Design and implementation of an MPLS based load balancing architecture for Web switching

Radu Dragos, Sanda Dragos & Martin Collier
School of Electronic Engineering - DCU
Switching and Systems Laboratory
Dublin City University
Glasnevin, Dublin 9, Ireland
e_mail: {dragosr, dragoss, collierm}@eeng.dcu.ie
Phone: +353 1 700 5854. Fax +353 1 700 5508

Abstract

The WWW has been the preferred technology used to provide information and e-services over the Internet. However, one major limitation of Web servers is that beyond a threshold, the server will not be able to process requests in a timely manner or will reject requests.

Addressing this problem requires new architectures such as Web clusters, and new technologies such as Web switching. The current available solutions do not offer end-to-end quality-of-service (QoS) or are cost prohibitive.

MultiProtocol Label Switching (MPLS) is the industry-standard approach to switching and forwarding for next-generation routed networks. The MPLS technology combines the simplified forwarding characteristics of link-layer switching with the control and scalability of network-layer routing.

In the context of a QoS enabled network with MPLS capabilities we propose an MPLS based solution to Web servers load balancing. We have designed and implemented an open source and cost-effective Web switching architecture for next-generation IP networks.

Moreover, our results prove the validity of the approach and verifies the performance of the MPLS based Web switch in stress conditions.

Keywords: Web switching, load-balancing, MPLS, server cluster, Linux

1 Introduction

The number of services offered over the Internet has dramatically increased in the last few years. The WWW service no longer consists in merely displaying the static content from a web server to a web browser. New and more traffic- and CPU-intensive services are widely used nowadays. Services like search engines, file servers, multimedia streaming, application servers, database servers and e-commerce servers, not only require higher bandwidth but also stretch to the limit the computation performance of servers.

Popular Web sites receive ten times more requests than three years ago, from 115 million page views per day in June 1998 to 1.5 billion page views per day in September 2001 (e.g., [1]). The value is around 17361 hits per second and may increase over the next few years. The rate of incoming requests is more than a single server will be able to answer in a timely manner. Consequently, Web service providers face technical challenges in meeting this burgeoning demand and must deploy new techniques in order to satisfy the continuously increasing number of requests.

This paper addresses the issue of increasing Web server availability and reliability. Sections 2 and 3 describe the background and related commercial solutions. In section 4 we propose a an open source, cost effective architecture and we prove the functionality by implementing the solution.

2 Background

This section describes issues related to solving the problem of over-congested Web servers.

2.1 Web content caching

One of the early approaches was the caching of the web content at the client side, initially on the client local machine (the cache maintained by the Web browsers) then at the corporation level by using proxy servers [2, 3, 4]. Caching mechanisms will deliver the local stored data, if data was previously requested by another client or a previous connection and if the content is up-to-date instead of the content requested from the remote server.

The caching solution was only a temporary attempt to reduce the number of requests by reducing the redundancies in the data transferred over the Internet. This only works with static web content. With the introduction of new services, a new type of information was processed at a Web site: dynamic data, which involves information construction at the server site before answering the request. This kind of information can not be cached; therefore, the use of caching will not reduce the server workload.

2.2 Mirroring

The second approach known as mirroring consists in maintaining a number of servers with similar content, but with a different geographic location and a different name and Internet address (e.g. [5]). The client has to choose, among the geographically distributed servers, the one that best suits to his requests. This approach leaves to the client the decision of choosing the mirror and the responsibility of choosing the right one or the wrong one. Many times the client will initiate more than one request to different servers, in order to determine the ”closest” mirror. On the other hand, maintaining a perfect synchronization between the mirrored servers may not be an easy job, especially for time critical applications. Moreover, the situation when the clients are not geographically distributed but concentrated within a single geographic area (or even the same WAN or LAN) can not be solved by spreading the servers around the area. The time spent by the client deciding the most suitable server may be too long for mission-critical applications.

2.3 Cluster of servers (server farms)

The next approach tries to avoid the user involvement in the process of choosing the best server. The technology has to transparently deviate the client's request to the optimal server. The technique consists in grouping the servers in so called server clusters and adding a new and transparent service to the network, which is responsible for distributing the requests uniformly among the servers [6].

Successful administration of server clusters or server farms requires the use of specialized mathematical and engineering theories like queuing theory and load balancing techniques. The most important goal from the Web service provider's point of view is to uniformly balance the workload among the servers within the cluster.

The two major methods of building Web clusters are described as follows.

2.3.1 Replicated content servers

The first method is to mirror the content of a Web server and to create two or more servers having identical content. This resembles the geographical distribution of mirrored servers, but has the advantage that the servers are grouped within the same building or room and under the same administration; thus the task of synchronizing the Web content between mirrors is much easier. Moreover, the choice of server is no longer the responsibility of the client.

2.3.2 Distributed content

The second method is to distribute the Web content among the servers within the farm. Therefore, the decision of choosing the server is based on the client's HTTP request and involves filtering the packets up to the application level. Thus, the problem is not anymore a problem of uniformly distributing the requests but of a priori distributing the content within the servers in a manner that will result in a balanced workload among the servers.

A cluster of servers is the typical solution to the problem of increasing Web server availability. In section 3 we describe some approaches to cluster implementation.

3 Related problems and solutions

This section describes the major approaches used to emulate a Web server using a cluster of servers in response to the problem mentioned above.

3.1 Problems

3.1.1 Overloading a Web server

Web server is considered overloaded when the number of incoming requests exceeds the server's capacity. The capacity of a Web server is marked by a soft or hard threshold.

A soft threshold is a limit in the number of simultaneous accepted requests. Beyond this limit the server will not be able to process the requests in a timely manner.

A hard threshold is the maximum number of simultaneous connections (e.g. 150 clients [7]. If this limit is ever reached, subsequent clients will be rejected. In e-business, an overloaded server is a critical problem for companies providing Web based services since they can lose clients and revenue. Therefore, the Web server has to be always available and reliable.

An overloaded server can be avoided using a Web farm, provided that the peak demand is known, thereby allowing the minimum number of servers required to be estimated.

Let's consider the situation when packets arrive at the server with an arrival rate uniformly distributed over the interval [0,20]; that is an average rate of $\lambda = 10$ connections/second. We consider connections to have randomly lengths, uniformly distributed over the interval [0,60] seconds. This is an average of $l = 30$ seconds. After a time t approximately equal to l the number of active connections will vary around the average value $c_1 = \lambda * l = 300$ connections as simulated in Figure 1.

Standardly configured Apache [8] Web server accepts a maximum of $max = 150$ simultaneous connections. The above situation exceeds the maximum server capacity and consequently not all the requests will be processed. More than one server is needed to deal with such a large number of connections.

In an ideal situation the average number of connections per server using n load-balanced servers, is $c_n = \lambda * l/n = 300/n$ and, two servers seem to be enough since $max = 150 = 300/2 = c_2$ connections/server. But in the real

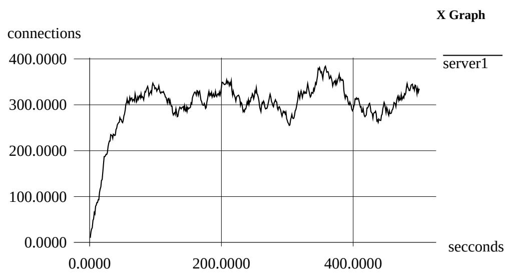

Figure 1: The number of active connection for 1 server

world, λ and l vary in time and c_n will take values grater than max (Figure 2).

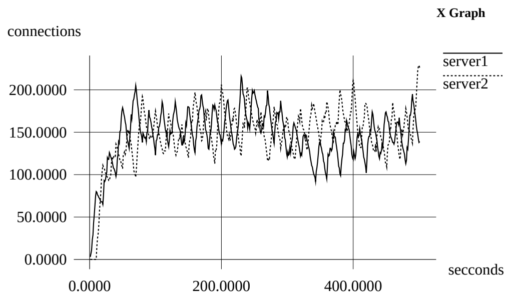

Figure 2: The number of active connection for 2 servers

Using the same simulation, we can see that acceptable results are obtained for 3 servers, $n = 3$ and we obtain an average load per server of $c_3 = \lambda * l/n = 100$ connections/server (Figure 3).

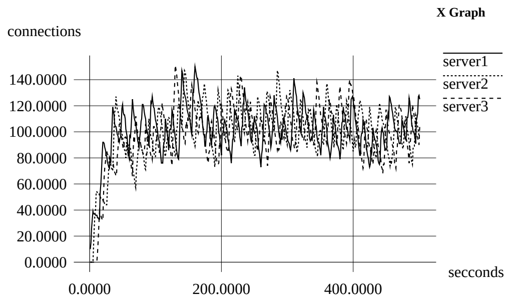

Figure 3: The number of active connection for 3 servers

In conclusion, more than one server is required for a high number of simultaneous requests. The number of servers can be estimated if the arrival rate λ and the average connection length l can be predicted.

3.1.2 The TCP continuity problem

Another major issue with HTTP traffic is that it uses the TCP protocol to establish and maintain the connection between the Web browser and Web server. Therefore, we deal with a connection-oriented protocol. This causes a major problem for load balancing techniques. Imagine the situation when the first request from a certain client is sent to the optimal server from the cluster. The connection will be established between the peers and then during the connection, the load balancing algorithm will choose another optimal server and send the remaining packets of the TCP session to the second one. This will result in breaking the connection and interrupting the flow.

The *TCP continuity* problem must be avoided and consequently the load balancing technology has to implement a mechanism for maintaining the TCP connections alive. Generally, this is done by applying the algorithm only for the first packet of the connection (marked with the SYN TCP flag). Then, all the subsequently packets of the session, will be sent to the same initial server elected at the arrival of the connection initiator packet.

The approaches that face this problem have to maintain state information for the active TCP connection in order to avoid the breakouts of the long HTTP transfers and inconsistency of the e-commerce transactions. The available solutions require the examination of the TCP or HTTP headers. Information such as the socket port, TCP flags, SSL session timeout or cookies can be used to identify the packets belonging to the same session and maintain the session uninterrupted [9].

3.2 Approaches

3.2.1 Round-robin Domain Name Service

The first method used to avoid the server congestion by distributing connection loads over a cluster was based on the Domain Name Service (DNS) [10]. In a standard scenario, a domain name is associated with an IP address. Since the client uses the domain name to access a Web site, a DNS has to translate the name into the correct IP of the destination server. Therefore, the DNS server is a part of the Web browsing process. Moreover, the server can be modified to answer with different IP addresses for different translation queries. The DNS server will rotate through the list of the IP addresses in a round robin fashion in such a way that each server in the cluster will receive a part of the incoming requests.

The main advantage of round-robin DNS is its simplicity. No additional hardware or software is required and since the client will initiate the DNS query only once per TCP session, we will not encounter the *TCP continuity problem*. On the other hand, there are also major drawbacks for this approach. The caching feature of DNS at the client side, prevents an accurate load balancing scheme since not every incoming request will get its address directly from the round-robin DNS server. Disabling caching may appear to solve the problem. However, every DNS query must then be resolved by our server; this is expensive and potentially slower for users. Moreover, a client may use the IP address of the Web server thereby bypassing the DNS server and all its requests will be sent to the same server.

The other major disadvantage of this approach is that the DNS server does not have any knowledge about the status of each server in the cluster. The round-robin scheme will continue to send traffic to all servers in turn, even if some of them are overloaded or out of service.

3.2.2 Load balancing switches

Load balancing switches, such as Cisco's LocalDirector [11] and Alteon's ACEdirector [12], are hardware solutions that distribute TCP connections over multiple servers. These Web switches act as a front-end dispatcher between the Internet connection and the Web farm. All the client requests will use the dispatcher IP as a destination address, to make the requests. The switch then forwards the requests to different Web servers based on various load-balancing algorithms implemented in the switch. The decision can be based on the content of the request. Using source IP address alone to create affinities between client and server will not work well since some companies use proxy servers that change the source IP of the request. Therefore, all the requests from behind the proxy will have the same IP thus the whole network behind the proxy will be treated as a single computer.

Load-balancing Web switches support up to millions of connections simultaneously at high speeds. Moreover, switches will frequently check the status of the servers so they can be relatively intelligent about load balancing. Using a Web switch is much better and more scalable than using other approaches but they are quite expensive. In addition, avoiding a single point of failure, may require the use of multiple switches which makes the solution uneconomic.

3.2.3 HTTP redirect

HTTP redirect can be used by the targeted server if it can not accept more connections. [13]. This technique will slow down the process since the request is sent back to the client along with another IP to use for the connection. The client will have to initiate another connection to the new IP and use the server to which was redirected.

The above mentioned solutions still have major drawbacks, nevertheless they are in use in today's best effort Internet. However, the Internet is evolving into a next generation QoS enabled global network and new standards and protocols are now available (e.g MPLS). Therefore, we propose an alternative, for the next generation MPLS capable networks.

4 An MPLS Approach

4.1 MPLS & TE

According to the Internet Engineering Task Force(IETF), traffic engineering is defined as that aspect of Internet network engineering dealing with the issue of performance evaluation and performance optimization of operational IP networks [14].
The objectives of traffic engineering in the MPLS environment are related to two performance functions [15]:

- **traffic oriented** performance which includes QoS operations.
- **resource oriented** performance objectives which deal with networking resources to contribute to the realization of **traffic oriented** objectives.

The both objectives of traffic engineering are related to load-balancing for Web servers. As a resource oriented problem, server over-utilization (congestion) occur if a web server is overloaded by a high number of requests. If a cluster is used to compensate the congestion, using an incorrect load-balancing algorithm will result in servers under-utilization or unbalanced utilization across the cluster. Then, for a working Web switching architecture, the traffic oriented issue of supporting QoS operation, has to be addressed.

MPLS plays an important role in engineering the network to provide efficient services to its customers.

MPLS is a "multiprotocol" which uses label switching technology. Label switching paradigm consists in using a short, fixed-length label to perform switching decisions. Unlike *longest prefix match* lookup algorithms used by standard IP routing protocols, label switching is based on an exact match and therefore is much faster.

The routers supporting MPLS are referred to as Label Switching Routers (LSRs) (Figure 4). Any other router or switch connected to a LSR (ATM switch, IP router) is referred to as non-LSR. An edge router is an LSR connected to a non-LSR. The router by which a packet enters the MPLS cloud is called the ingress LSR, and the one by which it leaves the MPLS cloud is called the egress LSR. Label Switching Path (LSP) is the route within the cloud, followed by a packet based on his label.

Labels are small, fixed identifiers inserted by the ingress LER, removed by the Egress LER and used in forwarding decisions at each LSR the packet traverses. Packets marked with the same label belong to the same Forwarding Equivalence Class (FEC) and are routed and treated the same way in the network.

RFC 2702 specifies the requirements of traffic engineering over MPLS and describes the basic concepts of MPLS traffic engineering like traffic trunks, traffic flows and LSPs [16]. The advantages of MPLS for traffic engineering include:

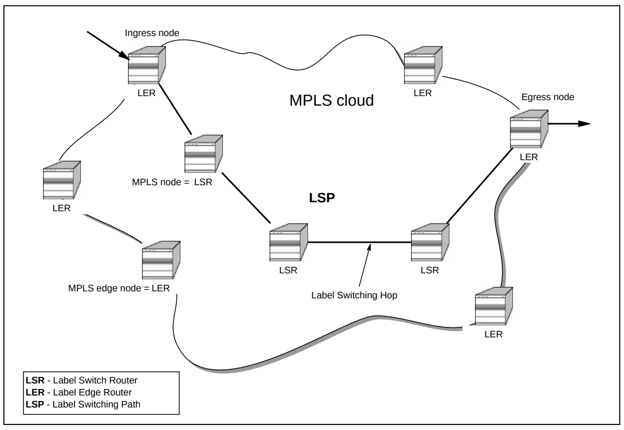

Figure 4: Elements of an MPLS cloud

- label switches are not limited to conventional IP forwarding by conventional IP-based routing protocols;
- traffic trunks can be mapped onto label switched paths;
- attributes can be associated with traffic trunks;
- MPLS permits address aggregation and disaggregation (IP forwarding permits only aggregation);
- constraint-based routing is easy to implement;
- MPLS hardware is less expensive than ATM hardware.

In sections 4.2 and 5, we propose an MPLS based architecture for Web servers load balancing and we describe our implementation.

4.2 Designing an MPLS load balancing architecture for server clusters.

The Internet is a connectionless network. Nevertheless, the WWW architecture, uses the HTTP application layer protocol to deliver information. Moreover, HTTP is based on TCP layer 4 protocol which is a connection-oriented protocol. Meanwhile, MPLS is a connection-oriented protocol. Therefore, a natural approach to load balance the HTTP flows is to map them into MPLS LSP's. The idea is to use different labels to specify the flows for each server across the cluster. This subsection depicts the framework for an MPLS based approach to Web switching.

The first proposal for the use of MPLS for Web routing was presented in a IBM research report [9]. Since MPLS provides better mechanisms to support QoS routing than the legacy IP [16, 17],it can more elegantly support such functions as:

- **content-based routing**
- **client affinity**
- **different classes of service**
- **load balancing**

The technique proposed in [9] requires that the dispatcher will maintain a table of associations between labels and the associated server weight. The dispatcher will then send a tuple $< \{L_1, W_1\}, \{L_2, W_2\}...\{Ln, Wn\} >$ to a proxy server situated in front of MPLS ingress nodes using a dedicated signaling protocol.

In our approach, we want to reduce the load of the dispatcher and the need for a dedicated signaling protocol. We also reduce the complexity of the solution by eliminating the proxy nodes used by [9] at the client side. Moreover, we want to preserve the functionality of an MPLS powered Web switching technique. We also implemented the framework using cost-effective PC based MPLS switches and run performance tests as we emphasize in section 5

Content-based-routing is a technique used when the content of the Web site is partitioned across the server cluster. In this case an MPLS Forwarding Equivalence Class (FEC) is constituted by all the requests for the same server. Having a FEC for each server along the farm, at the ingress nodes, packets can then be easily bound to the corresponding FEC. This solution has two major advantages. It will reduce the load at the dispatcher since the decisions are taken at ingress nodes. Moreover, we may eliminate the single point of failure at the dispatcher since LSP's can follow different routes toward their destinations within the cloud.

Client affinity may be used in the situation when clients have preferences for a certain server. The solution also requires establishing virtual connections between clients and server in a multiple to one fashion (m:1). This is yet another strong advantage of using a label switching technology and building FECs based on client's source IP. The packets can then be switched to their final destination using MPLS fast switching hardware.

The ISP may wish to provide **different classes of service** to clients, based on service level agreements or other administrative factors. The MPLS approach can provide different FECs for different classes of service. Packets can be labeled corresponding to the type of the class (gold, silver or bronze). If servers have different processing performances, the gold-labeled packets can then be switched to the best performing server. Label stacking also can be used to define and identify hierarchical classes of service.

The **load balancing** function performed using MPLS is the key element of this paper and will be described in more detail in section 5.

This approach presumes that the ISP providing the Web service already uses an MPLS enabled network. All the ISP's devices are MPLS capable. The clients for the Web service do not have to implement MPLS since the ingress of the ISP's administrative domain will be the ingress of an autonomous MPLS domain as well. The solution involves the use of a front-end dispatcher and a Web server farm as in Figure 5.

The main problem of using MPLS is that we are not able to access layer 4 or layer 7 (TCP, HTTP) headers at the

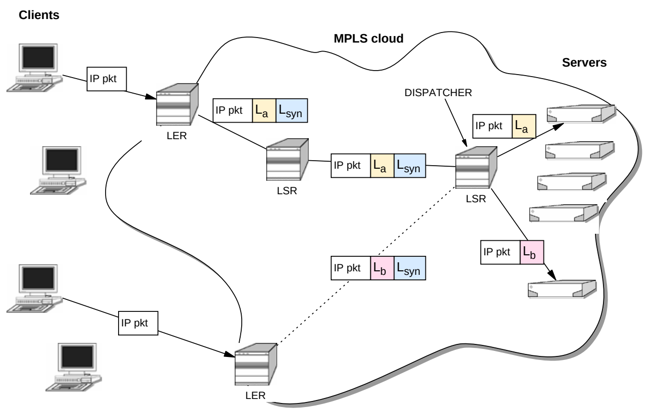

Figure 5: A framework for MPLS Web switching

dispatcher. This is because MPLS is a fast switching technology used at a lower level (between layer 2 and layer 3) and accessing higher level headers can dramatically slow it down. The access to the TCP or HTTP headers has to be done at the ingress nodes.

As we previously mentioned, we want to reduce the load of the dispatcher and the need for a dedicated signaling protocol. In order to accomplish this, we use dedicated labels to mark the beginning of a TCP connection. A layer 4 filter, placed at the ingress nodes, will classify the SYN packets (used to initiate the TCP session) and label the packets with a dedicated label (L_{SYN}) marking the beginning of a new session. The dispatcher will only be responsible for deciding which is the most lightly loaded server, then switch the incoming L_{SYN} label with the label associated for that server and forward the packet to the server. The optimal server can be decided based on the processor load, the number of active connections, or the traffic generated through its network interface or even a in round-robin fashion.

Once the packet reaches its destination, the MPLS label will be removed and the packet will be treated as a standard HTTP request. The server will generate the usual reply, label the packet with its personal label (L_{S1}) and send it back to the dispatcher.

The packets originated from the server will be relabeled at the dispatcher using an MPLS label stack. Another label will be pushed on top of the stack and used to switch the packet along the MPLS cloud, back to the ingress node. The label added initially by the server (L_{S1}) will remain in the stack and it will be used later to identify the server.

At the edge router, the top label is removed and the second label (L_{S1}) is used to maintain a table of active sessions for that server. The table is mandatory in order to keep the TCP sessions alive by forwarding all the subsequent packets of the session to the same server.

As we already anticipated, for the remaining packets of the connection we will use the table at the edge routers to preserve the destination server. In order to do that, the edge router will use again a two layer stack to label the packets. First, the label associated with the current connection and its corresponding server is pushed accordingly to the associations in the table. Another label is then pushed on top of the stack and used by the next hop to forward the packet to the dispatcher.

Here, the top label is removed, and the second label is used to switch the packet to its server. The server receives the packet, removes the label, and then processes the request. The cycle is completed and the HTTP connections remains uninterrupted during the TCP session.

The main advantage of this approach is that the edge routers will associate the requests to servers in a distributed manner. Consequently, the dispatcher will perform as an ordinary MPLS switch with an added load-balancing function. However, all it has to do is to apply the function for the first packet of each connection. The rest of the packets will arrive already classified and will be switched to their destination. Nevertheless, the connection tracking process is now distributed along the edge routers and not centralized in a single box.

5 Implementing the technology

5.1 Implementation

We chose Linux as a platform for implementing the MPLS based Web switching architecture. Linux is a free, open-source, POSIX compliant, UNIX clone operating system. Its true preemptive multitasking, multi-user support, memory protection and symmetric multiprocessing support characteristics together with its networking, graphical user interface, speed and stability make Linux a preferred tool for research and development. However, even if the platform is open source, we did not modify the operating system internals. This is because we want to prove that the architecture is operable using standard networking tools. Figure 6 depicts the software used to implement our technology.

The architecture described in section 4.2 has been implemented and evaluated on a soft router. Details of the implementation are presented below.

Operating System (OS)

As we already mentioned, we leveraged the use of UNIX-like operating systems at the edge of the Internet performing as routers, soft switches, firewalls or Web servers. We used the free Linux distribution from RedHat[redhat] as a platform for all our devices. The only add-ons to the standard distribution were:

- adding MPLS support to the Linux kernel
- adding MPLS support to the Linux standard firewall

Web server

Apache is the standard Web server shipped along with the OS. We kept the standard configuration of the HTTP server. The OS running Apache constituted the target for

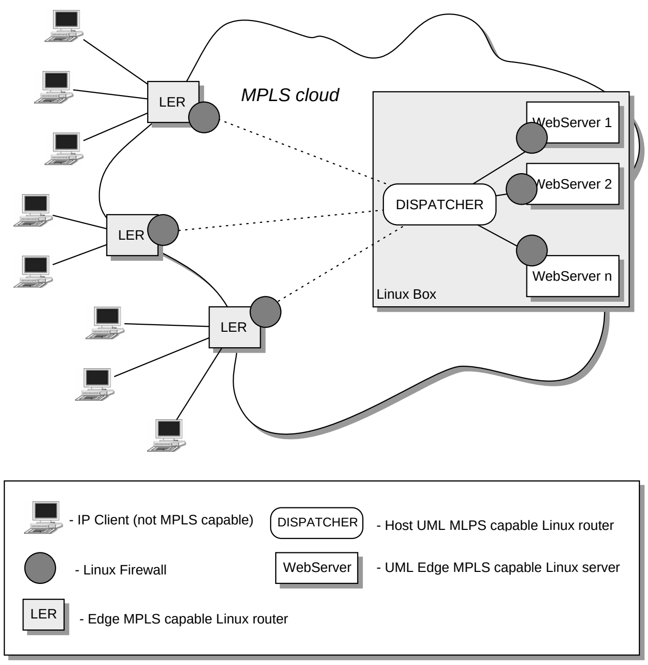
MPLS cloud
LER
LER
LER
DISPATCHER
WebServer 1
WebServer 2
WebServer n
Linux Box
- IP Client (not MPLS capable)
DISPATCHER - Host UML MLPS capable Linux router
- Linux Firewall
WebServer - UML Edge MPLS capable Linux server
LER - Edge MPLS capable Linux router

Figure 6: Elements of the MPLS based Web switching implementation

our load balancing scheme.

Mirroring the Web content
User Mode Linux (UML) is a simple and secure way to run and test multiple Linux kernels on a single PC. We use it to run multiple identical Linux Web servers using a single PC-based computer.

MPLS
MPLS for Linux is a open source effort to create a set of MPLS signaling protocols and an MPLS forwarding plane for the Linux OS[mpls-linux]. The current version (mpls-linux_1.128) is not yet a stable version and does not offer the high performance of hardware based MPLS switches but allowed us to test and develop MPLS in an open source environment.

Connection Tracking
Netfilter is a firewall, Network Address Translator (NAT) and packet mangling tool for Linux, [18] and is a standard application for RedHat Linux. The only modification to the standard distribution was the support for MPLS filtering.

The dispatcher
We used the host for the UML Web servers as a dispatcher also running MPLS. A script or C program was used to decide the optimal server based on a round-robin algorithm or a load balancing function.

The main issues of the implementation where the load-balancing function at the dispatcher and maintaining the active sessions table at the edge routers.

- **Load-balancing function** was implemented at the dispatcher side using *C* code and/or *bash* scripts. For a round-robin algorithm, a script was used to associate the FEC of the incoming requests to the LSP for the optimal server. For more complex algorithms, *C* programs are used to retrieve information about the load of each server in the cluster. Using Simple Network Management Protocol (SNMP) we can gather informations like CPU usage, bandwidth usage or the number of active connections and use these parameters to decide the best server for the incoming requests. If the traffic is predictable, we can use static algorithms (e.g. every 10 seconds) to elect the less loaded server. If fluctuations in the number and type of requests make the traffic unpredictable, we can use adaptive algorithms to dynamically deviate the traffic to the optimal server.

- **Active sessions table** is maintained at the edge routers and used to solve the *TCP continuity problem*. In our implementation we used the queuing to user-space capabilities of the Linux firewall [18]. We use *C* programs to filter the SYN, ACK responses from the web server. We then used the label a the bottom of the MPLS stack to identify the server and maintain the table of the active TCP sessions.

In our implementation, we benefited from another advantage of working with a label switching protocol. Using MPLS we can send different packets destined to a single IP to different Web servers since the IP is tunneled within the MPLS. The cluster is seen as a single IP (say IP_cluster) from the client side but the packets can be transparently distributed across the servers. Another problem occurs at the server side since the HTTP server has to accept an IP packet destined to another IP (IP_cluster). Moreover, it will have to reply with the IP_cluster instead of its own IP. The problem has an easy solution under Linux by using the netfilter's NAT module to change the destination address for incoming packets and source address for outgoing packets.

5.2 Performance Evaluation

The performance was evaluated empirically in a live test. We use an MPLS unaware machine to generate HTTP requests for the cluster. We use a primitive round-robin load balancing scheme to verify the protocol used to redirect HTTP Traffic. A two server Web farm was sufficient to test our implementation. The client requests a large file (with a download time greater than 3 seconds) every 3 seconds. The dispatcher rotates through the server list every 2 seconds. The files were downloaded from the server according to the scheme depicted in Figure 7:

Figure 7 shows in primitive form the behavior of the requests at the cluster side. For a more complex scheme, we

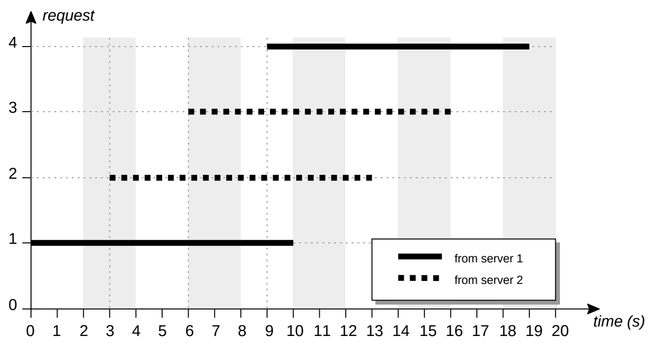

Figure 7: Distributed requests

considered 3 servers and we generated more concurrent requests.

The main concern was for multimedia and/or file servers where we deal with long connections and big files being requested simultaneously. Therefore, for the first performance test we considered relatively large files (4.2MB). We also presumed an average arrival rate $\lambda = 0.4$ connections/second; that is almost 35000 requests per day. We than ran 3 tests generating respectively 20, 30 and 50 connections. We used a round-robin algorithm to distribute the requests along the cluster. Table 1 presents server loads measured in the number of connection processed and the percentage of the total number of connections. The results show that servers share the workload in a balanced fashion varying around the value of 33.33%. The load percentage never increased over the value of 40% per server.

	Server 1		Server 2		Server3	
	no.	%	no.	%	no.	%
20 con	7	35%	5	25%	8	40%
30 con	11	36.66%	12	40%	7	23.33%
50 con	16	32%	16	32%	18	36%

Table 1: Round-robin load balancing for large files

The second test suite was intended to study the behaviour of the load-balanced cluster for a higher number of requests but for smaller files. We used files with sizes uniformly distributed over the interval [100KB,1024KB] and we varied the arrival rate λ from $\lambda = 3$ to $\lambda = 12$ connections/second, which means over 1 million hits per day. We initiated respectively 300 requests at $\lambda = 3$, 900 connections at $\lambda = 9$ and 300 connections $\lambda = 12$ connections/second. The results are shown in Table 2 and reveals even better percentages for shorter connection at a high arrival rate than for the previous test with longer connections..

Both tables proves that our architecture provide good results even if we used a statical load-balancing algorithm such as round-robin. This anticipated that superior performance will result from the use of adaptive load-balancing algorithms, and this will be a topic for future study.

	Server 1		Server 2		Server3	
	no.	%	no.	%	no.	%
300 con $\lambda = 3$	102	34%	99	33%	99	33%
900 con $\lambda = 9$	295	32.77%	303	33.66%	302	33.55%
300 con $\lambda = 12$	94	31.33%	96	32%	110	36.66%

Table 2: Round-robin load balancing for small files

6 Conclusions and future work

In this paper we described one of the problems faced by today's Web service providers. The traffic through the Web sites increases along with the number of clients and the number of services offered. Existing solutions, that use Web clusters to distribute the traffic across multiple servers, do not, in general, exploit the QoS capabilities of the underlaying network. We proposed and implemented a novel technique for Web switching, tailored to a next generation switching protocol, MPLS.

Our proposal is a working, cost-effective architecture, for small institutions or corporations, in an open source (Linux) environment. The performance tests showed that the MPLS based solution performs well even for highly loaded web sites (12 connections per seconds means over 1 million hits per day). Moreover, porting the technique to a hardware implementation may improve the performance and reliability of the proposed next generation Web switching technique.

The performance results where obtained empirically on a network using a simple round-robin load-balancing algorithm. Future work will explore adaptive load-balance algorithms, and the development of a queuing model of such a web server system. This will allow the most economic hardware to be deployed to meet the growing demand for diverse web services.

References

[1] Yahoo. Investor relations. URL: http://www.yahoo.com/info/investor.

[2] *Caching Proxies: Limitations and Potentials*, 1995. In Proceedings of 1995 World Wide Web Conference.

[3] Jean-Chrysostome Bolot and Philipp Hoschka. Performance engineering of the World Wide Web: Application to dimensioning and cache design. *Computer Networks and ISDN Systems*, 28(7–11):1397–1405, 1996.

[4] G. Abdulla. Www proxy traffic characterization with application to caching, 1998.

[5] Mirrors of www.gnu.org. URL: http://www.gnu.org/server/list-mirrors.html.

[6] Eric Dean Katz, Michelle Butler, and Robert McGrath. A scalable HTTP server: The NCSA prototype. *Computer Networks and ISDN Systems*, 27(2):155–164, 1994.

[7] httpd.conf: Standard apache configuration file for linux http server.
URL: http://www.apache.com.

[8] Apache Web Server. Apache web site.
URL: http://www.apache.com.

[9] A. Acharya, A. Shaikh, R. Tewari, and D. Verma. Scalable web request routing with mpls. Technical Report RC 22275, IBM Research Report, December 2001.

[10] Thomas T. Kwan, Robert McCrath, and Daniel A. Reed. NCSA's world wide web server: Design and performance. *IEEE Computer*, 28(11):68–74, 1995.

[11] CISCO. Load balancing. a multifaceted solution for improving server availability.
URL: http://www.cisco.com/warp/public/cc/pd/cxsr/400/tech/lobal_wp.htm.

[12] Nortel Networks. Alteon web switching portfolio.
URL: http://www.nortelnetworks.com/products/01/alteon/acedir/index.html.

[13] Daniel Andresen, Tao Yang, Vegard Holmedahl, and Oscar H. Ibarra. SWEB: Towards a scalable world wide web server on multicomputers. Technical Report TRCS95-17, 2, 1995.

[14] Daniel O. Awduche, Angela Chiu, Anwar Elwalid, Indra Widjaja, and XiPeng Xiao. Overview and principles of internet traffic engineering. work in progress.

[15] Uyless Black. *MPLS and Label Switching Networks*. Prentice Hall, 2001.

[16] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus. Requirements for traffic engineering over mpls. Technical Report RFC2702, IETF, September 1999.

[17] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick. A framework for qos-based routing in internet. Technical Report RFC2386, IETF, August 1998.

[18] Netfilter: firewalling, nat and packet mangling for lunix 4.2. URL: http://netfilter.samba.org/.

[19] Azer Bestavros, Mark Crovella, Jun Liu, and David Martin. Distributed packet rewriting and its application to scalable server architectures. Technical Report 1998-003, 1, 1998.