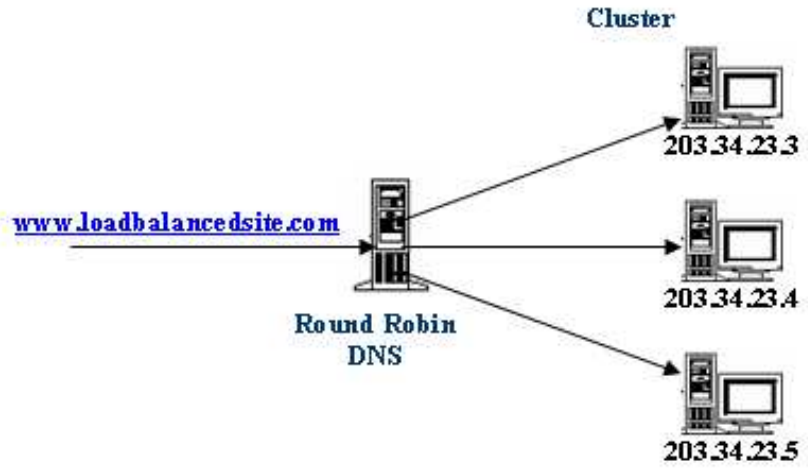

Figure 2

When the first request arrives at the DNS server, it returns the IP address 203.34.23.3, the first machine. On the second request, it returns the second IP address: 203.34.23.4. And so on. On the fourth request, the first IP address is returned again. Using the above DNS round robin, all of the requests to the particular site have been evenly distributed among all of the machines in the cluster. Therefore, with the DNS round robin method of load balancing, all of the nodes in the cluster are exposed to the net.

Advantages of DNS round robin The main advantages of DNS round robin are that it's cheap and easy to set up:

Inexpensive and easy to set up: The system administrator only needs to make a few changes in the DNS server to support round robin, and many of the newer DNS servers already include support. It doesn't require any code change to the Web application; in fact, Web applications aren't aware of the load-balancing scheme in front of it.

Simplicity: It does not require any networking experts to set up or debug the system in case a problem arises.

Disadvantages of DNS round robin Two main disadvantages of this software-based method of load balancing are that it offers no real support for server affinity and doesn't support high availability.

No server affinity (Session state) Server affinity is a load-balancing system's ability to manage a user's requests, either to a specific server or any server, depending on whether session information is maintained on the server or at an underlying, database level.

Server affinity is needed because some applications maintain state related to a client across TCP connections. Examples of such applications are web applications where a user fills out a multi-page

form or applications that use SSL. To accommodate this, a port can be defined as being "sticky" and a "sticky time-out" is configured. What this means is that the first time a client connects to an application associated with a "sticky" port, the client's request is load balanced as usual. However, when the TCP connection ends, the load balancer remembers the association between the client and the server. If a new TCP connection request is received from the same client before the "sticky" time-out has expired, the new connection is just assigned to the same server.

Without server affinity, DNS round robin relies on one of three methods devised to maintain session control or user identity to requests coming in over HTTP, which is a stateless protocol.

- cookies
- hidden fields
- URL rewriting

When a user makes a first request, the Web server returns a text-based token uniquely identifying that user. Subsequent requests include this token using either cookies, URL rewriting, or hidden fields, allowing the server to appear to maintain a session between client and server. When a user establishes a session with one server, all subsequent requests usually go to the same server. The problem is that the browser caches that server's IP address. Once the cache expires, the browser makes another request to the DNS server for the IP address associated with the domain name. If the DNS server returns a different IP address, that of another server in the cluster, the session information is lost.

No support for high availability Consider a cluster of "n" nodes. If a node goes down, then every "nth" request to the DNS server directs the clients to the dead node. An advanced router solves this problem by checking nodes at regular intervals, detecting failed nodes and removing them from the list, so no requests go to them. However the problem still exists if the node is up but the Web application running on the node goes down. Changes to the cluster take time to propagate through the rest of the Internet. One reason is that many large organizations –ISPs, corporations, agencies – cache their DNS requests to reduce network traffic and request time. When a user within these organizations makes a DNS request, it's checked against the cache's list of DNS names mapped to IP addresses. If it finds an entry, it returns the IP address to the user. If an entry is not found in its local cache, the ISP sends this DNS request to the DNS server and caches response. This caching is controlled by a time-to-live (TTL) value that is appended to each piece of information by our DNS server. Now there is a dilemma: When this TTL value is set too high, the DNS traffic is decreased on the Internet link, but the other DNS servers cache the information too long, which leads to bad HTTP traffic distribution over the Web cluster. On the other hand, when this TTL value is set too low, the DNS traffic and the request time for the visitor are increased dramatically because the other DNS servers expire this information faster, so they have to resolve it more often. But now there is a better balancing of HTTP traffic. The decision for the best TTL value is thus dependent on the level of balancing that someone wants. It also depends on the number of intermittent delays appreciated by administrator, that the visitor will accept before deciding that the Web-cluster approach has reduced

the quality of service. In practice, a TTL of one hour has been shown to be adequate. When a cached entry expires, the ISP updates its local database by contacting other DNS servers. When the list of servers changes, it can take a while for the cached entries on other organizations' networks to expire and look for the updated list of servers. During that period, a client can still attempt to hit the downed serve node, if that client's ISP's still has an entry pointing to it. In such a case, some users of that ISP couldn't access the site on their first attempt, even if the cluster has redundant servers up and running.

This is a bigger problem when removing a node than when adding one. When a node is dropped, a user may be trying to hit a non-existing server. When a node is added, that server may just be under-utilized until its IP address propagates to all the DNS servers. Although this method tries to balance the number of users on each server, it doesn't necessarily balance the server load. Some users could demand a higher load of activity during their session than users on another server, and this methodology cannot guard against that inequity.

Before server load balancers there were two main ways to scale services: use faster and bigger servers, or use multiple servers with round robin DNS.

Solutions based on faster, bigger servers improved responsiveness and increased capacity, but did not help with availability. Eventually, one would reach the limits of the server. Multiple servers with round robin DNS was implemented at the Domain Name Server (DNS) by providing the DNS with a list of server addresses that a name could be resolved to. The DNS would just rotate through the list choosing a different server every time a request to resolve the name was processed. This approach helped with capacity and server efficiency while being transparent to the clients and the servers. However, it was not a panacea. Clients and intermediate name servers often cached IP addresses associated with names and didn't request name resolution for some period of time. This bypassed the distribution that would be achieved by the DNS rotating through the list. Another problem with the DNS solution was that the DNS did not have any awareness of the servers' state and would select a server regardless of whether it was available or overloaded, causing users to experience failures or long delays. This would be aggravated by the caching done by clients which would repeatedly send requests to failed servers.

1.8.2 Hardware (Server) load balancers

Server load balancers distribute TCP and UDP application load across a group of servers which are running the same set of applications while providing a single server image to the network. A virtual IP (VIP) address is defined to the load balancer and is associated with the addresses of the individual servers running the applications.

Clients are only aware of a service name, like "www.loadbalancedsite.com", which resolves to the VIP defined to the load balancer. The load balancer handles requests sent to the VIP by passing the requests to one of the real servers associated with the VIP based on user-configured criteria and the server's availability, capability and load.

Since all requests for a service are processed by the load balancer, it can become a single point of failure. Load balancers need to support some type of hot-standby configuration. There is typically a primary load balancer and a backup one. If the primary fails, the backup takes over. The hot-standby

support combined with the server redundancy of a cluster makes this architecture highly reliable and available.

Load balancers keep track of TCP connections. A connection request from the client to the server is recognized when a TCP SYN packet is received.

The load balancer selects the best server to handle the request based on configured criteria and server load monitoring. All packets on this connection are forwarded to the same server until a TCP FIN packet is received indicating the end of the connection. New connections from the same client, or another client, will now be load balanced again. Sometimes applications require that the same client be assigned to the same server across multiple TCP connection requests such as when a user is filling out a multi-page form using HTTP or when SSL is being used. Load balancers usually have ways to configure this type of operation on an application basis. This is referred to as a sticky connection.

As for UDP, and since there are no connections, each UDP packet is individually balanced to a server. This limits the use of UDP load balancing to stateless applications.

Hardware load balancers solve many of the problems faced by the round robin software solution through virtual IP addresses. The load balancer shows a single (virtual) IP address to the outside world, which maps to the addresses of each machine in the cluster. So, in a way, the load balancer exposes the IP address of the entire cluster to the world.

When a request comes to the load balancer, it rewrites the request header to point to other machines in the cluster. If a machine is removed from the cluster, the request doesn't run the risk of hitting a dead server, since all of the machines in the cluster appear to have the same IP address. This address remains the same even if a node in the cluster is down. Moreover, cached DNS entries around the Internet aren't a problem. When a response is returned, the client sees it coming from the hardware load balancer machine. In other words, the client is dealing with a single machine, the hardware load balancer.

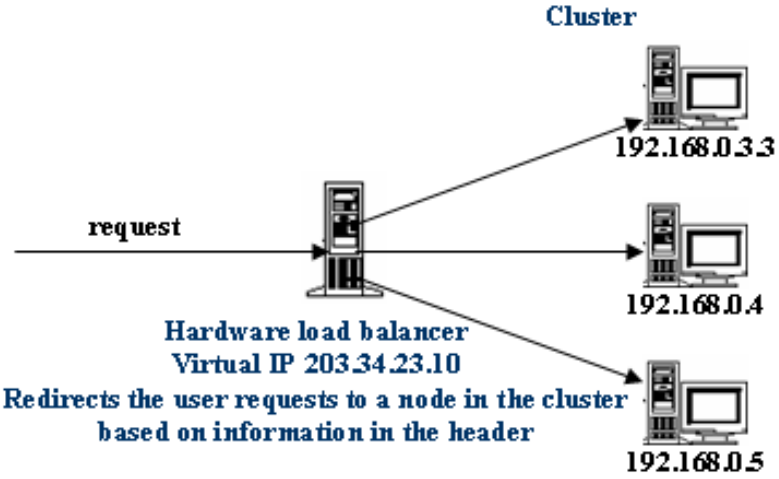

Figure 3

Advantages of hardware load balancers

High availability through fail-over: Fail-over happens when one node in a cluster can not process a request and redirects it to another. There are two types of fail-over:

- Request level fail-over
- Transparent session fail-over.

Request level fail-over: When one node in a cluster cannot process a request (often because it's down), it passes it along to another node.

Transparent session fail-over: When an invocation fails, it's transparently routed to another node in the cluster to complete the execution.

Hardware load balancers provide request-level fail-over; when the load balancer detects that a particular node has gone down, it redirects all subsequent requests for the dead node to another active node in the cluster. However, any session information on the dead node will be lost when requests are redirected to a new node.

Transparent session fail-over requires execution knowledge for a single process in a node, since the hardware load balancer can only detect network-level problems, not errors. In the execution process of a single node, hardware load balancers do not provide transparent session fail-over.

To achieve transparent session fail-over, the nodes in the cluster must collaborate among each other and have something like a shared memory area or a common database where all the session data is stored. Therefore, if a node in the cluster has a problem, a session can continue in another node.

It's difficult to load balance and maintain session information of requests that come in over HTTPS, as they're encrypted. The hardware load balancer cannot redirect requests based on the information in the header, cookies, or URL readings. There are two options to solve this problem:

- Web-servers proxies
- Hardware SSL Decoders

Web server proxies: A Web server proxy that sits in front of a cluster of Web servers takes all requests and decrypts them. Then it redirects them to the appropriate node, based on header information in the header, cookies, and URL readings.

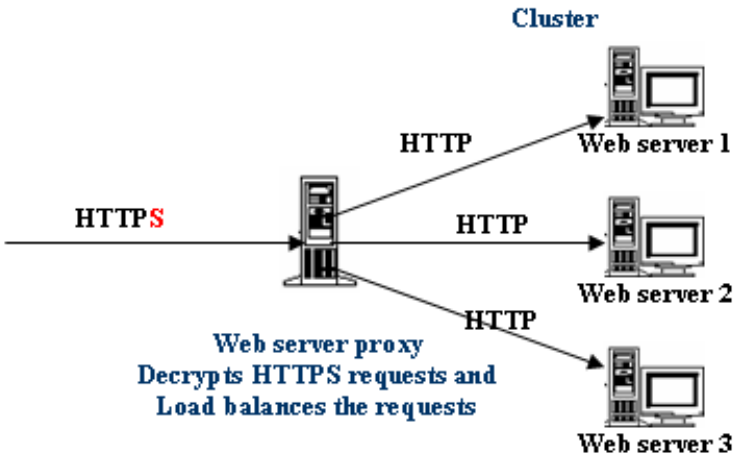

Figure 4

The advantages of Web server proxies are that they offer a way to get server affinity for SSL-encrypted messages, without any extra hardware but extensive SSL processing puts an extra load on the proxy.

Hardware SSL decoders Finally, there are hardware devices capable of decoding SSL requests. A complete description of them is beyond the scope of this project, but briefly, they sit in front of the hardware load balancer, allowing it to decrypt information in cookies, headers and URLs.

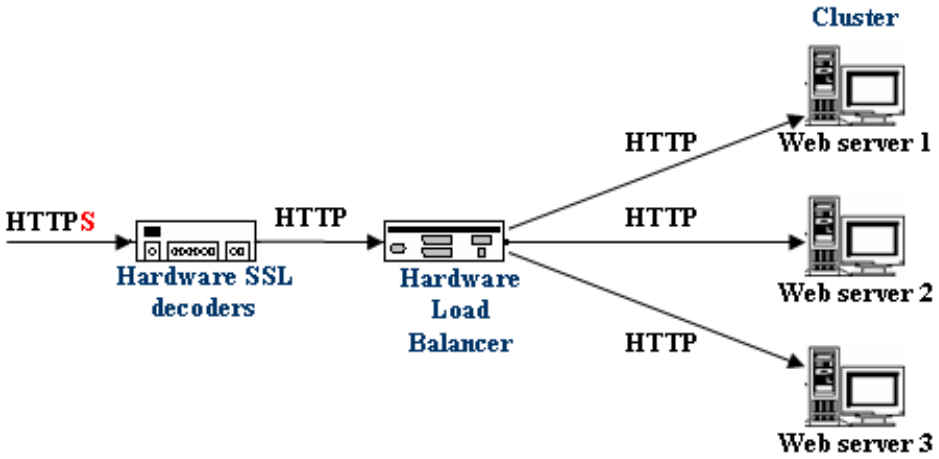

Figure 5

Since all requests to a Web application must pass through the load-balancing system, the system can determine the number of active sessions, the number of active sessions connected in any instance, response times, peak load times, the number of sessions during peak load, the number of sessions during minimum load, and more. All this audit information is used to fine tune the entire system for optimal performance.

Disadvantages of hardware load balancers The drawbacks to the hardware route are the costs, the complexity of setting up, and the vulnerability to a single point of failure. Since all requests pass through a single hardware load balancer, the failure of that piece of hardware sinks the entire site.

1.9 Summary

Once the web model's popularity increased, a great amount of HTTP traffic was experienced in the Internet. Web servers receive more and more requests and because people liked the web model this was implemented in more important applications. Companies rely some of their activities on web applications. Sometimes, because the web servers are struggled, some user requests are denied. Load balancing appeared as a need to increase the responsiveness of web based applications. This chapter presented the popular approaches and directions for improving the servers performance.

Chapter 2

The used tools

2.1 About PERL

Perl is a language for easily manipulating text, files and processes. It provides a more concise and readable way to do many jobs that were formally accomplished by programming in C language or in other shells. Intended to be a data reduction language, it became a convenient file manipulation language, that permits dealing with the files themselves, moving them, renaming them, changing their permissions, a convenient process manipulation language, allowing to create and destroy processes, to process their input and output, and a networking language, with the ability to communicate to other processes on other machines via sockets.

PERL scripting was chosen to implement this application because it offers all these facilities and it offers flexible methods to execute system commands, to take and parse a system command output and it offers libraries used for accomplishing tasks like making HTTP requests, detecting HTTP errors, packet construction and other things that are not easy to implement in other languages.

2.2 LWP - The World-Wide Web library for Perl

2.2.1 Description

The libwww-perl collection is a set of Perl modules which provides a simple and consistent application programming interface (API) to the World-Wide-Web. The main focus of the library is to provide classes and functions that allow the users to write WWW clients. The library also contains modules that are of more general use and even classes that help implementing simple HTTP servers.

Most modules in this library provide an object oriented API. The user agent, requests sent and responses received from the WWW server are all represented by objects. This makes a simple and powerful interface to these services.

The interface is easy to extend and customize for one's needs.

The main features of the library are:

- Contains various reusable components (modules) that can be used separately or together.

- Provides an object oriented model of HTTP-style communication. Within this framework http, https, gopher, ftp, news, mail are supported.
- Provides a full object oriented interface for a very simple procedural interface.
- Supports the basic and digest authorization schemes.
- Supports transparent redirect handling.
- Supports access through proxy servers.
- Implements HTTP content negotiation algorithm that can be used both in protocol modules and in server scripts (like CGI scripts).
- Supports HTTP cookies.

It is a simple command line client application called lwp-request.

2.2.2 How does it work?

The libwww-perl library is based on HTTP style communication. This section tries to describe what that means. The HTTP protocol is based on a request/response paradigm. A client establishes a connection with a server and sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content. The server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message containing server information, meta-information, and possible body content.

What this means for libwww-perl is that communication always take place through these steps:

- First a request object is created and configured.
- This object is then passed to a server
- Get a response object in return that can be examined.

A request is always independent of any previous requests, i.e. the service is stateless.

The same simple model is used for any kind of service that needs to be accessed. For example, if someone wants to fetch a document from a remote file server, he sends it a request that contains a name for that document and the response will contain the document itself. If a search engine is accessed, then the content of the request will contain the query parameters and the response will contain the query result. If a mail message must be sent to somebody, then a request object is sent. This object contains the message to the mail server and the response object will contain an acknowledgment that tells the sender that the message has been accepted and will be forwarded to the recipient(s).

2.2.3 The Request Object

The libwww-perl request object has the class name HTTP::Request. The fact that the class name uses HTTP:: as a prefix only implies that the HTTP model of communication is used. It does not limit the kind of services that can be used. For instance, HTTP::Requests will be sent both to ftp and gopher servers, as well as to the local file system.

The main attributes of the request objects are:

- The method is a short string that tells what kind of request this is. The most common methods are GET, PUT, POST and HEAD.
- The URI is a string denoting the protocol, server and the name of the "document" which will be accessed. The URI might also encode various other parameters.
- The headers contain additional information about the request and can also be used to describe the content. The headers are a set of keyword/value pairs.
- The content is an arbitrary amount of data

2.2.4 The Response Object

The libwww-perl response object has the class name HTTP::Response.

The main attributes of objects of this class are:

- The code is a numerical value that indicates the overall outcome of the request.
- The message is a short, human readable string that corresponds to the code.
- The headers contain additional information about the response and describe the content.
- The content is an arbitrary amount of data.

Since sometimes it is not desired to handle all possible code values directly in the programs, a libwww-perl response object has methods that can be used to query what kind of response this is. The most commonly used response classification methods are:

is_success() The request was successfully received, understood or accepted.

is_error() The request failed. The server or the resource might not be available, access to the resource might be denied or other things might have failed for some reason.

2.2.5 The User Agent

What happens when the user has created a request object? What is the next step in order to receive an response?

The answer is that the request object is passed to a user agent object and this object takes care of all the things that need to be done (like low-level communication and error handling) and returns a response object. The user agent represents the application on the network and provides the application with an interface that can accept requests and return responses.

The User Agent is an interface layer between the application code and the network. Through this interface the application is able to access the various servers on the network.

The class name for the user agent is LWP::UserAgent. Every libwww-perl application that wants to communicate should create at least one object of this class. The main method provided by this object is request(). This method takes an HTTP::Request object as argument and (eventually) returns a HTTP::Response object.

The User Agent has many other attributes that let the programmer configure how it will interact with the network and with the application.

- The timeout specifies how much time is given to the remote servers to respond before the library disconnects and creates an internal timeout response.
- The agent specifies the name that the application should use when it presents itself on the network.
- The from attribute can be set to the e-mail address of the person responsible for running the application. If this is set, then the address will be sent to the servers with every request.
- The parse_head specifies whether the response headers should be initialized from the <head> section of HTML documents.
- The proxy and no_proxy attributes specify if and when to go through a proxy server.
- The credentials provide a way to set up user names and passwords needed to access certain services.

2.2.6 Classes provided with the library

The library provides some classes. We use:

LWP::Simple – Simplified procedural interface for common functions.

Functions available in this class:

- get($url)
- head($url)

- getprint($url)
- getstore($url, $file)
- mirror($url, $file)

HTTP::Status constants and procedures.

This can used when the response code from getprint(), getstore() or mirror() is checked. Using these constants the programmer can check all the HTTP messages.

The HTTP::Status classification functions are:

is_success($rc) True if response code indicated a successful request.

is_error($rc) True if response code indicated that an error occurred.

The constants are:

RC_CONTINUE (100)
RC_SWITCHING_PROTOCOLS (101)
RC_PROCESSING (102)
RC_OK (200)
RC_CREATED (201)
RC_ACCEPTED (202)
RC_NON_AUTHORITATIVE_INFORMATION (203)
RC_NO_CONTENT (204)
RC_RESET_CONTENT (205)
RC_PARTIAL_CONTENT (206)
RC_MULTI_STATUS (207)
RC_MULTIPLE_CHOICES (300)
RC_MOVED_PERMANENTLY (301)
RC_FOUND (302)
RC_SEE_OTHER (303)
RC_NOT_MODIFIED (304)
RC_USE_PROXY (305)
RC_TEMPORARY_REDIRECT (307)
RC_BAD_REQUEST (400)
RC_UNAUTHORIZED (401)
RC_PAYMENT_REQUIRED (402)
RC_FORBIDDEN (403)
RC_NOT_FOUND (404)
RC_METHOD_NOT_ALLOWED (405)
RC_NOT_ACCEPTABLE (406)
RC_PROXY_AUTHENTICATION_REQUIRED (407)

RC_REQUEST_TIMEOUT (408)
RC_CONFLICT (409)
RC_GONE (410)
RC_LENGTH_REQUIRED (411)
RC_PRECONDITION_FAILED (412)
RC_REQUEST_ENTITY_TOO_LARGE (413)
RC_REQUEST_URI_TOO_LARGE (414)
RC_UNSUPPORTED_MEDIA_TYPE (415)
RC_REQUEST_RANGE_NOT_SATISFIABLE (416)
RC_EXPECTATION_FAILED (417)
RC_UNPROCESSABLE_ENTITY (422)
RC_LOCKED (423)
RC_FAILED_DEPENDENCY (424)
RC_INTERNAL_SERVER_ERROR (500)
RC_NOT_IMPLEMENTED (501)
RC_BAD_GATEWAY (502)
RC_SERVICE_UNAVAILABLE (503)
RC_GATEWAY_TIMEOUT (504)
RC_HTTP_VERSION_NOT_SUPPORTED (505)
RC_INSUFFICIENT_STORAGE (507)

2.3 Net::Pcap - packet capture library

2.3.1 Description

Net::Pcap is a Perl binding to the LBL pcap(3) library, version 0.7.2. The README for libpcap describes itself as: "a system-independent interface for user-level packet capture. libpcap provides a portable framework for low-level network monitoring. Applications include network statistics collection, security monitoring, network debugging, etc."

2.3.2 Functions:

All functions defined by Net::Pcap are direct mappings to the libpcap functions. The functions used in the script are described in the following lines.

1.Lookup functions

Net::Pcap::lookupdev(\$err);

Returns the name of a network device that can be used with Net::Pcap::open_live() function.

On error, the $err parameter is filled with an appropriate error message else it is undefined.

Net::Pcap::lookupnet($dev, \$net, \$mask, \$err);

Determine the network number and netmask for the device specified in $dev. The function returns 0 on success and sets the $net and $mask parameters with values. On failure it returns -1 and the $err parameter is filled with an appropriate error message.

2.Packet capture functions

Net::Pcap::open_live($dev, $snaplen, $promisc, $to_ms, \$err);

Returns a packet capture descriptor for looking at packets on the network. The $dev parameter specifies which network interface to capture packets from. The $snaplen and $promisc parameters specify the maximum number of bytes to capture from each packet, and whether to put the interface into promiscuous mode, respectively. The $to_ms parameter specifies a read timeout in ms. The packet descriptor will be undefined if an error occurs, and the $err parameter will be set with an appropriate error message.

Net::Pcap::loop($pcap_t, $cnt, \&callback_fn, $user_data);

Reads $cnt packets from the packet capture descriptor $pcap_t and call the perl function &callback_fn with an argument of $user_data. If $cnt is negative, then the function loops forever or until an error occurs.

The callback function is also passed packet header information and packet data like so:

```
sub process_pkt {
my($user_data, $hdr, $pkt) = @_;
...
}
```

Net::Pcap::close($pcap_t);

Closes the packet capture device associated with descriptor $pcap_t.

Net::Pcap::compile($pcap_t, \$filter_t, $filter_str, $optimize, $netmask);

Compiles the filter string contained in $filter_str and store it in $filter_t. A description of the filter language can be found in the libpcap source code, or the manual page for tcpdump . The filter is optimized if the $optimize variable is true. The netmask of the network device must be specified in the $netmask parameter. The function returns 0 if the compilation was successful, or -1 if there was a problem.

Net::Pcap::setfilter($pcap_t, $filter_t);

Associates the compiled filter stored in $filter_t with the packet capture descriptor $pcap_t.

2.3.3 Usage

This PERL library allows the user to define some packets he wants to peek from the network. It takes the defined packets as soon as they arrive. The packets are provided to the application and the information from them can be further more used by other libraries that have the ability to decode the packets and to understand the information included with them.

For using the abilities provided by the library, the following steps are performed:

1. create a packet capture descriptor
2. compile filters in order to define the type of packets to be taken from the network
3. associate the filters with the packet capture descriptor
4. use the loop function in order to start the packet capturing

When a defined packet is taken from the network it is processed inside the callback function defined with the loop. Next, the libraries that have the ability to process this packets will be presented.

2.4 NetPacket module

These are the modules used to assemble/disassemble network packets at the protocol level.

NetPacket provides a base class for a cluster of modules related to decoding and encoding of network protocols. Each NetPacket descendant module knows how to encode and decode packets for the network protocol it implements.

2.4.1 NetPacket::Ethernet

Methods:

$eth_obj=NetPacket::Ethernet->decode($raw_pkt);

Decode the raw packet data given and return an object containing instance data. The instance data can be:

- src_mac - The source MAC address of the ethernet packet
- dest_mac - The destination MAC address of the ethernet packet
- type - The protocol type for the ethernet packet
- data - The payload for the ethernet packet.

The instance data can be accessed using $eth_obj->{src_mac} , $eth_obj->{dest_mac} and so on.

$eth_data=NetPacket::Ethernet::strip($raw_pkt);

Returns the encapsulated data or payload contained in the ethernet packet. This data is suitable to be used as input for other NetPacket::* modules.

The $raw_pkt can be one of the packets peeked from the network using Net::Pcap library.

2.4.2 NetPacket::IP

Methods

$ip_obj=NetPacket::IP->decode($raw_pkt);

Decode the raw packet data given and return and object containing instance data. The instance data is represented by all the fields from an IP header :

ver - The IP version number of this packet
hlen - The Ip header length of this packet
flags - The IP header flags for the packet
foffset - The Ip fragment offset for this packet
tos - Type of service for this packet
len - The length in bytes for this IP packet
id - The identification number of this IP packet
ttl - Time-to-live value for this packet
proto - The IP protocol number for this packet
cksum - The IP checksum value for this packet
src_ip - The source IP address for this packet in dotted- quad notation.
dest_ip - The destination IP address for this packet in dotted-quad notation.
options - Any IP option of this packet.
data - The encapsulated data for this IP packet.
The fields of the packet can be accessed using $ip_obj->{field_name} ;

$ip_pkt=NetPacket::IP->encode($ip_obj);

Returns an IP packet with the fields from the header obtained from the instance data contained in the $ip_obj.

$ip_dat=NetPacket::IP->strip($raw_pkt);

Returns the encapsulated data contained in the IP packet. This data is suitable to use as input for other NetPacket::* modules. The $raw_pkt can be obtained with a decode method from NetPacket::Ethernet.

2.4.3 NetPacket::TCP

It is a module used to assemble and disassemble TCP (Transmission Control Protocol) packets.

Methods

$tcp_obj=NetPacket::TCP->decode($raw_pkt);

Decode a raw packet data and returns an object containing as instance data the values of the fields from the TCP packet .

Instance data:
src_port - The source TCP port for the packet
dest_port - The destination TCP port for the packet
seqnum - The TCP sequence number for this packet
acknum - The TCP acknowledgment number for this packet.
hlen - The header length for this packet
reserved - The 6 - bit “reserverd ” space in the TCP header.
flags - Contains the Flags for TCP packet (syn, ack, urg, psh, rst, fin, ece, cwr);
winsize - TCP window size for this packet.
cksum - The TCP checksum.
urg - The TCP urgent pointer
options - Any TCP options
data - The encapsulated data (payload) for this packet.
Obtaining a field from the TCP header is done with $tcp_obj->{field_name}.

$tcp_pkt=NetPacket::TCP->encode($tcp_obj);

Returns a TCP packet encoded with the information from the instance data of the object $tcp_obj.

$tcp_data=NetPacket::TCP->strip($raw_pkt);

Returns the encapsulated data from the TCP packet.

2.4.4 Usage

Using this libraries a packet can be peeked from the network, and and the programmer can obtain the values from the headers step by step until he gets to the TCP fields that are needed in the application.

Steps:

1. Take a packet picked from the network using Net::Pcap library
2. Use NetPacket::Ethernet->decode() and obtain all the Ethernet fields from the header and the data.
3. The data from the ethernet packet is decoded with NetPacket ::IP->decode(), and the fields from the IP header and the data are obtained.
4. The data from the IP packet is decoded with Net::Packet::TCP->decode(), and the TCP fields from the packet are obtained.

2.5 Net::RawIP library

2.5.1 Description:

This library can be used for creating, manipulating and sending raw IP packets with optional feature for manipulating ethernet headers.

The library is user in this application to build and sent this type of packets.

2.5.2 Functions

$a = new Net::RawIP

Creates a new packet object

$a->set({ip =>{IPKEY=>IPVALUE, ...}, ARGPROTO=>{PROTOKEY=>PROTOVALUE, ...}

Creates a packet encoded with the fields specified for the IP protocol and the another IP based protocol.

IPKEY: version, ihl, tos, tot_len, id, flag_off, ttl, protocol, ckeck, saddr, daddr.
ARGPROTO: tcp, udp, icmp
PROTOKEY:

- for tcp: source, dest seq, ack_seq, doff res1 res2 urg, ack, psh, syn, fin, window, check urg_ptr data
- for udp: source, dest, len, check, data
- for icmp: type, code, ckeck, gateway, id, sequence, unused, mtu, data

$a->send

Sends the packet that was created.

2.5.3 Usage

1. Create a new packet object
2. Setting the fields for every protocol using “set”
3. Send the packet on the network using “send”

2.6 About RRDTool

2.6.1 Overview:

RRDtool originated from MRTG (Multi Router Traffic Grapher).

RRD is the Acronym for Round Robin Database. RRD is a system to store and display time-series data. It stores the data in a very compact way that will not expand over time, and it presents useful graphs by processing the data to enforce a certain data density.

RRDtool can be used to store and process data collected from different sources.

RRDtool lets the users create a database, store data in it, retrieve that data and create graphs in GIF format for display on a web browser. Those GIF images are dependent on the collected data and could be, for instance, an overview of the average network usage, or the peaks that occurred.

It can also be used to display tidal waves, solar radiation, power consumption, number of visitors at an exhibition, noise levels near an airport, temperature on the favorite holiday location, temperature in the fridge and whatever the imagination can come up with. The only thing that is needed is a sensor to measure the data and be able to feed the numbers to RRDtool

2.6.2 HOW DOES RRDTOOL WORK?

Data acquisition

When monitoring the state of a system, it is convenient to have the data available at a constant interval. Unfortunately one may not always be able to fetch data at exactly the time he want to. Therefore RRDTool permits updating the logfile at any time. It will automatically interpolate the value of the data-source (DS) at the latest official time-slot and write this value to the log. The supplied value is stored as well and is also taken into account when interpolating the next log entry.

Consolidation

The data can be loged at a 1 minute interval, but one may also be interested to know the development of the data over the last year. This can be done by simply storing the data in 1 minute interval, for one year. While this would take considerable disk space it would also take a lot of time to analyze the data when creating a graph covering the whole year.

RRDTool offers a solution to this problem through its data consolidation feature.

When setting up an Round Robin Database (RRD), one can define at which interval this consolidation should occur, and what consolidation function (CF) (average, minimum, maximum, last) should be used to build the consolidated values. Any number of different consolidation setups can be defined within one RRD. They will all be maintained on the fly when new data is loaded into the RRD.

Round Robin Archives

Data values of the same consolidation setup are stored into Round Robin Archives (RRA). This is a very efficient manner to store data for a certain amount of time, while using a known amount of storage space.

It works like this: If someone wants to store 1000 values in 5 minute interval, RRDTool will allocate space for 1000 data values and a header area. In the header it will store a pointer telling which one of the values in the storage area was last written. New values are written to the Round

Robin Archive in a round robin manner. This automatically limits the history to the last 1000 values. Because several RRAs can be defined within a single RRD, another RRA can be set up, storing 750 data values at a 2 hour interval and thus keeping a log for the last two months although at a lower resolution.

The use of RRAs guarantees that the RRD does not grow over time and that old data is automatically eliminated. By using the consolidation feature, the data can be kept for a very long time, while gradually reducing the resolution of the data along the time axis. Using different consolidation functions (CF) allows storing exactly the type of information that actually presents an interest . (Maximum one minute traffic on the LAN, minimum temperature of the wine cellar, total minutes down time ...)

Unknown Data

As mentioned earlier, the RRD stores data at a constant interval. Now it may happen that no new data is available when a value has to be written to the RRD. Data acquisition may not be possible for one reason or other. The RRDTool handles these situations by storing an *UNKNOWN* value into the database. The value '*UNKNOWN*' is supported through all the functions of the database. When a new consolidated value is ready to be written to its Round Robin Archive (RRA) a validity check is performed to make sure that the percentage of unknown data in the new value is below a configurable level. If so, an *UNKNOWN* value will be written to the RRA.

Graphing

The RRDTool also allows one to generate reports in numerical and graphical form based on the data stored in one or several RRDs. The graphing feature is fully configurable. Size, color and contents of the graph can be defined freely.

2.6.3 RRDTool Functions

create Sets up a new Round Robin Database (RRD).

update Stores new data values into an RRD.

graph Creates a graph from data stored in one or several RRD. Apart from generating graphs, data can also be extracted to stdout.

dump Dumps the contents of an RRD in plain ASCII. In connection with restore, one can use it to transport an RRD from one architecture to another.

restore Restores an RRD in XML format to a binary RRD

fetch Gets data for a certain time period from a RRD. The graph function uses fetch to retrieve its data from an RRD.

tune Alter setup of an RRD

last Finds last update time of an RRD.

info Displays information about an RRD.

rrdresize Changes the size of individual RRAs

xport Exports data retrieved from one or several RRD.

rrdcgi This is a standalone tool for producing RRD graphs.

2.6.4 Creating the database

RRDTool create sets up a new Round Robin Database

SYNOPSIS

rrdtool create filename [-start|-b start time] [-step|-s step][DS:ds-name:DST:heartbeat:min:max][RRA:

DESCRIPTION

The create function of the RRDtool lets the users set up new Round Robin Database (RRD) files. The file is created at its final, full size and filled with *UNKNOWN* data.

filename

The name of the RRD database to be created. RRD files should end with the extension .rrd. However, rrdtool will accept any filename.

-start|-b start time (default: now -10s)

Specifies the time in seconds since 1970.01.01 UTC when the first value should be added to the RRD. RRDTool will not accept any data timed before or at the time specified.

–step|-s step (default: 300 seconds)

Specifies the base interval in seconds with which data will be fed into the RRD.

DS:ds-name:DST:heartbeat:min:max

A single RRD can accept input from several data sources (DS). (e.g. Incoming and Outgoing traffic on a specific communication line).

With the DS configuration option some basic properties must be defined for each data source used to feed the RRD.

- ds-name - Is the name used to reference this particular data source from an RRD. A dsname must be 1 to 19 characters long in the characters [a-zA-Z0.9_].
- DST - Defines the Data Source Type.

The Data Source Type must be one of the following:

GAUGE

COUNTER

DERIVE-will store the derivative of the line going from the last to the current value of the data source.

ABSOLUTE

- heartbeat - Defines the maximum number of seconds that may pass between two updates of this data source before the value of the data source is assumed to be *UNKNOWN*.
- min and max - Are optional entries defining the expected range of the data supplied by this data source. If min and/or max are defined, any value outside the defined range will be regarded as*UNKNOWN*. If the users do not know or care about min and max, they should set them to U for unknown. Min and max always refer to the processed values of the DS.

RRA:CF:xff:steps:rows

The purpose of an RRD is to store data in the round robin archives (RRA). An archive consists of a number of data values from all the defined data-sources (DS) and is defined with an RRA line.

When data is entered into an RRD, it is first fit into time slots of the length defined with the -s option becoming a primary data point.

The data is also consolidated with the consolidation function (CF) of the archive. The following consolidation functions are defined: AVERAGE, MIN, MAX, LAST.

- steps - Defines how many of primary data points are used to build a consolidated data point which then goes into the archive.
- rows - Defines how many generations of data values are kept in an RRA.

The HEARTBEAT and the STEP

RRD gets fed samples at arbitrary times. From these it builds Primary Data Points (PDPs) at exact times every "step" interval. The PDPs are then accumulated into RRAs.

The "heartbeat" defines the maximum acceptable interval between samples. If the interval between samples is less than "heartbeat", then an average rate is calculated and applied for that interval. If the interval between samples is longer than "heartbeat", then that entire interval is considered "unknown".

The "heartbeat" can be short (unusual) or long (typical) relative to the"step" interval between PDPs. A short "heartbeat" means one requires multiple samples per PDP, and if the samples are not get they mark the PDP unknown. A long heartbeat can span multiple "steps", which means it is acceptable to have multiple PDPs calculated from a single sample. An extreme example of this might be a "step" of 5 minutes and a "heartbeat" of one day, in which case a single sample every day will result in all the PDPs for that entire day period being set to the same average rate.

2.6.5 Updating the database

RRDTool update stores a new set of values into the RRD

SYNOPSIS

rrdtool update filename [–template|-t ds-name[:ds-name]...] N|timestamp:value[:value...] [timestamp:value[:value...] ...]

DESCRIPTION

The update function feeds new data values into an RRD. The data gets time aligned according to the properties of the RRD to which the data is written.

filename

The name of the RRD to be updated.

–template|-t ds-name[:ds-name]...

By default, the update function expects the data input in the order the data sources are defined in the RRD. This is not very error resistant, as someone might be sending the wrong data into a RRD.

The template switch allows the users to specify which data sources we are going to update and in which order. If the data sources specified in the template are not available in the RRD file, the update process will abort with an error message.

N|timestamp:value[:value...]

The data used for updating the RRD was acquired at a certain time. This time can either be defined in seconds since 1970-01-01, or by using the letter ’N’ the update time is set to be the current time.

The remaining elements of the argument are DS updates. The order of this list is the same as the order the data sources were defined in the RRA. If there is no data for a certain data-source, the letter U (e.g. N:0.1:U:1) can be defined. The format of the value acquired from the data source is dependent of the data source type chosen.

2.6.6 Plotting the gathered values

RRDTool graph creates a graph based on data from one or several RRD

SYNOPSIS

rrdtool graph filename [-s|–start seconds] [-e|–end seconds] [-x|–x-grid x-axis grid and label] [-y|–y-grid y-axis grid and label] [-v|–vertical-label text][-S|–step value] [-c|–color COLORTAG#rrggbb] [-t|–title title] [DEF:vname=rrd:ds-name:CF] LINE{1|2|3}:vname[#rrggbb[:legend]]]

DESCRIPTION

The graph function has the purpose to create graphical representations of the data stored in one or several RRDs. Apart from generating graphs, it can also extract numerical reports.

filename

The name of the graph to generate. Since RRDTool outputs GIFs and PNGs, it's recommended that the filename end in either .gif or .png. RRDTool does not enforce this, however. If the filename is set to '-' the image file will be written to standard out. All other output will get suppressed. PNG output is recommended, since it takes up to 40% less disk space and 20-30% less time to generate than a GIF file. If no graph functions are called, the graph will not be created.

-s|–start seconds (default end-1day)

The time when the graph should begin. Time in seconds since epoch (1970-01-01) is required. Negative numbers are relative to the current time. By default one day worth of data will be graphed.

-e|–end seconds (default now)

The time when the graph should end.

-x|–x-grid x-axis grid and label (default auto-configure)

The x-axis label and grid can be configured, using the following format:

GTM:GST:MTM:MST:LTM:LST:LPR:LFM

Three elements making up the x-axis labels and grid are configured. The base grid (G??), the major grid (M??) and the labels (L??).

The configuration is based on the idea that first is specified a well known amount of time (?TM) and then say how many times it has to pass between each grid line or label (?ST). For the label two additional items must be defined: the precision of the label in seconds (LPR) and the format used to generate the text of the label (LFM). The ?TM elements must be one of the following keywords: SECOND, MINUTE, HOUR, DAY, WEEK, MONTH or YEAR.

For a graph with a base grid every 10 minutes and a major one every hour, with labels every hour the following x-axis definition should be used:

MINUTE:10:HOUR:1:HOUR:1:0:%X

The precision in this example is 0 because the %X format is exact.

If the label was the name of the DAY, the precision would have been 24 hours, because when one says something like 'Monday' it means the whole day and not Monday morning 00:00. Thus the label should be positioned at noon.

v|–vertical-label text

Vertical label on the left side of the graph. This is normally used to specify the units used.

-c|–color COLORTAG#rrggbb (default colors)

Override the colors for the standard elements of the graph. The COLORTAG must be one of the following symbolic names: BACK ground, CANVAS, SHADEA left/top border, SHADEB right/bottom border, GRID, MGRID major grid, FONT, FRAME and axis of the graph or ARROW. This option can be called multiple times to set several colors.

DEF:vname=rrd:ds-name:CF

Define virtual name for a data source. This name can then be used in the functions explained below. The DEF call automatically chooses an RRA which contains CF consolidated data in a resolution appropriate for the size of the graph to be drawn. Ideally this means that one data point from the RRA should be represented by one pixel in the graph. If the resolution of the RRA is higher than the resolution of the graph, the data in the RRA will be further consolidated according to the consolidation function (CF) chosen

LINE{1|2|3}:vname[#rrggbb[:legend]]

Plot for the requested data, using the color specified. Write a legend into the graph. The 3 possible keywords LINE1, LINE2, and LINE3 generate increasingly wide lines.

2.7 IPTABLES

2.7.1 Networking (Very) Basics

All traffic through a network is sent in the form of packets. For example, downloading a package (say it's 50k long) might cause someone to receive 36 or so packets of 1460 bytes each, (to pull numbers at random).

The start of each packet says where it's going, where it came from, the type of the packet, and other administrative details. This start of the packet is called the header. The rest of the packet, containing the actual data being transmitted, is usually called the body.

Some protocols, such TCP, which is used for web traffic, mail, and remote logins, use the concept of a connection' – before any packets with actual data are sent, various setup packets (with special headers) are exchanged saying 'I want to connect', OK' and Thanks'. Then normal packets are exchanged.

2.7.2 Packet Filter

A packet filter is a piece of software which looks at the header of packets as they pass through, and decides the fate of the entire packet. It might decide to deny the packet (ie. discard the packet as if it had never received it), accept the packet (ie. let the packet go through), or reject the packet (like deny, but tell the source of the packet that it has done so).

Under Linux, packet filtering is built into the kernel (as a kernel module at the moment), and there are a few trickier things can be done with packets, but the general principle of looking at the headers and deciding the fate of the packet is still there.

Why Would Anybody Want to Filter Packets?

Control. Security. Watchfulness.

Control: When using a Linux box to connect an internal network to another network (say, the Internet) there is the opportunity to allow certain types of traffic, and disallow others. For example, the header of a packet contains the destination address of the packet, so one can prevent packets going to a certain part of the outside network.

Security: When the Linux box is the only thing between the chaos of the Internet and an internal network, it's nice to know that what comes from the Internet can be controlled . For example, it is desired to allow anything to go out from a network, but one might be worried about the well-known Ping of Death' coming in from malicious outsiders. As another example, it is usually restricted to outsiders TELNETTing to an internal Linux box, even though all the accounts have passwords; maybe someone wants (like most people) to be observer on the Internet, and not a server (willing or otherwise) – simply no one is allowed to connect in, by having the packet filter reject incoming packets used to setup connections.

Watchfulness: Sometimes a badly configured machine on the local network will decide to spew packets to the outside world. It's important to tell the packet filter to announce if anything abnormal occurs; maybe there is something that can be done about it, or maybe the administrator just wants to know about it.

Iptables splits the packet handling into three different tables, each of which contain a number of chains. The firewall rules that are created are included within a particular chain. The three tables are:

filter - used for packet filtering

nat - used to provide packet modification capabilities

mangle - used for setting packet options

2.7.3 Filter table

The kernel starts with three lists of rules; these lists are called firewall chains or just chains. The three chains are called INPUT, OUTPUT and FORWARD.

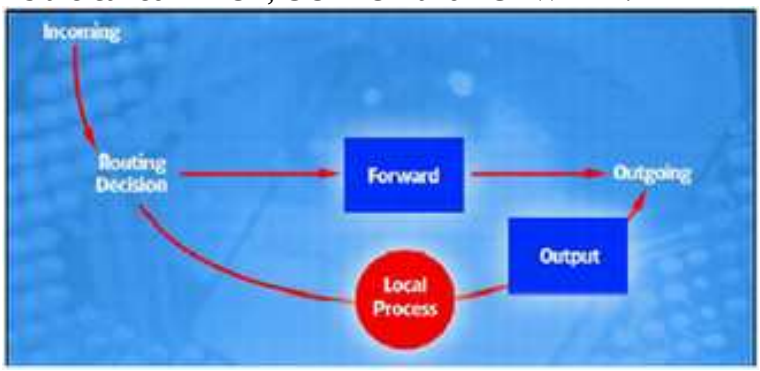

Figure 6

The figure represent the chains mentioned above. When a packet reaches a chain in the diagram, that chain is examined to decide the fate of the packet. If the chain says to DROP the packet, it is killed there, but if he chain says to ACCEPT the packet, it continues traversing the diagram.

A chain is a checklist of rules. Each rule says if the packet header looks like this, then here's what to do with the packet'. If the rule doesn't match the packet, then the next rule in the chain is consulted. Finally, if there are no more rules to consult, then the kernel looks at the chain policy to decide what to do. In a security-conscious system, this policy usually tells the kernel to DROP the packet.

When a packet comes in (say, through the Ethernet card) the kernel first looks at the destination of the packet: this is called routing.

If it's destined for this box, the packet passes down-wards in the diagram, to the INPUT chain. If it passes this, any processes waiting for that packet will receive it.

Otherwise, if the kernel does not have forwarding enabled, or it doesn't know how to forward the packet, the packet is dropped. If forwarding is enabled, and the packet is destined for another network

interface (if there is another one), then the packet goes rightwards on the diagram to the FORWARD chain. If it is ACCEPTed, it will be sent out.

Finally, a program running on the box can send network packets. These packets pass through the OUTPUT chain immediately: if it says ACCEPT, then the packet continues out to whatever interface it is destined for.

2.7.4 Network Address Translation table

NAT allows modifying the source or destination address or port of a packet, allows it to be redirected so that it appears to come from another system.

The NAT table has three different chains, each of which is checked at a particular position in the routing of a packet.

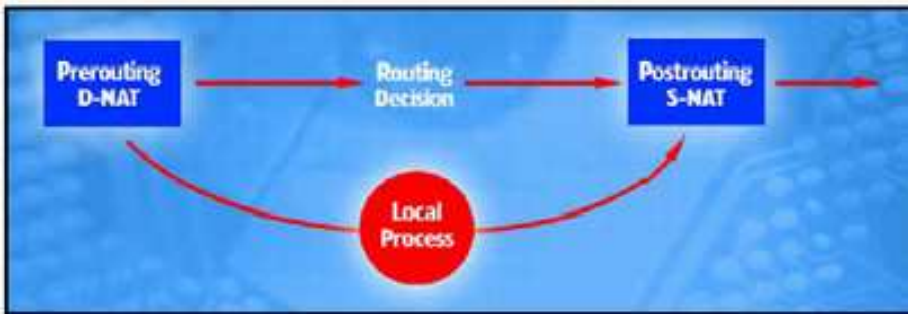

Figure 7

The first is PREROUTING , who's rules are checked for match before the system attempts to route the packet anywhere. PREROUTING is where destination NAT or DNAT is performed. DNAT can also be performed in the output chain, which is for packets that are originated locally, as they are never checked by PREROUTING. Finally the POSTROUTING is the last chain to be checked for a rule match, which is where the source NAT, or SNAT is performed.

Destination NAT

On a large network with many systems running public-facing services, such as web and email servers, giving each front-end system its IP address is often rather expensive. Each system may be running one or two services, but requires one IP address to perform this function.

Instead, another option is to use Destination NAT, or DNAT capabilities of iptables in order to redirect packets from the front end IPs onto the servers using internal IP addressing. This permits running many services on a single IP address, even if the servers are provided by more than one physical box.

Source NAT

SNAT became very useful due to the crises of IP version 4 addresses. Instead of using public addresses, one can use as many private addresses as needed. The problem is that the first major Internet Service Provider will filter this packets. SNAT enables an administrator to use private addresses by

replacing the source address of packet with a public address as they leave the network. The advantage is that many computers with private addresses can be used and all these source addresses will be replaced with only one that is public. The Internet Service Provider will see all the packets coming from the computer with the public address that was used.

2.7.5 Mangling

As well as performing filtering or NAT packets can also be “mangled”. Mangling simply allows to modify packet options, such as TOS field or to mark the packet with a value. The “mangle” table in iptables has five different chains, so one can mangle packets at almost every stage of their processing and routing. However, if someone wants to make use of mangled packets in something other than iptables, he needs to compile in other options which will allow , for example, to perform QoS matching based on the packet mark.

2.7.6 Options

When iptables is used some options are usually specified specify:

- iptables
- command — specify if to insert, add, list a table or chain
- -t table —the table that will be used
- chain — the chain that will be used
- matching rules — the packet should match all the criteria specified in the rule
- -j target — if the packet matches a rule in the chain, the target is applied

Commands used with iptables:

-I used to insert a rule in a specific location in a chain. If the slot in which to insert the rule is not specified, it will be inserted in the first slot, so it will be the first to be checked.

-A used to add a rule in a chain. When adding a rule, it is added in the end of a chain, so it will be the last one checked. It should be specified that whichever rule matches first, will be the target for the packet and no other rules will be checked.

-D deletes a rule from the chain.

Matching packets

Matching packets is one of the most important aspects of the packet filtering because is necessary to have the ability to clearly define which packets to block and which packets to allow, or to clearly define which packets to modify in order to redirect then.

Here are presented some basic match conditions that will be used:

-p, –protocol —- this match is used to check for certain protocols (TCP, UDP, ICMP)

-s, –src, –source —- this is the source match, used to match packets based on their source IP address.

-d, –dst, –destination —- the destination match, used to match packets based on their destination address.

–sport, –source-port — this match is used with TCP or UDP protocols and matches packets based on their source port

–dport, –destination-port — this match is used with TCP or UDP protocols and matches packets based on their destination port

Targets

The target tells the rule what to do with a packet that is a perfect match with the match selection of the rule.

- Accept target

As soon as the match specification for a packet has been fully satisfied, and ACCEPT was specified as a target, the rule is accepted and will not continue traversing the current chain or any other ones in the same table. A packet that was accepted in one chain might still travel through chains within other tables, and could still be dropped there.

- DNAT target

It is used to do Destination Network Address Translation, which means it is used to rewrite the destination IP address of the packet. If a packet is matched, and this is the target of the rule, the packet and all subsequent packets in the same stream will be translated, and than routed on to the correct device, host or network.

This target is very useful for the application. Using this target, the packets are redirected from the load balancer to one of the balanced web servers.

- SNAT target

It is used to do Source Network Address Translation, which means it is used to rewrite the source IP address of the packet. If a packet is matched, and this is the target of the rule, the packet and all subsequent packets in the same stream will be translated after routing.

2.8 Summary

This chapter describes the libraries and the tools used in with the application. The purpose is to understand how they are used, what are their features, and to understand what is the role of each component.

Chapter 3

APD Load Balancer

3.1 Design goals

APD Load Balancer is a program for Unix operating systems that provides load balancing capabilities between web servers. The main focus of the application is to offer an inexpensive solution for sharing the WWW traffic between servers in a local network .

The application solves the problem that was described in the first chapter. When having an web server that is experiencing an increased amount of WWW traffic, the clients are affected by a high HTTP response time. The proposed solution to improve the application's responsiveness is to increase the number of servers that offer this service. A cluster of WWW servers is created and the WWW traffic will be balanced between these servers.

All the servers will be seen as a single server in the Internet and the whole load balancing process will be transparent for the clients. The clients will make the request for "www.loadbalancedsite.com" and from their perspective the load balancing system does not exist. They are aware of the existence of a single server.

APD Load Balancer will run on the load balancer machine, will gather the criteria used for load balancing and will apply the load balancing policy.

3.2 Topology details

The traffic balancing is implemented by a load balancer. This load balancer is a computer that has two network interface cards. One network interface card is for the connection with an external network or the Internet where the clients that are making the request usually are, and one network interface card for the connection with the LAN and with the balanced servers. A request may come also from the local network not only from the Internet, or other external network.

When a computer wants to communicate with another computer it actually wants to send data to a destination IP address. The destination computer may be in the same IP network with the source computer or in different networks. If the destination IP is in the same network, the data will be sent directly to the destination computer. If the destination computer is in a different IP network, the request will first go to a default gateway which is usually a router, and the default gateway will route

the data to the destination. In conclusion, all traffic destined outside the IP network will pass through the default gateway.

The servers and the computer from the local network will have the default gateway the IP address of the network interface card of load balancer connected to the local network (IP 2) and all traffic destined outside the network must pass through the load balancer.

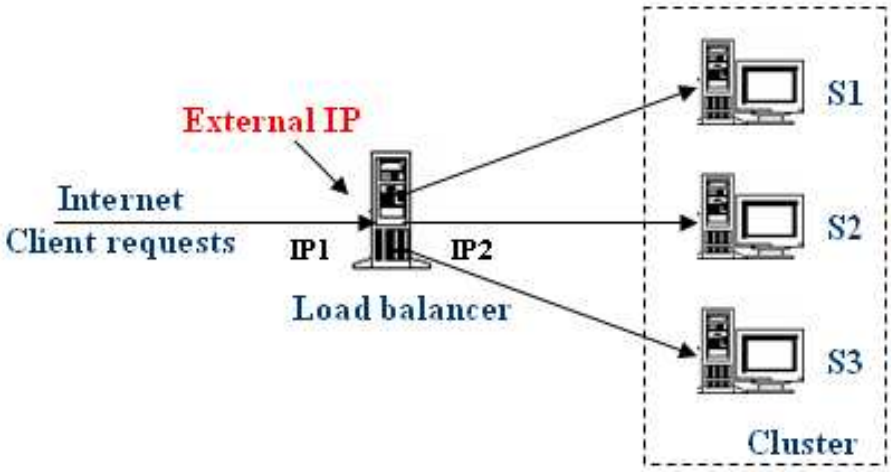

Figure 8

When users want to access the service from a web browser, they usually know their service by its name. A client makes a HTTP request for “www.loadblancedsite.com” and it will first make a DNS request to a DNS server asking what is the IP address associated with the service name. The DNS server will return the IP address of the network interface card of the load balancer that connects to the external network (IP1).

The HTTP requests that will come from clients will have the destination IP address IP1.

When a client from the internal network wants to access the web page “www.loadbalancedsite.com” it will first make a DNS request for this URL and will receive the IP address IP 1. If a client from the local network wants to communicate with one of the balanced servers using a different protocol (not HTTP), the traffic will go directly from the client to the server. For the HTTP protocol, the traffic will go from the client, to the load balancer and in the end to the balanced server. So every client will make the HTTP request to the load balancer. The load balancer will apply the load balancing policy and will choose the most convenient server based on the criteria it already computed. Because the clients from the local network will also access the servers through the load balancer, the web traffic generated in the local network can be included in the balanced traffic.

The maximum number of balanced servers is 6.

3.3 Input and Output

The application has a user interface for configuration and for monitoring.

Using the interface for configuration the administrator will introduce the informations required by the application:

- IP address of www.loadbalancedsite.com

- Number of balanced servers
- IP addresses of balanced servers
- Location of monitoring folder
- Location of iptables
- Location of RRDTool
- The optimum HTTP response time of the servers
- The device connected to the balanced servers

In the next section will be presented more informations about how the application uses this data.

The output data is presented in some web pages that can be found in the folder “www”. In the output, the user will be able to check the criteria used by the algorithm.

The folder “www” contains the following pages:

- index.html
- monitorh.html
- monitord.html
- monitorw.html

In this pages, the HTTP response time and the network delay can be monitored as an average in different periods of time.

The application will make a copy of this folder in the location specified by the administrator during the configuration process in the field "Monitoring folder".

The graphs are realized using RRDTool.

3.4 Features of the application:

3.4.1 Server monitoring:

The load balancer implements a mechanism for server monitoring and application monitoring.

- Server monitoring is realized by sending TCP SYN packets and waiting TCP ACK response. If the machine is not functioning no response will be received. This server will be excluded from the cluster.
- Application monitoring is realized by sending HTTP requests and waiting for the response. In case a response is received, it is also checked if it represents a HTTP error code or the requested page was included into the response. The servers that do not respond or the servers that will send an error message will be excluded.

3.4.2 Availability:

The application supports multiple servers on multiple machines and if one server fails another server will take over. The load balancer has the ability to detect server failures using server monitoring. If one server fails it is excluded from the list of the available servers and the load balancing process will continue using the other servers from the list. If one server was down but it started to work it will automatically be included in the list of available servers.

3.4.3 Scalability:

The application has the ability to support a great number of users because it can use up to 6 servers for load balancing. It provides a much better response time and it can serve more users in high traffic load conditions than the single server approach.

3.4.4 Dynamic changes in configuration

In case some parameters on one server need to be changed, this can be done without affecting the system. If changes need to be performed the specific server can be disconnected and after the changes are performed the server can be plugged back into the cluster and will be automatically reintegrated in the process.

3.4.5 Support for different versions of servers.

The system is independent on server configuration. This allows having one version of the server program on one machine and a different one on another. The balancer is only concerned about the responsiveness of the server. This is important if the servers from the cluster are gradually updated with new versions of software.

3.5 Load balancing method

3.5.1 Known algorithms

Load balancing of servers can be implemented in various different ways. These methods of load balancing can be setup in the load balancer based on available criteria. There are various algorithms used to distribute the load among the available servers.

Random Allocation In a random allocation, the HTTP requests are assigned to any server picked randomly among the group of servers. In such cases, it may happen that one of the server is assigned many more requests to handle, while the other servers are sitting idle. However, based on the random selection, on an average, each server gets it share of the load.

Pros: It is simple to implement.

Cons: Can lead to overloading of one server while under-utilization of others.

Round-Robin Allocation In case of a round-robin algorithm, the IP load balancer assigns the requests to a list of the servers on a rotating basis. The first request is allocated to a server picked randomly from the group, so that if more than one IP load balancer is involved, not all the first requests go to the same server. For the subsequent requests, the IP load balancer follows the circular order to redirect them. Once a server is assigned a request, the server is moved to the end of the list. This keeps the servers equally assigned.

Pros: Better than random allocation because the requests are equally divided among the available servers in an orderly fashion.

Cons: Round robin algorithm is not enough for load balancing based on processing overhead or if the server specifications are not identical to each other in the server group.

Weighted Round-Robin Allocation Weighted Round-Robin is an advanced version of the round-robin that eliminates the deficiencies of the plain round robin algorithm. In case of a weighted round-robin, one can assign a weight to each server in the group so that if one server is capable of handling twice as much load as the other, the powerful server gets a weight of 2. In such cases, the IP load balancer will assign two requests to the powerful server for each request assigned to the weaker one.

Pros: Takes care of the capacity of the servers in the group.

Cons: Does not consider the advanced load balancing requirements such as processing times for each individual request.

3.5.2 Criteria used for the load balancing policy

APD Load Balancer user 3 criteria for implementing the load balancing policy.

Static criteria

The normal HTTP delay It is introduced by the administrator when the program is installed. This represents a response time that pleases the users. This value should be smaller than 5 seconds.

Dynamic criteria

The HTTP response time

The network delay

How are the dynamic waits measured?

The HTTP response time: This is measured by making a HTTP request and waiting for the response. The page is stored in a file. The script takes the time before the request is made and takes the time after the balancer received a response. The HTTP request is made using functions from PERL libraries.

The network delay: The packet construction features of PERL libraries are used in order to measure the network delay. The application builds a TCP SYN packet with the destination port 80, a random source port (greater than 1024) a random sequence number (less than 64768). This packet will have the source IP the IP of load balancer and the destination IP the IP of one balanced server. The program will wait a TCP ACK packet as a response from the balanced server. The TCP ACK packet must have an acknowledgment number equal with the sequence number sent in the SYN packet plus 1.

The time measured from the moment the TCP SYN packet was sent until the moment a response packet with the characteristics mentioned above was received, represents the network delay.

When are the waits measured ? The load balancer applies the polices every minute. This interval is divided by the number of servers and it is obtained a new interval for each server. In this new interval, the HTTP response time and the network delay are measured. If the time it takes to measure both network delay and HTTP response time is greater than 5 second, the servers are considered unreachable. The rest of time (from 5 seconds to the interval allocated for each server) the script will wait. Unless we have these breaks between requests, the system might crash because this application must function on machines that don't necessarily have a great processing power. After one minute the criteria is available for every server and the algorithm can be applied.

3.5.3 The implemented algorithm

The following data will be referred as:

NR_SERV - Number of servers

Si - the balanced server "i"

HTTP_RESPi - the HTTP response of server "i"

N_DELAYi - the network delay of server "i"

OPTIMUM_DELAY - the delay that is considered convenient for users, introduced by administrator

The algorithm

The load balancing policy is a composed one. All the servers that have the delay smaller than OPTIMUM_DELAY are treated as equals and the Round Robin algorithm described above will be applied using these servers.

If all the servers have the delay greater than OPTIMUM_DELAY, a weighted algorithm will be performed. The weights are represented by the network delay and from the list of reachable servers, the one that has the optimum combination of HTTP Response Time and Network Delay will be preferred.

Initially the requests are redirected to the first server. In every moment the server that receives the requests is known. This server will be referred as the active server.

As it was mentioned before, the algorithm is applied every minute and all the criteria are available. Also, the criteria will change dynamically every minute.

From this point the steps that are performed by the algorithm are:

Find all servers with a HTTP_RESP < OPTIMUM_DELAY;

If servers with HTTP_RESP < OPTIMUM_DELAY exist

The algorithm follows the circular order to redirect the requests considering the servers found at step 1 and will determine the new active server.

If all the balanced servers have the HTTP_RESP>OPTIMUM_DELAY

1.For each server is calculated weighted_response=HTTP_RESP*N_DELAY

2.Calculate the minimum weighted_response

3.The server with the minimum weighted_response becomes Active Server

Example

In order to better understand the process, experimental data from some steps is presented:

NR_SERV=3;

OPTIMUM_DELAY=1;

Following values for the criteria are gathered from each server and the active server is also specified:

- *HTTP_RESP1=0.934 seconds*
- *HTTP_RESP2=0.9 seconds*
- *HTTP_RESP3=0.92 second*s
- Active Server = S2;

In this case the algorithm checks the servers starting with Active Server+1 or with the first server if the Active Server = S3. The order in which the servers are checked is :S3,S1,S2. The first server with the HTTP_RESP < OPTIMUM_DELAY that is found is server S3. Now the Active Server=S3; When the policy will be applied again the check will start from the first server: S1,S2,S3. In this case, because there are servers with the HTTP response time smaller than the optimum HTTP response time the network delay is not taken into consideration.

A second example will be analyzed:

- HTTP_RESP1=1 seconds
- HTTP_RESP2=1.22 seconds
- HTTP_RESP3=1.373 seconds

- N_DELAY1= 23 milliseconds
- N_DELAY2= 23 milliseconds
- N_DELAY3= 35 milliseconds

The calculated weighted response for each server:

- weighted_response 1 = 23
- weighted_response 2 = 28.06
- weighted_response 3 = 48.055

In this case all the balanced servers have the HTTP response time greater than the optimum HTTP response. Because the minimum value for weighted_response is 23 for S1, S1 will become the active server.

3.6 Frequently asked questions

When the development of this application started, some choices were made related to the used tools or related to the way the application was implemented. After working on this application and after the tests, some changes about the criteria were also made. The changes refer to the way the load balancing policy was implemented. This application is implemented as a load balancing method for servers in a LAN, however it can also be extended to servers located in large geographic areas. The development of the application will continue and other criteria we included in the load balancing policy

3.6.1 Why was PERL used?

PERL is a scripting language that is very flexible and offers a lot of possibilities to programmers. First of all it is very easy to work with processes when someone uses perl. It offers a very flexible way to work with files and offers the possibility to take and use the output of system commands very easy. It also gives the possibility to work directly with the system by executing system commands. It has some libraries that contain advanced classes enabling the applications to make HTTP requests, or to build TCP packets encoded with the information the programmer wants, to take and process the packets from the network or to send packets on the network. In order to understand this choice some examples will be presented, highlighting the features of this scripting language .

```
open(SYSTIME,"date  +%s |");

$systime=<SYSTIME>;

close(SYSTIME)
```

These three lines written above are executing the system "date" command, take the output of this command and put this output in a variable called $systime. This sounds very simple, but depending on the programming language that is chosen this may be more or less complicated. PERL does this very easy.

The fact that PERL gives the possibility to work with processes is very important for the application. When the responsiveness of servers is verified, some requests for those servers are sent and the application waits for an response. The question is what happens if the server is not responding? How much time should the program wait? If the server has a problem for a short interval but in a while it will respond, it is ok. In case the connection to the server does not exist anymore the application will wait until that connection comes up again. This could happen in ten minutes, in a hour or in a day. Perl implements an interface with the processes in the system and enables the application to control the process that is blocked.

3.6.2 Why was RRDTool used?

An alternative to RRDTool is MRTG MultiRouter Traffic Grapher

Started as a tiny little script for graphing the use of a connection to the Internet, MRTG evolved into a tool for graphing other data sources including temperature, speed, voltage, number of printouts and the like.

RRDTool originated from MRTG, it is more stable and it has some extra features.

RRD is the Acronym for Round Robin Database. RRD is a system to store and display time-series data (network bandwidth, machine-room temperature, server load average).

In this application, it is used for storing and displaying network delay and HTTP response time.

It stores the data in a very compact way that will not expand over time, and it presents useful graphs by processing the data to enforce a certain data density.

It can be used either via scripts like Perl

Round robin is a technique that works with a fixed amount of data, and a pointer to the current element. Think of a circle with some dots plotted on the edge, these dots are the places where data can be stored. Draw an arrow from the center of the circle to one of the dots, this is the pointer. When the current data is read or written, the pointer moves to the next element. On a circle there is no beginning nor an end, it goes on and on. After a while, all the available places will be used and the process automatically reuses old locations. This way, the database will not grow in size and therefore requires no maintenance. RRDTool works with Round Robin Databases (RRDs). It stores and retrieves data from them.

The application keeps the records of network delay and HTTP response time for the last 10 minutes, for the last day, last week. RRDTool prevents from creating huge databases with data that is not necessary anymore .

It is pretty easy to gather status information from all sorts of things, ranging from the temperature in the office to the number of octets which have passed through the FDDI interface of a router. But it is not so trivial to store this data in a efficient and systematic manner. This is the main reason why RRDTool is used in this program. It allows logging and analyzing the gathered data (network

delay and HTTP response time) and it quickly generates graphical representations of the data values collected over a definable time period. RRDTool will be able to store the data efficiently.

3.6.3 Why is NCURSES used?

NCURSES is a freely distributable library containing functions that manages an application display on character-cell terminals. CURSES is a pan name from Cursor Optimization. The library provides highly flexible and efficient Application Programming Interface and the programmer doesn't have to worry about the underlying terminal capabilities. It provides functions to move the cursor, create windows, produce colors and other desirable features. The fact that is independent of terminal capabilities is very important because this application is intended to provide a solution that can apply to machines with medium computing power. The load balancer may be a UNIX machine that does not have a X Server installed in order to provide high graphic capabilities.

3.6.4 Why is iptables used?

When the load balancing policy is applied the best server is found. The next step is to actually redirect the packets to that server.

This is accomplished using iptables.

Over the past several years, the use of Linux as a firewall platform has grown significantly. Linux firewalling code includes netfilter architecture which was introduced in stable kernel 2.4. Any distribution that comes with a 2.4 kernel has included the iptables tools that is needed to obtain user space program to control the firewall. So it is available on most systems and it is one of the most powerful tools for dealing with the packets from the network

3.6.5 Why is the algorithm applied every minute?

In order to apply the algorithm the values for criteria must be known. For obtaining the criteria the application makes some HTTP requests. The HTTP requests are used in order to obtain some data about the availability of the web server. This requests take some time and considerable processing power. This is proportional with the number of servers.

During tests it was noticed an increase in the load average of the load balancer in the cases when a small interval was chosen. After making the requests for one server the system will wait for an interval which is dependent on the number of servers. This way the system will remain stable and the machine will not crash.

3.6.6 Number of connections

Initially, the number of HTTP connections to each server was one criteria.

This information was obtained from the file "ip_conntrack" which is found on UNIX systems and keeps track of all connections that pass through a machine. During tests an unexpected behavior was noticed.

If the IP address was changed and there was a connection with the old IP, the connection will stuck in the file “ip_conntrack” for an undetermined period of time. When the application will try to find the number of connections to a certain server, it will obtain a false information

If the connection with one of the balanced servers is lost, all the requests that were established to that server in that moment will be kept in the file for an undetermined period of time.

3.6.7 Load of balanced servers

One will say that this is an useful information. If the servers have a high load this means they have a lot of things to calculate, there are a lot of processes waiting to be executed. This will eventually mean that the server will not process our HTTP request very fast.

The load balancing application runs on a UNIX operating system. But the balanced server can run on any type of operating system either UNIX based, Windows, Novel or any type of system. All this operating systems measure the load of the machine they are running on in different ways and it is not correct to compare them. This criteria is not used because the application must be independent of the operating systems running on the balanced servers.

3.7 License

This program is under the GNU General Public License (GPL)

This is the GNU General Public License, publicly available at www.gnu.org/licenses:

Version 2, June 1991

Copyright c ! 1989, 1991 Free Software Foundation, Inc.

59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software - to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation’s software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not react on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that re-distributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and

to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

(a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

(b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

(c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

(a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

(b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-

readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

(c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sub-license, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sub-license or distribute the Program is void, and will automatically terminate your rights under this License. However,parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program species a version number of this License which applies to it and “any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD

THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

one line to give the program’s name and a brief idea of what it does.

Copyright (C) yyyy name of author

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation,

Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

Also add information on how to contact you by electronic and paper mail. If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) yyyy name of author

Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.

This is free software, and you are welcome to redistribute it under certain conditions; type ‘show c’ for details.

The hypothetical commands show w and show c should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than show w and

show c; they could even be mouse-clicks or menu item–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program ‘Gnomovision’ (which makes passes at compilers) written by James Hacker.

.

signature of Ty Coon, 1 April 1989

Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Chapter 4

Architecture and Components

APD Load Balancer has 3 main parts:

1. The ncurses based application written in C programming language for configuring the application
2. The PERL scripting part that implements the load balancing policy.
3. The web pages with the graphs for monitoring and viewing the results

The package APDlb contains

- One C source file graphint.c
- A Makefile, user for compiling the application
- A configuration file called servers.cfg
- One documentation file README that will be discussed in Chapter 5
- A web page “page.html” - the file where the web page from the servers is copied
- The www folder containing

1. the graphs created by RRDTool:
2. the web pages used for monitoring from which the graphs are accessed

- Perl scripts:

1. APDlb.pl
2. HTTPdelay.pl
3. ndelay.pl
4. control.pl

5. install

- Some files used by the application for transmitting some parameters between scripts:

1. HTTPtime
2. ntime
3. flag

- The RRDTool creates the files:

1. monmh.rrd
2. mond.rrd
3. monw.rrd

These are the RRDTool databases.

4.1 The ncurses based application

4.1.1 General

The C application contains one source file called graphint.c.

The “Makefile” used to compile the application:

all: graphint

graphint: graphint.c

gcc *.c -o graphint -lncurses -lpanel -lform

clean:

rm -rf graphint *.o

When using “make all” the compiler will read the file “Makefile” and will generate the executable program “graphint”. This executable will be launched by the installation scrip.

4.1.2 Libraries

The libraries used are:

#include<ncurses.h>

#include<panel.h>

#include<form.h>

#include<string.h>

#include<stdlib.h>

#include<malloc.h>

#include<stdio.h>

4.1.3 Components

Here are some details about the graphint.c file. A short description of the functions and the parameters they use will be presented.

createwin - creates a window

Parameters:
int height - height of the window
int width - width of the window
int startx - the left corner horizontal coordinate
int starty - the left corner vertical coordinate
int cp - the color pair used for the window, the colors for background and foreground
int b- if the window will have a border or not

- WINDOW* createwin (int height, int width, int startx, int starty, int cp, int b);

destroy - destroys a window

Parameters:
WINDOW* localwin - the window to be destroyed

- void destroy (WINDOW* localwin);

print_in_win - prints a string at given coordinates in a given window

Parameters:
Window* win - the window in which the string is written
int startx - the horizontal coordinate for the string that has to be printed
int starty - the vertical coordinate for the string that has to be printed
char* string - the string that has to be printed in the window

- void print_in_win (WINDOW* win, int starty, int startx, char* string);

dig_to_int - transforms some digits from strings to integers. The function return the integer value of the transformed string.

Parameters:
char* string - the string of digits that must be transformed

- int dig_to_int (char* string);

createpanel - creates a panel that contains the fields in which the user will introduce the informations, a main window in which the fields will be, a window for accepting the informations and one window for rejecting the information. Four types of panels will be created. The function returns an integer representing the type of panel that was created.

Parameters:

WINDOW* fereastra - the main window of the application.
WINDOW* reject - the window for rejecting the information
WINDOW* accept - the window for accepting the information
WINDOW* smallwin - a subwindow in which the forms with the fields will be created
FIELD** field - the fields that will be created
FORM* my_form - the form that will contain the fields
PANEL* my_panel - the panel in which all the components will be put
int nr_of_fields - the number of fields in which the user will introduce the information
int identif - an identifier for the type of panel
char** info - keeps the information inserted by the user.

- int createpanel (WINDOW* fereastra, WINDOW* accept, WINDOW reject, WINDOW* smallwin, FIELD**FIELD, FORM * my_form, PANEL* my_panel, int nr_of_fields, int identif, char** info);

int main() - uses all the functions

This is a pseudo-code for the basic steps in this function:

int main() {

- Initialize ncurses
- Initialize the colors
- color background

//The application needs 4 panels for introducing all the necessary informations
while(i<number_of_panels){

1. Create panel number “i”
2. If the user accepted the information he introduced -> Go on, next panel or finish
3. If the user rejected the information -> create the panel number “i”

}

- Print the information introduced by the user

- Ask if it is correct
- If the user rejects the information the panels will be created again
- If the user accepts the information this will be written in the file “servers.cfg”

}

4.2 PERL scripting

In this section a detailed description of every script will be presented.

4.2.1 The APDlb.pl script

It is the most important script of the application. It is used with one command line argument. The values of the argument can be “start” for starting the application, “stop” for stopping the application. For any other values, the user is notified about the way the script should be used.

Starting and stopping the application

If the argument is “start” the script will run normally as described below. In the beginning the value “0” will be written in the file “flag”. Every minute, before measuring the criteria for the load balancing algorithm, the value from this file is checked. If it is “0”, the script will continue. If it is “1”, the script will stop.

As long as the value from the file “flag” is “0”, the script will loop every minute. During one step it gathers the criteria, plots it, applies the load balancing algorithm.

If the argument is stop, the file “flag” will be opened for writing in it the value “1”. So, next time the script will try to gather the criteria it will stop. This will not take more than a minute.

Subroutines

sub str_to_int - transforms a string into an int

Takes a string as a parameter and returns the integer value.

sub modpage - modifies a html page

Takes as arguments the page that must be modified and a string that will be inserted in the page.

Details

All the steps that are performed and the way the script connects with the other scripts are presented.

Defining colors

At the beginning is created a list of colors used for plotting the HTTP response time and the network delay .

Obtaining the input

The configuration file is opened and the values that were introduced by the user during configuration are extracted.

The variables that are used for this data are:

- $rrd - location of RRDTool
- $monitor - location of www folder where we have the monitoring pages
- $nrserv - the number of balanced servers
- $iptables - location of iptables
- $lbip - IP address of the load balancer
- @serverslist - the list of IP address for all the balanced servers
- $dev - the network interface card used to connect to servers

Modifying the html monitoring pages

When the user introduces the input data, he introduces the IP addresses of the balanced servers.

- A line with the IP addresses of the balanced servers must be written in every monitoring page. Every IP must be printed with the color used in the graph for plotting the network delay or HTTP response time of the balanced server to which the IP belongs. This way the user can identify very easy which informations are for a certain server.

The pages are modified with the subroutine“modpage” . The subroutine has 2 arguments. The first is the string that will be copied in the HTML page and the second is the HTML page that has to be modified. In order to modify the page, each line is copied in a list. The application changes the 11th line in the HTML file by replacing the 11th element in the list with the string received as parameter. The HTML file is opened again, this time for writing, and every element from the modified list will be written on its own line. Now the page is modified. Here is the code for this subroutine:

```
# WRITE THE IP ADDRESS OF THE SERVERS WITH THE SAME COLOR USED IN GRAPH IN THE HTML PAGE #
sub modpage
{
local ($string,$pname)=@_;
local @list=($string);
```

```
open(mon,"$pname")||die("Can't open file mon $!");
while($line=<mon>){
push(@raw,$line);
}
close mon;
splice(@raw,11,1,@list);
open(mon,">$pname")||die("Can't open file $!");
print mon @raw;
close mon;
$nrelem=@raw;
$lelem=@list;
splice(@list,0,$lelem,());
splice(@raw,0,$nrelem,());
}
```

It is very important to know how the pages are modified if the users want to add or to delete some lines from the monitoring page.

The application will open the file that contains the monitoring page and will put the content in a list. The line that has to be modified in the monitoring pages that come with the application is the 11th line. This line is replaced with a string that is created dynamically depending on the number of servers and the modified list is copied again in the file.

In order to modify this pages there are two choices: either the 11th line will remain the same, either the function in the script will be modified.

For example, if 3 lines are added before the 11th, then, in the PERL script, the line "splice(@row,11,1,@list)" will be modified to "splice(@row,14,1,@list)".

Creating the RRDTool Database

The first step is to dynamically create the arguments. These depend on the number of servers. If the number of balanced servers is "n", the RRDTool databases will have "2*n" DS (Data Sources). "n" DSs are for the HTTP response time and "n" for the network delay.

- The Data Sources (DSs) are:

For HTTP response time

```
DS:ip1:GAUGE:120:U:U
.
.
DS:ipn:GAUGE:120:U:U
For network delay:
DS:nip1:GAUGE:120:U:U
.
.
```

DS:nipn:GAUGE:120:U:U

The RRDTool databases will have data sources with Data Source Type GAUGE and if no value is obtained for a data source in 120 seconds, that value is marked as unknown.

- The step is 60 seconds

The initial time, needed as parameter for RRDTool create, is taken using the system “date” command.

Three databases are created:

- monmh.rrd - for monitoring the delays in the last 10 minutes and in the last hour.
- mond.rrd - for monitoring the delays in the last day
- monw.rrd - for monitoring the delays in the last week

The first RRDTool database has two Round Robin Archives :

- RRA:MAX:0.5:1:12
- RRA:AVERAGE:0.5:5:14

The first archive will update the data every 1*step (1*60 seconds) interval and will keep 12 updates. This means the data is available for the last 12*60=720 seconds. Because the consolidation function is MAX, it will update the database with the maximum value for each 60 seconds interval.

The second RRA will update the data every 5*step (5*60seconds=300seconds=5 minutes) interval and will keep 14 updates. This means the data is available for the last 14*300=5200 seconds. Because the consolidation function is AVERAGE, it will update the database with the average value for each 5 minutes interval.

The second RRDTool database has one Round Robin Archive:

- RRA:AVERAGE:0.5:60:25

This means it will update every 60*step (60*60=3600 second=1 hour) and will keep the last 25 updates. The data is available for the last 25 hours.

The third RRDTool database has one Round Robin Archive:

- RRA:AVERAGE:0.5:1440:8

This means it will update every 1440*step (1440*60=86400 second=24 hours) and will keep the last 8 updates. The data will be available for the last 8 days.

Redirecting the requests (initially to the first server)

At the beginning the requests are redirected to the first server. The PERL scrip will run the “iptables” command using the following line:

system(“$iptables -t nat -I PREROUTING -p tcp -d $lbip/32 –dport 80 -j DNAT –to-destination $serverslist[0]:80 ”);

The scrip will replace $iptables, $lbip and $serverslist[0] with their values (location of iptables ex: /usr/sbin/iptables, IP address of load balancer, the IP address of the first server).

The options used with iptables command:

-t nat specifies the table NAT

-I insert a rule in the NAT table

PREROUTING specifies the chain in the NAT table

-p tcp it will match the packets that use the TCP protocol

-d $lbip it will match the packets that have the destination IP address $lbip

–dport 80 it will match the tcp packets that have the destination port 80 (i.e. HTTP packets)

-j DNAT defines the target (Defines what to do with the packet). This target will replace the destination address of the packet.

–to-destination $serverslist[0]:80 replaces the destination IP address from the packet with the IP address of the first balanced server.

After this command all HTTP packets that come for the load balancer on port 80 will be taken, their destination IP address will be modified and they will be resent to the balanced servers.

Gathering the criteria

For every server, the HTTP response time and the network delay are calculated using the script “control.pl”.

This script will be presented in the next subsection. For each server the script “control.pl” will be launched and the HTTP response time will be available in the file “HTTPtime” and the network delay will be available in the file “ntime” . In order to apply the algorithm using these values, they must be transformed from strings to numbers using the function “str_to_int()” .

After the script will run for one server, the program will stop for ($interval-5) seconds. The $interval variable is obtained dividing one minute by the number of servers.

Updating the database

In this moment the criteria is available and the script must update the Round Robin Database in order to plot these values.

The argument for updating all the databases is created automatically and it depends on the number of servers.

For the time stamp of the update the script uses“N” which specifies to the RRD Database that the value is measured at the current moment.

Implementing the algorithm

Here is the code used to implement the algorithm:

- #Verify the delay for all servers
- # Take the servers with the HTTP response time smaller than the optimum response
- #Apply the Round Robin Algorithm for these servers

.

$k=0;

- # If $k=0 all the servers have the HTTP response greater than the optimum HTTP response introduced by the user

$counter=0;
$end=$nrserv;

- # $end keeps the number of the server where the algorithm should stop. Initially it is the last server. If the second server is the active server, the order in which the algorithm tries to find the best server is 3, 4, ..., n,1,2. After the last server is checked, this variable is modified: $end=2.
- # $table keeps the number of active server (The one that currently receives the requests)
- #The algorithm starts with the next server if the active server kept in variable $table is not the last one, $i=$table+1.
- #If the active server is the last one, then $i<=0, so the algorithm will start with the first server.

$i=$table+1;
if($table==($nrserv-1)){
$i=0;
}

- #the requests are redirected to the first server that has the HTTP response time smaller then the optimum one and the active server is updated: $table=$i

```
while(($i<$end)&&($counter<$nrserv)){
    $counter++;
    if($delay[$i]<1){
    system("$iptables -t nat -D PREROUTING -p tcp -d $lbip/32 –dport 80 -j DNAT –to-destination $serverslist[$table]:80" );
    system("$iptables -t nat -I PREROUTING -p tcp -d $lbip/32 –dport 80 -j DNAT –to-destination $serverslist[$i]:80");
    $table=$i;
    $i=$end;
    $k=1;
    }
    else{
    if($i<($end-1)){
    $i++;
    }
    else{
    $end=$table+1;
    $i=0;
    }
    }
    }
```

- # If all servers have the HTTP response time greater than the optimum response
- # 1.Take the servers that are reachable
- # 2.Find the server with min (HTTP response time * network delay)
- # 3.Redirect the requests to this server

```
if($k==0){
    $i=0;
    while(($weight[$i] eq "UNREACHABLE")&&($i<$nrserv))
    {
    $i++;
    }
```

- # $i will indicate now the first reachable server

```
if($i<$nrserv){
    $weighted_response=$delay[i]*$weight[i];
    $poz=$i;
    }
```

- # Calculate the minimum value for $weighted_response considering only the reachable servers

```
while($i<$nrserv){
    if($weight[$i] ne “UNREACHABLE”){
    $temp=$delay[$i]*$weight[$i];
    if($temp<$weighted_response){
    $weighted_response=$temp;
    $poz=$i;
    }
    }
    $i++;
    }
```

- # Minimum weighted_response and the server with this value are known
- # Redirect the request to this server
- # Update the current server;

```
system("$iptables -t nat -D PREROUTING -p tcp -d $lbip/32 –dport 80 -j DNAT –to-destination $serverslist[$table]:80");
    system("$iptables -t nat -I PREROUTING -p tcp -d $lbip/32 –dport 80 -j DNAT –to-destination $serverslist[$poz]:80");
    $table=$poz;
    }
```

- #Empty the list with HTTP response time

```
splice(@delay,0,$nrserv,());
```

- # End of algorithm

Plotting the data

In this script eight graphs are created.

- mons.gif
- nmons.gif

These graphs plot the data from the last 10 minutes.

In order to keep this interval the graph is shifted.

The end time for the graph is the current moment.

If the end_time - start_time>10 minutes (600 seconds) then the start_time=end_time-600;

The x-axis and grid is configured using the following format:

-x SECOND:$step:MINUTE:2MINUTE:2:0:%x The horizontal axis will have:

- a base grid for every $step (60 seconds) interval.

- a main greed for every 2 minutes

- a label for every 2 minutes

For both graphs background color is defined with the RGB code #cae1ff.

The title for the first graph is “HTTP-Response-Time” and the vertical label is “seconds”, because the HTTP response time is measured in seconds.

The title for the second graph is “Network-Delay” and the vertical label is “milliseconds”, the network delay is measured in seconds.

For both graphs the values used are stored in monmh.rrd Round Robin Database and the Round Robin Archive that stores the data for the last 10 minutes. The values coming from Data Sources referring to different servers will be plotted with different colors.

- monh.gif
- nmonh.gif

These graphs plot the data from the last hour.

The end time for the graph is the current moment. The graph is shifted in order to maintain this interval. The end time must be with 3600 seconds above the start time.

The x-axis and grid are configured using the following format:

-x MINUTE:5:MINUTE:10MINUTE:10:0:%x This graphs are using the values from “monmh.rrd” but Round Robin Archive is the one with the consolidation function AVERAGE keeping the values from the last hour.

The graph will update every 5 minutes.

- mond.gif
- nmond.gif

These graphs plot the data from the last day.

The x-axis and grid are configured using the following format:

-x HOUR:1:HOUR:2:HOUR:4:0:%x This graphs are configured using the values from “mond.rrd” database and will update every hour.

- monw.gif
- nmonw.gif

These graphs plot the data from the last week.

We configure the x-axis and grid using the following format:

-x DAY:1:DAY:1:DAY:1:0:%x This graphs are using the values from “monw.rrd” database and will update every day.

The script will continuously loop every minute unless it is stopped by the user. In the loop the following steps are performed:

- gather the criteria
- update the database
- apply the algorithm
- plot the results

4.2.2 The control.pl script

This script is launched from the script “APDlb.pl” . It’s main purpose is to control the scripts “HTTPdelay.pl” and “ndelay.pl”. When it runs it has as the first command line argument the IP address of the server for which the HTTP response time and the network delay will be measured and as second argument, the network interface card used to connect to the balanced servers.

It uses 2 files: “HTTPtime” and “ntime”.

HTTPtime - it is used to write the HTTP response time.

ntime - it is used to write the network delay.

How does it work?

The first step is to take the IP address and the device name from the command line .

The next step is to create two separate processes. The processes are created using fork().

In the child process :

1. At the beginning it is considered that the server did not respond and the value 5.000 is printed in the file “HTTPtime”.
2. The scripts “ndelay.pl” and “HTTPdelay.pl” will be started.
3. The time it took to make a HTTP request will be measured and the value will be written in the file “HTTPtime” . Now, if the requests was successful, the file “HTTPtime” will contain the real HTTP delay (at the beginning it was considered that the server has a delay of 5 seconds). However, the script also checks if the web server responded with an error message. In this case, the time it took to make the request could be small but the server will be unreachable.

In order to check if an error occurred the file “HTTPtime” is opened and the first line is checked. If the line is:

- 5.000 - no error occurred. The time it took to make the request and to receive an answer can be written.

- 5.000 1 - this is an error response code. In this case the file “HTTPtime” will not be modified.

After reading the “HTTPtime” file, its content will be parsed.

In order to verify the content of the line the following regular expression is used:

$serverup=~/\d.\d{1,3}\s\d/.

This expression is true if the line contains: 1 digit, a dot followed by 1,2 or 3 digits, a space and another digit. “5.000 1” will match the expression (this way the script knows that an error occured). “5.000” will not match the regular expression.

If the server for which the HTTP response time and the network delay is measured is disconnected from the network, the first script will be blocked waiting for a response packet from the server. The code written in the parent process will avoid this situation.

In the parent process: .

1. Wait for 5 seconds.
2. Open the file “HTTPtime”.
3. If the string from the file is not greater or equal with “5.000” the script will finish. No error occurred and the web server responded. The value of the HTTP response time is written in this file.
4. If the string from the file is greater or equal with “5.000” check if an HTTP error occurred. The same criteria described before will be used.

- If there is an HTTP error the script will open the file “HTTPtime” and will write “5.000” in stead of “5.000 1”.
- If there is no HTTP error, this means that the server did not respond in 5 seconds and one of the scripts “HTTPdelay.pl” or “ndelay.pl” is blocked waiting for an response from one unreachable server. The scripts are stopped using “killall” system command: system(“killall -9 ndelay.pl 2>/dev/null”);
- Open the file “ntime” and write “UNREACHABLE”.

In the child process the script “ndelay.pl” will run first. If the server is unreachable the script will remain blocked. If this happens, the parent process, after will wait for 5 seconds, will read from the file “HTTPtime” and will see the value “5.000”. This value indicates no response from server and the script “ndelay.pl” will be killed.

4.2.3 The HTTPdelay.pl script

This script is used to make an HTTP request for a web server and if the server is working, the web page is taken and we copied it in the file “page.html” from the disk. This is what a normal user does. The only difference is that when someone makes the request, the requested page is not printed in a file, it is printed on the screen.

The script takes as command line arguments the IP address of the target balanced server.

Libraries and classes :

- use LWP::Simple
- use HTTP::Status

How does it work?

1. The first step is to take the IP address of the web server from the command line.
2. The second step:

is_error(getstore($ip,"page.html"))

- The variable $ip has the form: "http:// 192.168.2.1".
- getstore is the function that makes the request and tries to copy the web page in the file "page.html".
- is_error will check if the response is the requested page or one error codes .

In case of error, the file "HTTPtime" will be opened for writing the line "5.000 1", which will indicate an HTTP error.

4.2.4 The ndelay.pl script

This script makes a TCP request to one of the balanced servers in order to measure the network delay. The value of this delay is written in the file "ntime".

It takes the following command line arguments:

- the device used to send the packet
- the source IP address - the IP address of the load balancer
- the destination IP address - the IP address of a balanced server

Libraries and classes :

use Math::Round;
use Net::Pcap;
use NetPacket::Ethernet qw(:strip);
use NetPacket::IP qw(:strip);
use NetPacket::TCP;
use Net::RawIP;
use Time::Stopwatch;
use Socket

Subroutines

sub syn_packets - used to handle the packets that are captured from the network while waiting a response packet from the balanced server.

How does it work?

TCP, IP, Ethernet (very basics)

Transmission Control Protocol (TCP) is a connection oriented transport layer protocol. It establishes a connection between the source and the destination, it is responsible with taking the data created by applications, segmenting it into parts that can be transmitted, adding information about the other protocols used in applications, adding sequence and acknowledgment numbers to the packets so that they are delivered in the same order they were transmitted and reassembling the data at destination. TCP provides virtual circuits between end-user applications. The Internet is based on the TCP protocol. The HTTP protocol relies on TCP to deliver the data across the network.

When a TCP packet is built, apart from other information it sends the following fields :

- source port - number of the calling port
- destination port - number of the called port
- sequence number - number used to ensure a correct sequencing of the data
- acknowledgment number - The expected octet

TCP is a connection-oriented protocol. It requires connection establishment before data transfer begins. For a connection to be established or initialized, the two hosts must synchronize their Initial Sequence Numbers (ISNs). Synchronization is done through an exchange of connection establishing segments that carry a control bit called SYN, for synchronize, and the ISNs. Segments that carry the SYN bit are also called “SYNs”. This solution requires a suitable mechanism for picking an initial sequence number and a slightly involved handshake to exchange the ISNs.

The synchronization requires each side to send its own initial sequence number and to receive a confirmation of exchange in an acknowledgment (ACK) from the other side. Each side must also receive the INS from the other side and send a confirming ACK.

If A and B want to communicate the sequence is as follows:

1. A->B SYN – (A) initial sequence number is X, ACK number is 0, SYN bit is set, but ACK bit is not set.
2. B->A ACK – (A) sequence number is X + 1, (B) initial sequence number is Y, and SYN and ACK bit are set.
3. A->B ACK – (B) sequence number is Y + 1, (A) sequence number is X + 1, the ACK bit is set, but the SYN bit is not set.

This exchange is called the three-way handshake.

The TCP packets are sent to the IP protocol that will add a Source and a destination IP address. These are necessary for any end systems that want to communicate in order to locate each other.

The packet is next sent to the lower layer protocols that usually depend on the technology used to connect the devices. The most common technology for connecting computers on a LAN is Ethernet. So the packet will receive other information in order to be possible to be delivered to the destination.

Steps performed in the script

1.Taking the arguments from the command line

$dev the device used to connect to the balanced servers

$src the source IP address

$dest the destination IP address

2.Creating a New Packet Object.

$packet=new Net::RawIP;

3.Creating the SYN packet

The application creates a TCP packet that has the following values in the fields:

- destination port - port 80 used for HTTP
- source port - a random port greater than 1024 and smaller than 65535
- sequence number - an random number written on 32 bits (the number of bits from TCP sequence number field)

.

The network layer information are added to the packet:

- source IP address - the IP address of the load balancer
- destination IP address - the IP address of one of the balanced servers (the IP address of this server was transmitted as an argument in the command line).

4.Capturing packets from device dev:

4.1 Look up address information

The network number and the netmask for the device specified in “$dev” are determined .

Net::Pcap::lookupnet($dev, \$address,\$netmask, \$err). The function returns 0 on success and sets "$address" and '$netmask" parameters with values. On failure returns -1 and the "$err" parameter is filled with the error message.

4.2 Create a packets capture descriptor:

$object=Net::Pcap::open_live($dev, 1500,0, 0.3,\$err); The function returns a packet capture descriptor for the packets captured from device $dev, the maximum number of bytes to be captured is 1500, the read timeout is 0.3 milliseconds. The packet descriptor will be undefined if an error occurs and the $err parameter will be filled with the error message.

4.3 Compiling a packet filter:

After defining a packet capture descriptor the application will be able to capture the packets from the network. The program only needs the HTTP packets and processing the whole traffic is not efficient. A filter is defined in order to keep only these packets.

$filter_string="host $dest && port=80";

Net::Pcap::compile($object,\$filter,$filter_string, 0, $netmask); The filter is compiled using $filter_string and this filter will be stored in the variable "$filter" . The filter is not optimized. The function returns 0 if the compilation was successful or 1 it there was a problem.

4.4 Setting the filter:

Net::Pcap::setfilter($object, $filter) The compiled filter is associated with the packet capture descriptor.

4.5 Packet capturing

In order to capture the packets two processes are created:

- in one a packet capture loop is initialized and callback function is set
- in the second the SYN packets that were built before are sent on the network

The next step is to capture the packets until a TCP ACK response packet is received.

Capturing the packets:

Net::Pcap::loop($object,-1,\&syn_packets,' '); The function loops until an error occurs and after each captured packet it will call the subroutine "syn_packets" which will process the packets;

Sending the syn packets :

$packet->send;

4.6 Handling the captured packets:

The packets are captured because it is necessary to know when a response packet was received from the balanced server. The HTTP packets which have the destination or source IP equal to“$dest” (these were defined in the filter) are further checked. For the TCP ACK response packet, the acknowledgment number from the TCP header must be the sequence number sent in the packet built by the script plus 1 .

Transport layer informations must be checked.

When the packet is captured, it has layer 2 informations, which means information used by the Ethernet technology. The Ethernet and the IP headers and tailer will be striped or decoded until the fields from the TCP header are reached:

$ip_obj=NetPacket::IP->decode(eth_strip($pkt));

$tcp_obj=NetPacket::TCP->decode($ip_obj->{data});

When a packet that matches the criteria specified above will be received, the script will measure the time from the moment the SYN packet was sent until the moment the ACK packet was received. This value will be written in the file “ntime”.

4.2.5 The installation script

Code:

```
#!/usr/bin/perl
```

- #run the configuration program

```
system("./graphint");
    print "If you did not finish the configuration process press n !!!!!!\n";
    while(($var ne "y")&&($var ne "n")){
    print "Do you want to continue the installation? (y or n)";
    $var=<STDIN>;
    chop $var;
    }
```

- #if the user pressed <y> the installation will continue

```
if($var eq "y"){
    print "type the instillation path: ";
    $path=<STDIN>;
    chop $path;
    print "Creating the $path/APDlb-1.0 folder \n";
```

- #if such a folder already exists, it is removed and a new one is created

```
system ("rm -r $path/APDlb-1.0 2>/dev/null");
    system ("mkdir $path/APDlb-1.0");
```

- #The scripts and all the other needed files are copied in the installation path
- #The rights for the PERL scripts are modified

```
print "Installing APDlb.pl\n\n";
    system ("cp APDlb.pl $path/APDlb-1.0");
    system("chmod 744 $path/APDlb-1.0/APDlb.pl");
    print "Installing control.pl\n\n";
    system ("cp control.pl $path/APDlb-1.0");
    system("chmod 744 $path/APDlb-1.0/control.pl");
    print "Installing ndelay.pl\n\n";
    system ("cp ndelay.pl $path/APDlb-1.0");
    system("chmod 744 $path/APDlb-1.0/ndelay.pl");
    print "Installing HTTPdelay.pl\n\n";
    system ("cp HTTPdelay.pl $path/APDlb-1.0");
    system("chmod 744 $path/APDlb-1.0/HTTPdelay.pl");
    print "Installing ntime\n\n";
    system ("cp ntime $path/APDlb-1.0");
    print "Installing HTTPtime\n\n";
    system ("cp HTTPtime $path/APDlb-1.0");
    print "Installing servers.cfg\n\n";
    system ("cp servers.cfg $path/APDlb-1.0");
    print "Installing flag \n\n ";
    system("cp flag $path/APDlb-1.0");
    open(servers,"servers.cfg");
```

- #Open the configuration file in order to find the path for the monitoring folder

```
$i=0;
    while($i<2){
    $monpath=<servers>;
    chop $monpath;
    $i++;
    }
```

- #The monitoring folder is copied in the location specified by the administrator during the configuration process

```
print "Installing www in $monpath \n";
    system ("rm -r $monpath/www 2>/dev/null");
    system("cp -r www $monpath");
```

- #The user is notified about the perl libraries that need to be installed

```
print ("Uses Math::Round perl library \n");
    print ("Uses Net::Pcap perl library \n");
    print ("Uses NetPacket::Ethernet perl library \n");
    print ("Uses NetPacket::IP perl library \n");
    print ("Uses NetPacket::TCP perl library \n");
    print ("Uses Net::RawIP perl library \n");
    print ("Uses Time::Stopwatch perl library \n");
    print ("Uses LWP::Simple perl library \n");
    print ("\n\n\n");
    print ("Start the program with $path/APDlb-1.0/APDlb.pl start\n\n");
    print ("Stop the program with $path/APDlb-1.0/APDlb.pl stop \n\n\n");
    }
```

Description:

The first step in the installation is to run the program “./graphint” . This executable is obtained after typing “make all”. The graphic configurator tool will start and the user will introduce the input data. If the user wants to abort the installation during the configuration process, he must press <CTRL-C>. The user is asked if he wants to continue the installation process. In case the configuration process was aborted the option should be <n>.

The next step is to introduce the path where the script will be installed. The program will create the directory “APDlb-1.0”. If this directory already exists, it will be replaced with the new one.

The scripts and the other files that are used are copied in this directory one by one.

The “www” - monitoring folder is copied in the location introduced by the user during the configuration process.

In case the user did not read the documentation or the “readme” file he is notified about the PERL libraries that need to be installed before starting the program.

In the end the user will know the commands necessary to start or to stop the application.

4.3 Summary

This chapter presents the architecture of the application, all its files and the way they connect. Every step performed in the application is explained with details. Combined with Chapter 2, this chapter could be considered as a manual for those who want to develop similar applications.

Chapter 5

Practical use

5.1 Presentation of the developed software

This section presents the user interface of the application and will explain the correct way to use it. The user interface has two parts:

- The graphic configurator
- The monitoring page

The graphic configurator is used by the administrator in order to install the application. In this part the user will introduce the input data necessary for the scripting part. If this information is not correctly introduced, the application will not work properly or unexpected error messages could be received from the operating system. The interface is easy to use but the administrator must have some basic knowledge about the load balancing procedure and must know very well the topology and the environment where the application will be installed. Before the installation, the user must read the "README" file provided in the installation package, or the user guide presented in the next section. Before starting the application all the tools and libraries mentioned in the "README" file should be installed.

The monitoring pages provide informations about the servers responsiveness. This way everyone can always see the results of the application, can see the way in which the HTTP response time and the dynamic weight will change during different amounts of time. If a web server runs on the load balancing machine the servers can be monitored from remote locations. Every user can access these pages.

5.1.1 The graphic configurator

The graphic configurator has four panels. In each panel the user must fill in the information that are required. Here it is presented a step by step configuration example:

In the first panel the user is asked about the location of the tools or programs the application uses.

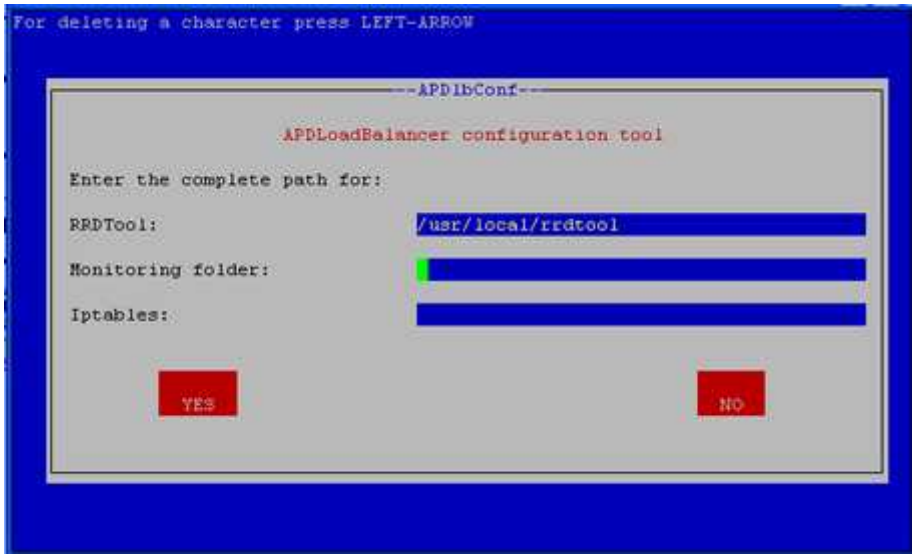

Figure 9

The location of RRDTool and of iptables are required. Later on, the folder “www” will be copied in the location introduced at “Monitoring folder:”.

If the user makes a mistake, he can delete the characters from a field using <left arrow>. Using the <up arrow> the cursor is moved between fields. Moving from the fields area to “YES” or “NO” is performed by pressing <TAB>. In order to cancel all the data that was introduced at a certain moment in this panel press <NO> and a new panel with empty fields will be created. If the data is accepted, press <YES> and the configuration will continue with the next level .

In the second panel the topology dependent informations are introduced.

Figure 10

All computers will associate this IP address of the load balancer with “www.loadbalancedsite.com”.

Number of servers is the field that is verified by the application. If other characters except digits are inserted, all the information inserted in this panel will be erased and a new empty panel will

appear.

The device connected to the server represents the network interface card used to connect to the balanced servers.

In the third panel the user must insert the IP addresses of each balanced server.

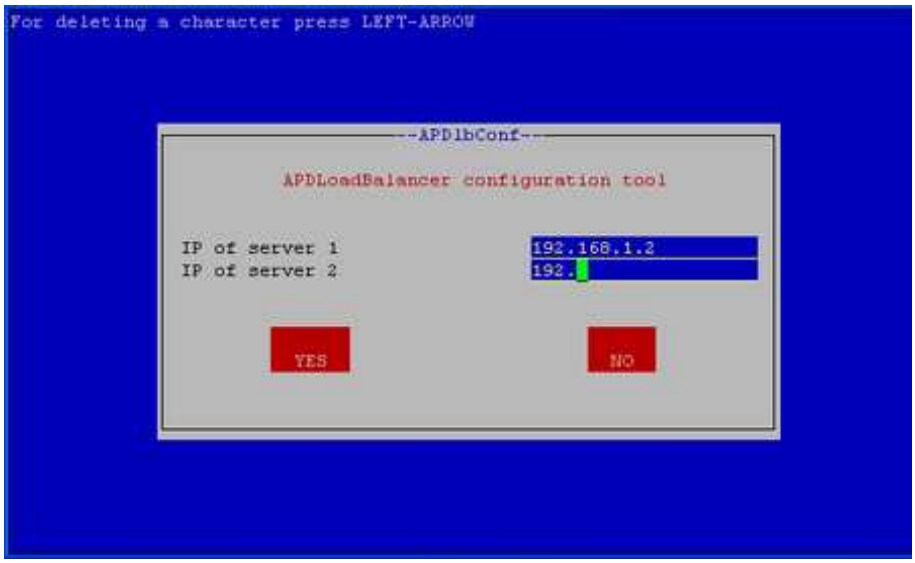

Figure 11

The program will accept any data that is introduced. It is the users concern to type a valid IP addresses. If the introduced IP addresses are not valid, those servers will be considered unreachable and will never be used.

The 4th panel asks the user about the optimum HTTP response time.

This is expressed in seconds. All the servers that have a HTTP response time smaller than the optimum HTTP response time will be treated equally and the algorithm will prefer these servers. The unexperienced users should type the recommended value. The value must be smaller than 5 because after 5 seconds a server is considered unreachable.

Figure 12

After all data will be introduced, a window will appear and all the information that will be used will be printed in there. It is the administrator's responsibility to check if the information is correct or not. This information will be written in a configuration file called "servers.cfg". It is very important to introduce correct data. If the data is not correct the application will not work properly and the user might notice unexpected behavior after a period of time

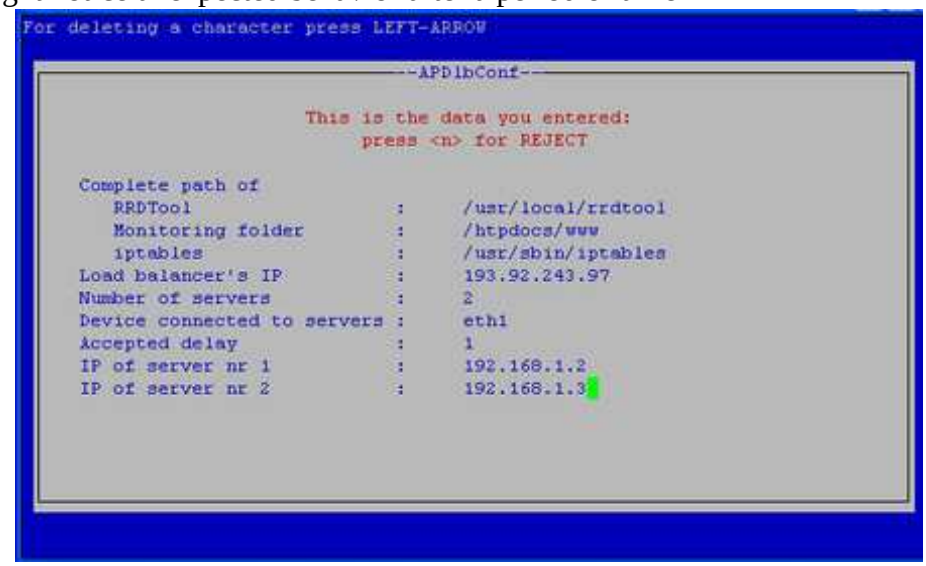

Figure 13

For rejecting the data, the user must press <n>. If the data is correct, any key can be pressed. This was the last step in the configuration process.

5.1.2 The monitoring part

The monitoring part contains four HTML pages. The monitored variables are the HTTP response time and the network delay. In the first page both variables are plotted for the last 10 minutes and the graph updates every minute. The second page monitors the average values for the two criteria mentioned above during the last hour with the graph updating every 5 minutes. The third page will present the data during the last day with the graph updating every hour and the forth page during the last week with the graph updating every 24 hours .

The next figure will present the page:

Figure 14

Each balanced server has an allocated color. In the upper part, the IP addresses for each server are written with different colors. In the graph, the criteria for every server is plotted with the same color that was used to write the IP addresses. The horizontal axes also presents the time labels so the user will know exactly the moment in which a certain value was reached for one of the variables. The process of updating the graph is made according to the time from the load balancer.

Each web page will update every minute.

In the left side there are the links that allow the user to switch between the different monitoring pages.

5.2 User guide

Here is the README file included with the application.

APD Load Balancer

Introduction

The APD Load Balancer is a program for UNIX operating systems that provides load balancing capabilities between web servers.

The main focus of this application is to offer a simple and inexpensive solution for sharing the WWW traffic between servers in a local network.

The maximum number of balanced servers is 6.

It is based on ncurses and PERL and uses RRDTool, iptables and data system command.

Name

The name of the program comes from the Universities that were involved in this project: “Polytechnic University of Bucharest” and “Democritus University of Thrace” —- A PUB and DUTH Load Balancer.

Requirements

In order to install and use this package you will need:

PERL version 8.4 or later

Source code: http://www.perl.com/download.csp#unix
Readme file: http://www.perl.com/CPAN/src/README

WWW library for PERL

Source code: http://search.cpan.org/dist/libwww-perl/
Readme file: http://search.cpan.org/src/GAAS/libwww-perl-5.79/README

Net::Pcap PERL library

Source code and readme file: http://search.cpan.org/~plister/Net-Pcap-0.01/

NetPacket PERL library

Source code and readme file: http://search.cpan.org/timpotter/NetPacket-0.0.3/

Net::RawIP PERL library

Source code and readme file: http://search.cpan.org/~skolychev/Net-PawIP-0.1/

Math::Round PERL library

Source code and readme file: http://search.cpan.org/~grommel/Math-Round-0.05/

Time::Stopwatch PERL library

Source code and readme file: http://search.cpan.org/~iltzu/Time-Stopwatch-1.00/

RRDTool

Source code: http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/download.html

IPTABLES

Source code: http://www.netfilter.org

Installation

First of all you must untar the package:

- tar -xzvf APDlb-1.0.tgz
- cd APDlb-1.0
- make all
- ./install

Please note that you must install everything that was specified in the Requirements section!

It is very important to consult the readme file for every program or library that is required.

Starting and stopping the application

During the installation the user is asked where the program should be installed.

In order to start the application, run “APDlb.pl start”.

In order to stop the application, run “APDlb.pl stop”.

The script can be found in the installation path.

About load balancing

General informations

Load balancing (also known as high availability switch over) is a mechanism where the server load is distributed to different nodes within the server cluster, based on a load balancing policy. Rather than execute an application on a single server, the system executes application code on a dynamically selected server. When a client requests a service, one (or more) of the cooperating servers is chosen to execute the request. Load balancers act as single points of entry into the cluster and as traffic directors to individual web or application servers.

Two popular methods of load balancing in a cluster are:

1. DNS round robin and
2. Hardware load balancing:

- DNS round robin provides a single logical name, returning any IP address of the nodes in the cluster. This option is inexpensive, simple, and easy to set up, but it doesn’t provide any server affinity or high availability.
- In contrast, hardware load balancing solves the limitations of DNS round robin through virtual IP addressing. Here, the load balancer shows a single IP address for the cluster, which maps the addresses of each machine in the cluster. The load balancer receives each request and rewrites headers to point to other machines in the cluster. If any machine in the cluster is removed, the

changes will take effect immediately. The advantages of hardware load balancing are server affinity and high availability; the disadvantages are that it's very expensive and complex to set up.

Load balancing Algorithms

There are many different algorithms to define the load distribution policy, ranging from a simple round robin algorithm to more sophisticated algorithms used to perform the load balancing.

Some of the commonly used algorithms are:

1. Round-robin
2. Weight-based
3. Random
4. Minimum load
5. Last access time

Round-Robin Allocation

In case of a round-robin algorithm, the load balancer assigns the requests to a list of the servers on a rotating basis. The first request is allocated to a server picked randomly from the group, so that if more than one load balancer is involved, not all the first requests go to the same server. For the subsequent requests, the load balancer follows the circular order to redirect the request. Once a server is assigned a request, the server is moved to the end of the list. This keeps the servers equally assigned.

Pros: Simple to implement and the requests are equally divided among the available servers in an orderly fashion.

Cons: Round robin algorithm is not enough for load balancing based on processing overhead required and if the server specifications are not identical to each other in the server group.

Weighted Round-Robin Allocation

Weighted Round-Robin is an advanced version of the round-robin that eliminates the deficiencies of the plain round robin algorithm. In case of a weighted round-robin, one can assign a weight to each server in the group so that if one server is capable of handling twice as much load as the other, the powerful server gets a weight of 2. In such cases, the IP load balancer will assign two requests to the powerful server for each request assigned to the weaker one.

Pros: Takes care of the capacity of the servers in the group.

Cons: Does not consider the advanced load balancing requirements such as processing times for each individual request.

Random allocation

The best server is selected randomly.

It has the advantage that the traffic will not be sent to the same server

The disadvantage is that some servers will never be used

Minimum load

The criteria for selecting the best server is the minimum load. The server that is free will receive the requests.

Last access time

The server that can be accessed in a shorter time will be preferred.

In order to implement the last 2 methods we need a way to measure the criteria on the servers and a way to transmit the information back to the load balancer.

The implemented algorithm

Criteria used by the algorithm

The program uses the following criteria:

The HTTP response time for each server This criteria is obtained using a PERL library for WWW. For each server the application makes a HTTP request and measures the HTTP response time. If the server's HTTP response time is greater than 5 seconds, the server is considered unreachable.

The PERL library offers the ability to check if the response contains some HTTP errors. In this case, the server is considered unreachable.

Weights - the network delay The network delay is obtained using the PERL libraries for packet construction. A TCP SYN packet is constructed and is sent to one of the balanced servers. The next step is to wait for an TCP ACK response packet. The time it takes from sending the packet until receiving a response is the network delay.

The acceptable delay for each server - introduced by the user.

Criteria that is not used .....

The number of connection for each server could be used as one criteria, but this information can be obtained using the file ip_conntrack.

During the tests it was noticed an unexpected behavior and this criteria :) was not used.

As future work, this criteria will be taken using packet construction technique.

How does it work ?

At the beginning, the configuration tool will run and it will create a configuration file. After this the following informations are available:

The IP address of the load balancer

- Everybody will see this IP address as the IP address of “www.loadbalancedsite.com”. The load balancing process will be transparent for the users.

The IP address for each server

- This IP addresses are used when the HTTP response time or network delay are measured for each server.

Device connected to the servers

- The network interface card used to connect the load balancer and the servers. It is used when measuring the network delay.

The optimum HTTP response time for each server

ATTENTION!!!!!!!!!!

- For this criteria the beginners should use the recommended value. After changing this value, one should monitor the application in order to be sure that the desired results are obtained

Initially, all the requests are redirected to the first server. The server to which the requests are redirected is known in every moment. This server will be referred as the active server.

Every minute, the algorithm will measure the HTTP response time and the network delay for each server. So this criteria will change periodically.

The implemented algorithm :

1. Finds all servers with a HTTP response time smaller than the optimum HTTP response.
2. If there are servers with the HTTP response time smaller than the optimum HTTP response, the algorithm follows the circular order to redirect the requests, considering only the servers found at step 1 and will determine the new active server. This is similar to Round Robin algorithm.

If all the servers have a HTTP response time greater than the acceptable HTTP response:

1. For each server calculate weighted_response=HTTP_response_time*weight.
2. Calculate the minimum weighted_response.
3. The server with the minimum weighted_response will become the active server.
4. Redirect the requests to the active server using iptables.

Usage

During installation the administrator will create the configuration file using a ncurses based graphic interface. After installation, the script will run on the load balancing machine and the user will be able to see the delay of each server in the last 10 minutes, last hour, last day and the last week.

This informations are displayed in some web pages that are found in the folder “www” .

Data that need to be introduced:

1. Location of RRDTool

This program is used in order to create the graphs with the delay for every server.

2. Location of monitoring page

The program will copy the “www” directory in any place on the monitoring machine.

3. Number of servers

The number of servers between which the HTTP traffic is balanced.

4. IP address for each server

6. Optimum HTTP response time

7.The IP address of load balancer

8.The device connected to the balanced servers

At any step during the configuration process the data can accepted or rejected . In the end, all the introduced data will be printed on the screen. If a mistake was made, by pressing <n> the configuration process will restart.

If the data is accepted, a file called servers.cfg will be created, and the algorithm will run using data that is written in this file.

5.3 The performed tests

5.3.1 The first test

The topology for the first test is presented in the following figure:

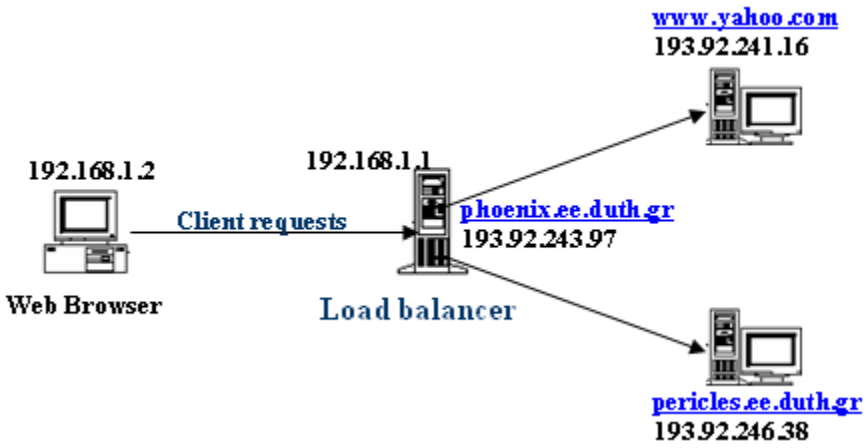

Figure 15

In this test the load balancer has two network interface cards. The clients that make the requests are in an internal private network that is connected to the load balancer. The balanced servers are “www.yahoo.com” which is somewhere in the Internet and the second balanced server is “pericles.ee.duth.gr”. The second server is located in the same local network with the load balancer, so there is a better connection with this server. The load balancer will balance the traffic coming from the internal network (having the source IP address in the network 192.168.1.0/24) to 193.92.243.97, which is known in the Internet as “phoenix.ee.duth.gr”.

The clients have as default gateway the IP address of the load balancer’s network interface card connected to the internal network. The load balancer will perform SNAT for all the packets coming from clients because they are in the private network. When a client will type “phoenix.ee.duth.gr” in the address line of his web browser, he will first make a DNS request to the DNS server asking “Who is phoenix.ee.duth.gr ? ”. The DNS server will respond “phoenix.ee.duth.gr has the IP address 193.92.243.97”. The client will notice that the destination IP address is not in the same network with him and will sent the packet to the default gateway (the load balancer). The load balancer will see a HTTP request for its external IP address and before routing the packet, it will replace the destination IP address of the packet (193.92.243.97) with the IP address of one of the balanced servers. The packet will be routed and the load balancer will decide to sent it on interface eth0 (dev eth0). After routing, the load balancer will also replace the source IP address from the request with its external IP address which is public. Unless SNAT is performed, the first Internet Service Provider will reject the packets. After the packets arrive to the servers, they will see a HTTP request from the load balancer and they will send back a HTTP response. The load balancer will receive the response and will replace back the destination IP address with the one of the client and the source IP address with its internal IP. The client will see an HTTP response coming from the load balancer. The whole process in transparent for the clients.

During certain moments of the day (in the morning or in the evening) the HTTP response time for the balanced servers were comparable. This might sound a bit strange when considering the fact that one server is in a remote location and the connection between the load balancer and the server is worth than load balancer’s connection with the second server. The explanation is that the network delay is in the range of milliseconds. On the other side, the HTTP response time is in the range of

seconds. The HTTP response time is composed of the network delay and the time it takes the server to process the request. When a client will make a HTTP requests for “http:// 193.92.243.97” he will receive one minute www.yahoo.com page and in the next minute pericles.ee.duth.gr.

In the middle of the day, when people are using more www.yahoo.com the HTTP response time for www.yahoo.com is greater than the optimum delay and all the HTTP requests coming from clients for “http://193.92.243.97” were served by pericles.ee.duth.gr.

5.3.2 The second test

In the following figure we have the topology for the second test:

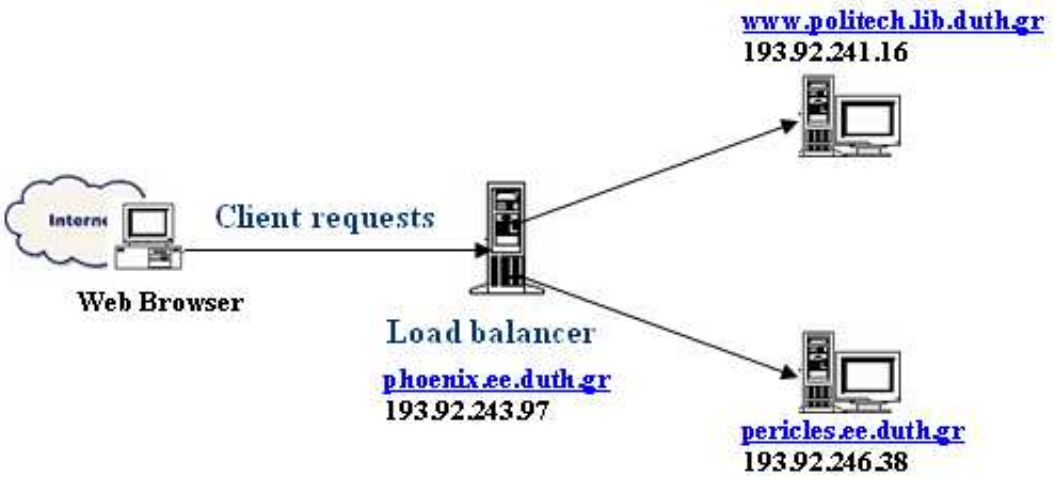

Figure 16

In here the load balancer is “phoenix.ee.duth.gr”, “pericles.ee.duth.gr” is kept as one of the balanced servers and another server, “www.politech.lib.duth.gr”, is added. This server is in the same local area network with the load balancer. So, the connections between the load balancer and the two servers are comparable. The clients can be in any place in the Internet. In this test, both balanced servers have another connection to the Internet. They do not have as default gateway the IP address of the load balancer.

When the clients make a request for “phoenix.ee.duth.gr” they will use 193.92.243.97 as destination IP address and their own public IP address as source. When the requests come to the load balancer, the load balancer will replace the destination IP address with the IP of one balanced server and will route the packet. The packet will finally arrive to the web server. The web server will see a HTTP request from the client and will send back a response. Because the client and the web server are not in the same IP network, the web server will send the request to its default gateway (which is not the load balancer), and from the default gateway the packet will be routed to the client. The client will see a HTTP response from the web server and not from the load balancer, to whom he made the request, and will drop the packet.

In order to solve this problem the response packets must be forced to pass through the load balancer. One solution is to set as the default gateway for the balanced servers the IP address of the load balancer, but there might be situations when this is not desired, for example if the balanced servers

use different connections to the Internet. Another solution is to replace the source address for all the HTTP packets coming to the load balancer with its IP address. This way, when the web servers will see the HTTP request they will think it came from the load balancer and will send the response to this machine.

One common mistake is to specify the destination IP address in the rule when replacing the source address for all HTTP packets (port 80) . This will not work because when the HTTP packet arrives to the load balancer they will be checked against the PREROUTING chain. In here, the destination IP address will be replaced. The packet is routed and next it is checked against the POSTROUTING chain. In here the source address must be replaced in order to force the response packets from the servers to pass through the load balancer. If the IP address of the load balancer is specified as destination match for SNATing the packets, no packets will match anymore because the destination address was already replaced in the PREROUTING chain. As a result, the web servers will see the packets coming from the client, not from the load balancer.

All the packet manipulation is made using iptables, in the NAT table.

The servers have similar response time because they are used by comparable number of clients. If more requests are made to one of the servers a greater response time is noticed and the second server will be used for serving the clients.

The colors are:

red - 193.92.241.16

blue - 193.92.246.38

Here are the graphs that were taken when more requests were sent to one server or another.

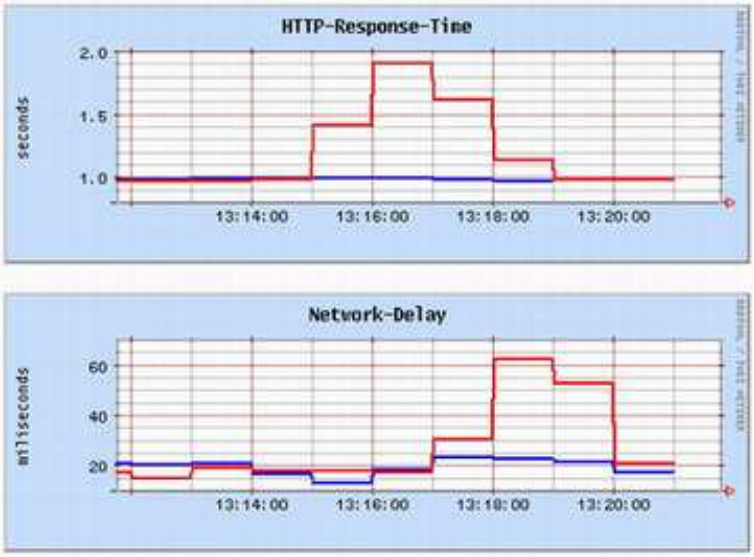

Figure 17

In this moment the server “www.politech.duth.gr” experiences a big number of requests. There is an increase in HTTP response time comparing with “pericles.ee.duth.gr”. The deference between network delays is in the range of milliseconds and the HTTP response time depends mostly on the processing power of the web server.

This is the iptables output for this moment:

Chain PREROUTING (policy ACCEPT)

target prot opt source destination

DNAT tcp – anywhere phoenix.ee.duth.gr tcp dpt:http to:193.92.246.38:80

Chain POSTROUTING (policy ACCEPT)

target prot opt source destination

SNAT tcp – anywhere anywhere tcp dpt:http to:193.92.243.97

Chain OUTPUT (policy ACCEPT)

target prot opt source destination

As indicated in the DNAT chain, all the HTTP requests to the load balancer “phoenix.ee.duth.gr” are redirected to 193.92.246.38 “pericles.ee.duth.gr”.

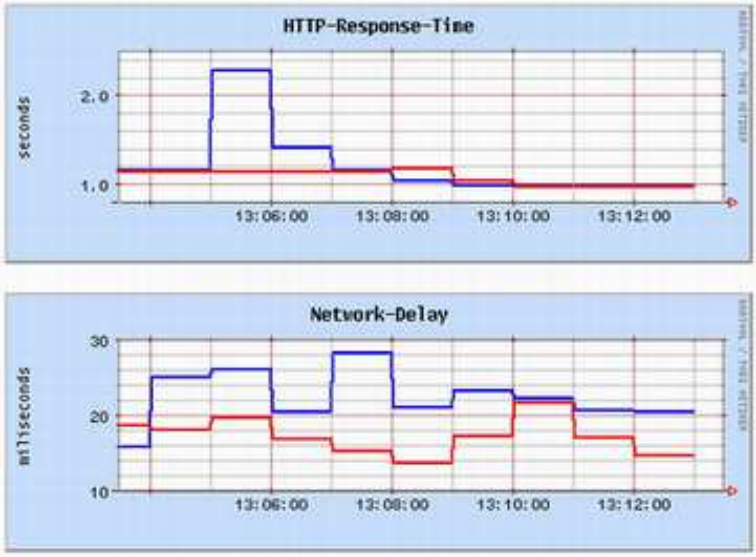

Figure 18

In this figure the server “pericles.ee.duth.gr” received a greater number of requests. This is reflected in the difference of HTTP response time between the two servers. The output of iptables presented below shows that all the TCP requests from any source to “phoenix.ee.duth.gr” will be redirected to 193.92.241.16 “www.politech.duth.gr”.

Chain PREROUTING (policy ACCEPT)

target prot opt source destination

DNAT tcp – anywhere phoenix.ee.duth.gr tcp dpt:http to:193.92.241.16:80

Chain POSTROUTING (policy ACCEPT)

target prot opt source destination

SNAT tcp – anywhere anywhere tcp dpt:http to:193.92.243.97

Chain OUTPUT (policy ACCEPT)

target prot opt source destination

5.4 Summary

In this chapter was presented the way one should use the application. The performed tests and the problems encountered during testing were also presented. On the other hand the testing topologies prove that the application is flexible and can be used in different situations and environments.

Chapter 6

Conclusions

6.1 Summary of the project

The end for this project was to create an application that will implement a simple algorithm for balancing the HTTP traffic between a cluster of web servers located in the same LAN.

It started with the documentation during which the project was designed. It was in this stage when the implementation method was chosen. The steps that were decided:

1. gathering the criteria for load balancing using PERL libraries
2. creating a load balancing algorithm
3. implementation of the algorithm using PERL
4. testing
5. creating the web pages for monitoring the criteria
6. creating a graphic configurator for inserting the input data using ncurses
7. creating the installation package.

After deciding the main steps of the application, the documentation stage was split in several parts:

- documentation about web and HTTP communication model
- documentation about load balancing and existing implementations
- documentation about networking protocols and technologies: TCP, IP, Ethernet
- documentation about packet filters
- documentation about PERL scripting
- documentation about ncurses C library
- documentation about RRDTool.

Once the documentation part ended, the design of the application started. The challenge was to find efficient ways to gather the criteria for load balancing and to combine this criteria in the most efficient way.

The next step was to implement PERL scripts that gather the criteria. During the implementation, the stability of the system was also tested.

After finding ways to gather the criteria the algorithm was implemented and tested. The algorithm was adjusted according to the test results.

Once the algorithm was stable, the graphs were created using RRDTool. This stage was implemented after studding RRDTool behavior.

The final step was to implement the graphic configurator using ncurses C library.

A time was allocated to writing this documentation which is intended to be not only a presentation of the software we created, but a guide for those who use the application, especially for those who want to participate in future development.

6.2 Achieved goals

The development of the application involved dealing with different areas of computer science, from basic networking aspects to issues related to Internet design and HTTP communication, UNIX operating systems, integrating existing tools with scripting languages, and C programming. The development was not easy and the application had to be developed in a short time.

In this moment the result is a stable, efficient and scalable application that will help network administrators to increase their web server responsiveness. One of the most important aspects, and one that makes the difference between APD Load Balancer and other existing applications is the fact that the users don't need to use expensive equipments. It is intended to offer a inexpensive solution for load balancing the web traffic to small companies, organization or universities.

The application will be published as an Open Source project with the hope that there will be interest in it and people will share their ideas about other possible solutions to the load balancing policy.

As a final product, the software achieved the targets that were set at the beginning and will offer a start for new versions that will cover aspects related not only to load balancing policies but will involve quality of service aspects and step by step, as many people will share their ideas, the application will be more and more flexible and will meet the needs of a lager category of users.

6.3 Future plans

In the future a new version of the application will be developed. At that moment some libraries, that now are experimental in PERL, will become stable and will give us more freedom in implementing scripts that will be able to gather more criteria. Among the things that will be implemented we mention: initial server condition, introducing the number of connection to each balanced server

as a criteria, changing the static value of HTTP optimum response using simulated annealing and configuration parser.

Bibliography

[1] Server Load Balancing by Tony Bourke

[2] Load Balancing Servers,Firewalls, *Caches by Chandra Kopparapu

[3] Programming Perl, 3rd Edition by Larry Wall,Tom Christiansen, Jon Orwant

[4] Iptables tutorial by Oscar Andreasson

[5] NCURSES Programming HOWTO by Pradeep Padala