RC 22275 (Log# W0112-022) (12/05/2001)
Computer Science/Mathematics

IBM Research Report

Scalable Web Request Routing with MPLS

A. Acharya, A. Shaikh, D. Verma

IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

R. Tewari

IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

This page intentionally left blank.

Scalable Web Request Routing with MPLS

Arup Acharya, Anees Shaikh, Renu Tewari, Dinesh Verma

IBM T.J. Watson Research Center
Hawthorne, NY 10532

Abstract

High-volume Web server and cache installations achieve scalability and reliability by using a front-end dispatcher to route incoming client requests among a cluster of server machines. Dispatchers typically operate at layer-4, using transport-layer information (e.g., IP address/port), or at layer-7 using application-layer information (e.g., HTTP headers), to direct clients to the appropriate server. Layer-7 dispatchers, while more flexible than layer-4 approaches, suffer from limitations on scalability and performance since they must perform TCP connection termination and management for a large number of clients. In this article we describe an approach that, when combined with an intelligent client-side proxy, can implement a dispatcher using commercial, high-performance, off-the-shelf switching hardware, while also providing the flexibility of a content-aware router. We leverage the growing migration of networks to Multiprotocol Label Switching (MPLS) in order to provide more flexible routing, but rather than using labels to express routing and forwarding policies, our scheme maps application-layer information onto labels to enable high-performance Web request routing. Unlike conventional schemes, our technique assigns some of the dispatch function to an MPLS-aware client-side proxy that applies the appropriate label for a given connection. This approach removes the main bottleneck from the system (i.e. TCP connection termination at the dispatcher), and lends itself to realization in a standard MPLS switch, thus obviating the need for costly, specialized layer-7 Web switch hardware while providing the same functions at a much improved price-to-performance ratio.

Introduction

Current high-volume, high-availability, Internet data centers use clusters of Web servers or caches to achieve scalability and reliability. To serve a large and diverse client population, content can be replicated across servers, or partitioned with a dedicated server for particular content or clients. In such environments a front-end dispatcher (or router) directs incoming client requests to one of the physical server machines, as shown in Figure 1. The physical servers often share one or more virtual IP addresses so that any server can respond to client requests. In other scenarios, the servers have only private addresses, so the dispatcher accepts all connections destined for the site virtual address. The request-routing decision can be based on a number of criteria, including server load, client request, or client identity.

Dispatchers (also called “Web switches” or “content switches”) are typically required to perform several functions related to the routing decision:

- monitor server load and distribute incoming requests to balance the load across servers
- examine client requests to determine which server is appropriate to handle the request
- identify the client to maintain session affinity with a particular server for e-business applications

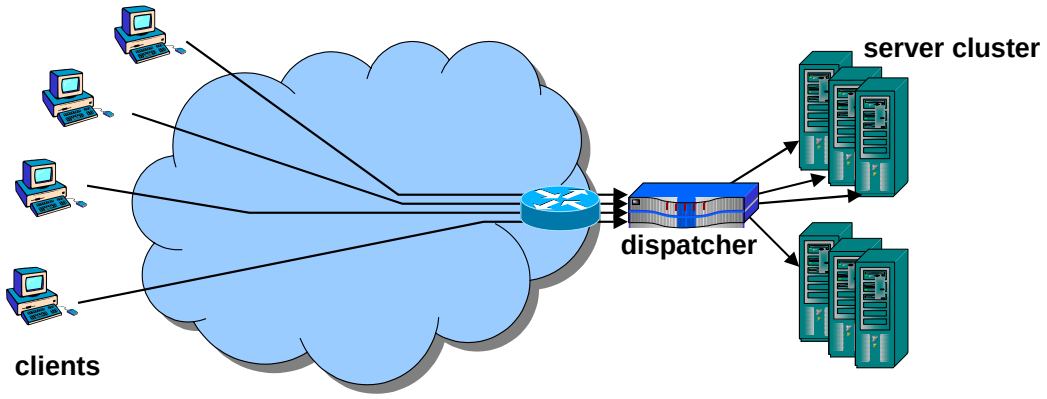

Figure 1: A server-side dispatcher directs incoming client Web requests to one of the physical servers in the cluster.

In addition, most commercial dispatchers provide functions important in a production data center environment. These include:

- failover to a hot standby to improve availability
- detection and avoidance of many common denial-of-service attacks
- SSL acceleration to improve the performance of secure applications
- simplified configuration and management (e.g., Web browser-based configuration interface)

A variety of networking equipment and software vendors offer dispatcher products, including Cisco Systems [1, 2], Nortel Networks [3], IBM [4], Intel [5], Foundry Networks [6], and F5 Networks [7].

Web Dispatcher Technology

Dispatchers may be broadly categorized into two types: layer-4 dispatchers which base decisions on information in the TCP and IP headers alone, and layer-7 dispatchers which use application layer information (e.g., HTTP headers) to direct client requests. Request-routing may be done primarily in hardware, completely in software, or with a hardware switch combined with control software. For example, several of the vendors mentioned above offer dedicated hardware solutions consisting of multiple fast microprocessors, several Fast Ethernet and Gigabit Ethernet ports, and plenty of memory and storage. Others offer software-only solutions that can be installed on a variety of standard platforms.

In this article we begin with an overview of layer-4 and layer-7 dispatcher technology. We focus in particular on scalability issues in layer-7 dispatchers, and describe various techniques that have been proposed to solve them. The second part of the article describes a new approach to implement Web switches that overcomes many of the scalability problems of dedicated layer-7 dispatchers.

Layer-4 dispatchers

The use of layer-4 or layer-7 dispatchers depends on the request-routing goal. Load-balancing across replicated content servers, for example, typically does not require knowledge about the client request or identity, and thus is well-suited to a layer-4 approach. Simple session affinity based on client IP address, or directing requests to servers based on application (e.g., port 80 traffic vs. port 110 traffic) is also easily accomplished by examining layer-3/4 headers of packets while in transit through the dispatcher. For example, the dispatcher may peek at the TCP header flags to determine when a SYN packet arrives from a client indicating a new connection establishment. Then, once a SYN is identified, the source and destination port numbers and IP addresses may be used to direct the request to the right server. This decision is recorded in a table so that subsequent packets arriving with the same header fields are directed to the same server.

Layer-4 dispatchers, due to their relative simplicity, are often implemented as specialized hardware since they need not perform any layer-4 protocol processing or maintain much per-connection state. Although traffic from the clients must be routed via the dispatcher, the response traffic from the server, which accounts for the bulk of the data in HTTP transactions, can bypass the dispatcher, flowing directly back to the client. This is typically done by configuring each server to respond to traffic destined for the virtual IP address(es), using IP aliasing, for example.

Although layer-4 dispatchers are usually deployed as front-end appliances, an alternative is to allow back-end servers to perform load-balancing themselves by redirecting connections to relatively underloaded machines [8]. However, even without such an optimization, hardware-based layer-4 dispatchers are able to achieve very high scalability and performance.

Layer-7 dispatchers

Request-routing based on the URL (or other application-layer information), on the other hand, requires the dispatcher to terminate the incoming TCP connection and receive enough information to make a routing decision. In the case of Web traffic, for example, the dispatcher must accept the incoming TCP connection and then wait for the client to send an HTTP request in order to view application-layer information such as the requested URL, or HTTP cookie. Once enough information to make a routing decision is received, the dispatcher can create a new connection to the appropriate server and forward the client request. The server response is then passed back to the client via the dispatcher on the client's original connection. Figure 2 outlines these steps.

In the simplest realization, a layer-7 dispatcher may be implemented as a software application-level gateway that transparently accepts incoming client connections (destined for port 80 to the server virtual IP address) and reads the HTTP GET requests. After deciding which server should handle the request, the application can forward it on a new or pre-established connection to the server. The dispatcher serves as a bridge between the two connections, copying data from one to the other. From a networking point of view, the dispatcher behaves much like a forward Web proxy installed at an enterprise site, though the forward proxy's primary function lies primarily in filtering and content caching, rather than request routing. In this approach, the dispatcher can quickly become a bottleneck, since it must perform connection termination and management for a large number of clients [9, 10]. This limits the overall scalability of the data center in the number of clients it can support

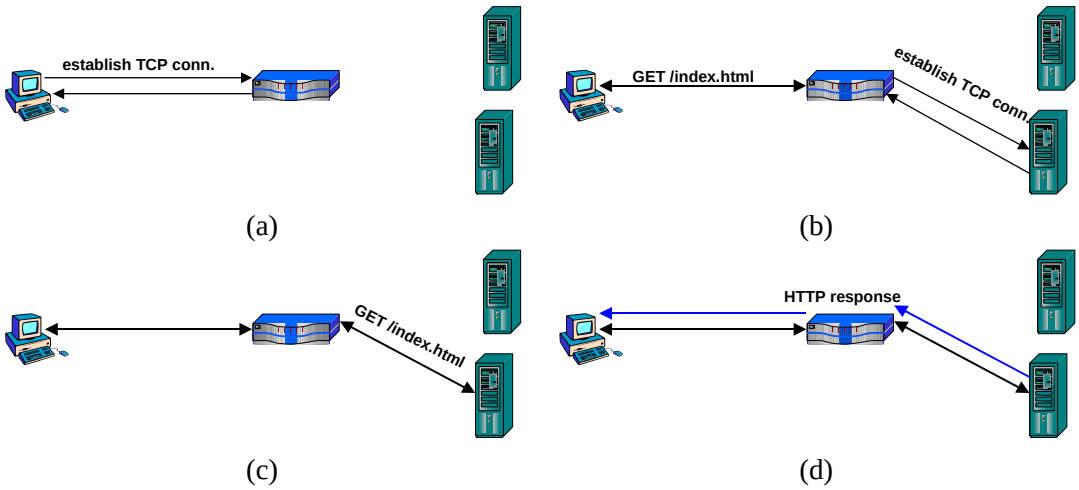

Figure 2: The dispatcher accepts the TCP connection transparently) from the client (a). Next, in (b) the client sends a `GET` request on the established connection, prompting the dispatcher to open a new TCP connection to the appropriate server. In (c) the dispatcher forwards the client request to the server, and, in (d), returns the response to the client.

simultaneously.

Improvements to layer-7 request routing scalability

Several techniques have been proposed to improve the performance and scalability of application-level gateways used in various contexts, including as HTTP proxies. TCP connection splicing is one such optimization in which packets are forwarded from one connection to the other at the network layer, avoiding traversal of the transport layer and the user-kernel protection boundary [11, 12, 10]. TCP splicing mechanisms are usually implemented as kernel-level modifications to the operating system protocol stack with an interface to allow applications to initiate the splice operation between two connections. Once the TCP splice is completed, data is relayed from one connection to the other without further intervention by the application. Figure 3(a) depicts the operation of TCP splicing. With splicing, care must be taken to ensure that TCP header fields such as sequence numbers, checksums, and options, are correctly relayed. TCP splicing has been shown to improve the performance of application-layer gateways to the level of software IP routers. Variations to the kernel-based implementation include implementation in the kernel socket library (as opposed to the network layer) [13] and in a hardware switch [14].

Though TCP splicing can improve the scalability of a layer-7 dispatcher it is still limited by the fact that a centralized node must terminate incoming connections, examine application-layer information, and make request-routing decisions before initiating a splice. In addition, all traffic to and from the servers must pass through the dispatcher to allow the header translation to occur. To address these limitations, an alternate scheme using connection handoff was proposed [15, 9]. In this approach, each back-end server can function as a dispatcher, effectively distributing the content inspection and request-routing operations to multiple nodes. Client connections are initially routed to any one of the servers, perhaps using a fast hardware switch. If the initial server decides that another server is better suited to handle the request, it transfers the TCP connection state to the alternate

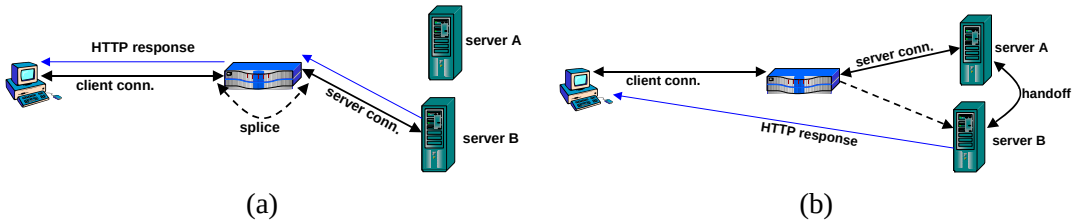

Figure 3: With TCP splicing (a), the dispatcher splices the two connections after determining that server B should handle the request. The response must be sent back through the dispatcher. In the TCP handoff approach (b), the dispatcher, a simpler layer-4 device, initially forwards the connection request to server A. After receiving the request, server A hands off the connection state to server B. The response traffic can then flow directly back to the client, bypassing the dispatcher. Client acknowledgements, however, still come through the dispatcher, and must be forwarded to the server B.

server. Using the transferred state, the new server can resume the connection with the client without requiring that data pass through a front-end dispatcher. Figure 3(b) shows the operation of TCP connection handoff.

While the connection handoff approach does remove the bottleneck of connection termination at a single front-end, its scalability and performance are ultimately limited by the number and overhead of TCP handoff operations. Furthermore, it still requires a special front-end layer-4 dispatcher, since incoming packets (e.g., TCP acknowledgements) must be forwarded to the appropriate server after the connection is handed off. Finally, TCP handoff requires kernel modifications to the server operating systems to support handoff. The splicing approach, on the other hand, is transparent to both servers and clients.

As Web application requirements evolve, there will be a need for more sophisticated dispatching, based on a variety of application information. This trend implies that the layer-4 approach of examining only transport-layer headers provides insufficient functionality. But layer-7 dispatchers, while more sophisticated, suffer from the limitations on scalability and performance described above.

Ideally, a dispatcher should provide the flexibility of layer-7 forwarding, while maintaining performance levels comparable with layer-4 hardware switches. In the remainder of this article we describe an approach that, when combined with an intelligent client-side proxy, can implement a dispatcher using commercial, high-performance, off-the-shelf switching hardware, while also providing the flexibility of a layer-7 dispatcher.

MPLS-based Solution Overview

In our proposed solution, we leverage the growing migration of networks to Multiprotocol Label Switching (MPLS) to provide flexible routing and the ability to perform network traffic engineering [16, 17]. MPLS provides a circuit-switching service in a hop-by-hop routed network [18]. It achieves this by grouping related packets by assigning them a common, fixed-size label. Packets sharing a label belong to the same forwarding equivalence class (FEC) and can be routed and treated the same way in the network. Standard usage of MPLS involves establishment of arbitrary label-switched paths (LSPs) for forwarding particular classes of traffic. LSPs may also be nested by stacking MPLS labels where an outer label might be used to assign traffic to a common network-wide path, while an inner label could be used to demultiplex traffic among classes of traffic on that path. An

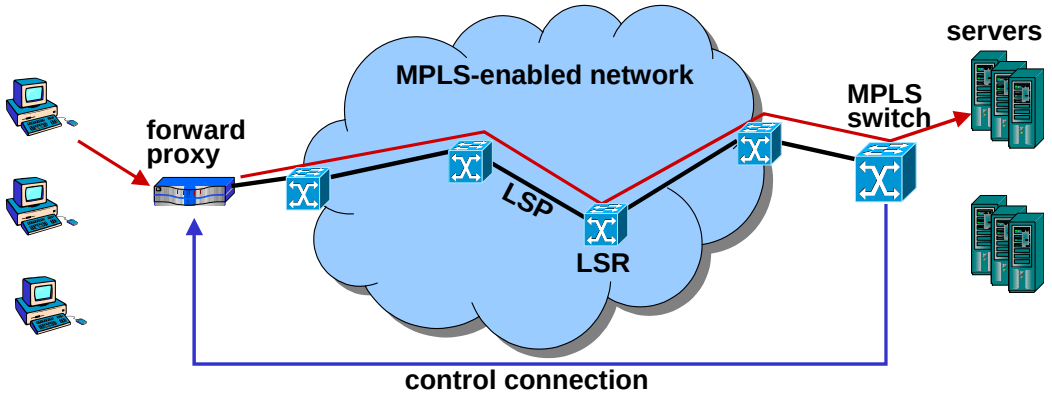

Figure 4: The MPLS-based Web switching architecture consists of an MPLS-enabled client-side proxy, an MPLS network, and an MPLS switch acting as a dispatcher at the server cluster. The dashed line shows a connection request created by a client, terminated and examined by the proxy, and switched through the network and dispatcher directly to the appropriate server.

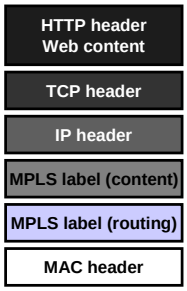

Figure 5: With an Ethernet link layer, an MPLS label stack resides between the MAC layer header and the packet's IP header. The remainder of the packet (TCP and HTTP headers) remains unchanged.

MPLS-enabled network consists of label-switched routers (LSRs) that implement the MPLS protocols. Several major network service providers recently announced deployments of MPLS, and most core network router vendors support MPLS in their products [19]. In addition, next generation optical networks are likely to use MPLS in the control plane [20].

Figure 5 illustrates the structure of packets containing MPLS labels. In an Ethernet link layer, the labels are inserted between the layer 2 MAC header and the layer 3 IP header. Other link layer technologies use different labeling mechanisms. For example, in ATM the labels are placed in the VPI/VCI fields in each cell.

An attractive feature of MPLS is that labels do not have a built-in semantic, as with transport-layer port numbers or network-layer addresses. That is, labels are used only to map a packet from an LSR input port to an output port. Our scheme takes advantage of this flexibility by mapping application-layer information onto labels to enable high-performance Web switching, rather than using labels to express routing and forwarding policies (as is customary). Figure 4 shows the overall architecture, which consists of an MPLS-enabled client proxy, an MPLS network, and a server cluster fronted by a dispatcher which consists of a software controller and a standard MPLS switch.

The proxy is responsible for mapping labels onto packets belonging to client connections. The label applied

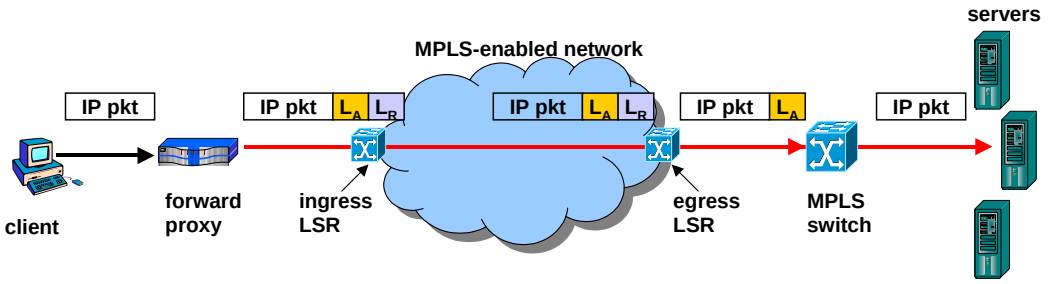

Figure 6: MPLS label stacking allows the use of a application-layer label (L_A) that is visible only to the endpoints along with a separate label (L_R) for routing in the core network.

by the proxy is used at the dispatcher to choose which server should handle the request. The label stacking feature of MPLS allows this inner label to be independent of any outer labels used to route the request through the network. As shown in Figure 6, for example, the client sends its request to the Web proxy as a standard IP packet. When the MPLS-aware proxy makes a corresponding connection to the server cluster, it pushes an appropriate label (L_A) onto the label stack. As packets enter the network, the ingress LSR pushes another label (L_R) onto the label stack to facilitate routing through the MPLS network. This routing label is popped off at the egress LSR leaving only L_A for the dispatcher. Finally, based on L_A, the dispatcher routes the packet to the corresponding server using IP routing. If the server network (including the servers) is also MPLS-enabled, the dispatcher need not route the packet using IP; it can simply switch the packet all the way to the server.

The mapping of client connections to labels is communicated by the dispatcher to the proxy using a persistent control connection. Depending on the mapping, the dispatcher can support a variety of functions without having to terminate TCP connections:

- content-based routing
- client-server affinity
- client-specific service differentiation
- server load balancing

The approach has several key advantages over competing solutions that use a front-end Web switch. First, it removes the main bottleneck from the system, namely the single point where all TCP connections are terminated and application-layer information is examined. Second, it lends itself to realization in a standard off-the-shelf MPLS switch, thus obviating the need for specialized, layer-7 dispatcher hardware. Finally, using this approach, many of the functions typically available only with layer-7 Web switches, are provided at a much improved price-to-performance ratio.

Unlike conventional schemes, this technique assigns some of the dispatch function to the proxy, since the proxy must apply the correct label for a given connection. This requires that the proxy is MPLS-aware and that it implements the protocol to receive mapping information from the dispatcher. Below we describe briefly how this protocol could be implemented, along with a discussion of how each of the four functions mentioned above may be realized in this architecture.

Label distribution to proxies

We assume that the dispatcher maintains persistent connections with each of the MPLS-enabled proxies accessing the server (e.g., using HTTP). The proxy initiates the communication by first contacting a well known server address for label distribution (`label.server.com`) or a well known port on the server (`www.server.com:8090`). This control connection is kept-alive and the commands from the dispatcher to the proxy are sent as User extensions to the HTTP header.

The dispatcher can distribute labels that are shared across proxies, or send each proxy a unique label set. If labels are shared, it is difficult to identify requestors in order to determine which clients are contributing the most traffic to the site, or to provide service differentiation. On the other hand, if the number of participating proxies is very large, providing non-overlapping labels results in inflation of the label mapping table at the dispatcher. Hence, we adopt a hybrid approach that assigns unique labels to proxies that require service differentiation and common labels to other proxies.

The dispatcher directs the proxy to insert labels according to some policy, depending on the functionality required. In the following sections we describe some details of the information communicated to the proxy to facilitate the request-routing functions.

Content-based routing

Content-based routing is useful when content is partitioned across the server cluster such that only a subset of servers can respond to a given request. In this case the proxy assigns a label based on the client request. The dispatcher can provide the request-to-label mappings in a number of ways, depending on how much flexibility is required.

One mechanism is to distribute the labels along with a hash function to the proxy. The proxy can apply the hash function to the URL being requested to determine which label to use. In practice it may be sufficient to divide content among servers in a coarser fashion. For example if content is partitioned based on directory paths, the dispatcher could send (path, label) pairs such as $<$(/, L_1),(/pc, L_2),(/linux, L_3)$>$ to the proxy.

Another possibility is to serve Web pages with hyperlinks that encode labels based on the URL. For example the first request for `http://www.example.com/index.html` could be served from any Web server, using a default label. But the links on the `index.html` page could be transformed into a form like `http://www.example.com/<image_content_label>/image.gif`. The forward proxy, on seeing such a URL, could strip the label from the URL and insert it in the request packets. The dispatcher then switches the request to the appropriate server.

Client affinity

In e-business applications a single transaction may consist of multiple requests and responses before the transaction is completed. Once a client is directed to a particular server where some session state is established, it is desirable to direct the client to the same server for the duration of the transaction. In this case, the label attached by the proxy is used to identify which server earlier serviced the client for the ongoing session.

To handle persistence, the proxy assigns the same label (from a set of labels provided by the dispatcher) to a given client for the duration of a session. In this case a client will always use the same label and be directed to the same server. One problem is that the proxy has to be able to identify the start and end of a session. One simple approach is to use a fixed timeout value (e.g., corresponding to SSL session timeout). However, this does not guarantee that ongoing sessions will not be prematurely transferred to another server. We also consider other approaches, such as designating specific URLs as indications of the beginning and end of a transaction. The proxy could then assign labels based on whether client affinity is active or not. Still another way is to use cookies. The server sets a "session-on" cookie when a particular URL is accessed. As long as the cookie is set, the proxy checks for cookies in the HTTP header and labels it as an ongoing session. If the cookie is reset or absent the proxy assigns labels without regard for affinity. This requires that the proxy be informed of the "session-on" cookie by the label distributor.

Service differentiation

It may be desirable to provide different classes of service to clients based on service level agreements or other administrative arrangements. The dispatcher can provide different label sets for the different classes of service. The proxy then assigns labels to clients based on the type of service they require. For example, the dispatcher could provide the proxy with three labels corresponding to gold, silver, and bronze service. At the dispatcher, requests can be dispatched to servers based on the service class, with gold-labeled packets switched to the best performing server, for example. Label stacking can also be used to identify the organization and then class within the organization to provide hierarchical classes of service.

Load balancing

For load balancing the proxy assigns labels to client requests such that the load across all the servers is approximately equal, assuming that each request can be serviced by more than one (or all) servers. The dispatcher sends a list of labels to the proxy, along with an associated weight for each label and a selection policy. For example the dispatcher could send a tuple $<\{(L_1, w_1)$, $(L_2, w2)$, $\ldots\}$, *WRR*$>$ listing labels and their corresponding weights to be used in a weighted round robin fashion. This scheme will achieve coarse-grained (i.e., not per-connection) load balancing, but temporary load imbalances may arise from the random nature of the requests.

If a load imbalance occurs, the dispatcher can send the proxy a new set of weights for the label assignment such that incoming connections are shifted away from a heavily loaded server until the load is back within limits. In the case when a server becomes unavailable, sending a weight of zero for the corresponding label(s) implicitly removes the server from operation. During the transition, however, the proxy may initiate new connections labeled for the unavailable server. To prevent these new connections from failing, the original label can be remapped at the server side by modifying the switching table in the MPLS dispatcher.

It is worth noting that providing client affinity and load balancing together requires some additional consideration. If the dispatcher wishes to correct a load imbalance with a new set of labels, the proxy must continue to use the old label set for all ongoing sessions. Only when they complete can the proxy transition to use the new label set. In the interim new client sessions may be initiated using the new labels.

	DNS-based	MPLS	L-4 dispatcher	L-7 dispatcher
load-balancing granularity	per name resolution request (subject to caching)	per-connection at proxy	per-connection at dispatcher	per-connection at dispatcher
location of request-routing decision	authoritative nameserver	split between proxy and dispatcher	server side	server side
scalability limits	number of name resolutions	layer-2 switching operations and number of participating proxies	layer-4 switching operations	TCP connection terminations and layer-7 switching operations
deployment	modified nameservers and possibly modified content	MPLS-enabled proxies	layer-4 dispatcher in front of server cluster	layer-7 dispatcher in front of server cluster

Table 1: Comparision of approaches for request-routing in server clusters.

Request routing trade-offs

In Table 1 we summarize some trade-offs between the MPLS request-routing approach and layer-4 and layer-7 dispatchers for several criteria. Load balancing granularity refers to how quickly incoming connections can be shifted away from overloaded servers in the cluster. The request routing decision location is which entity in the system decides which server will handle a request. The scalability limits column lists which functions ultimately limit the ability to handle a growing number of client requests. Finally deployment refers to network elements that must be modified to support each approach. We also include in the table a comparison with DNS-based request routing. Though DNS is primarily used for wide-area request routing, it can also be used for load balancing within a server cluster, for example using round-robin DNS.

Deployment Issues

As with any proposal involving changes to established network infrastructure, there are several deployment issues that arise. Below we comment on some of these.

Proxy participation scenarios

As discussed earlier, our scheme relies on two primary assumptions:

- wide deployment of MPLS in service provider networks
- proxies at the edges of the network will be modified to implement MPLS and support the protocol to communicate with dispatchers in server installations.

The first assumption reflects our belief regarding MPLS acceptance and deployment, and is supported by reports from some large ISPs and equipment vendors [16, 17, 19]. The second assumption is appropriate in

settings where proxies and Internet data centers are under a single administrative control. For example many ISPs also offer Web hosting service. In these cases, it is in the interest of the service provider to deploy such proxies because they contribute to improving the capacity of the provider's installations. Service providers are already able to provide MPLS-based VPN and firewall services as an alternative to enterprises installing these proxies on their premises [21, 22].

Scalability is still a concern if we assume that every forward proxy accessing a web site participates in this scheme. Instead, we expect that in many cases a relatively small set of proxies contributes the bulk of the traffic to a site, and it is feasible to modify only those proxies. Another scenario that justifies proxy participation is one in which the Web hosting provider wants to provide service level agreements to particular clients. In this case the hosting provider requires proxy participation from those clients as part of the agreement.

Interim solutions

The benefits of MPLS-based request-routing can be fully realized only after the conditions described above are satisfied. Our proposed scheme can, however, be decoupled from MPLS by viewing it simply as a way to encode application layer information in lower-layer network headers. In the case of MPLS, we map layer-7 information onto layer-2 labels. Instead, we could encode layer-7 information about the connection onto the transport layer or network layer in port numbers or IP addresses, respectively. For instance, rather than distributing labels to the proxy, the dispatcher can distribute port numbers along with corresponding URL paths to achieve content routing. At the server-side, a network address translation (NAT) device or layer-4 dispatcher could examine the port number in the incoming TCP header and direct the packet to the appropriate server without having to terminate the connection.

This alternative approach is attractive in that it does not immediately require MPLS-enabled forward proxies. Moreover, Web proxy software can be configured to rewrite port numbers in Web requests, though proxies would still need to be modified to support the protocol to communicate with the dispatcher. This scheme overloads the port number semantics, however, which may cause additional complications. For example, the dispatcher cannot re-map port numbers independently of servers, since servers must be listening on those ports to respond to requests. Also, if the dispatcher re-maps the port number, the return path must go through the dispatcher if the target server was not listening on the original port.

Summary

In this article, we surveyed the state-of-the art in Web switch technology, identifying the primary issues that limit scalability for sophisticated layer-7 dispatchers. We proposed a novel architecture to improve the Web request-routing capacity of server and cache clusters. We capitalize on the growing deployment of MPLS by mapping application-layer information onto layer-2 labels to facilitate sophisticated request-routing functions without the bottleneck of TCP connection termination. We require client-side proxies to participate by applying appropriate labels to client requests, and can use off-the-shelf MPLS switches to perform the dispatching. This approach allows the dispatcher to perform key functions including content-based routing, server load-balancing, client affinity, and service differentiation, at hardware switching speeds.

References

[1] "Cisco LocalDirector 400 series," http://www.cisco.com/warp/public/cc/pd/cxsr/400/.

[2] "Cisco CSS 1100," http://www.cisco.com/warp/public/cc/pd/si/11000/.

[3] "Alteon ACEdirector," http://www.nortelnetworks.com/products/01/acedir.

[4] "Websphere edge server," http://www.ibm.com/software/webservers/edgeserver/.

[5] "Intel NetStructure 7175 traffic director," http://www.intel.com/network/idc/products/director_7175.htm.

[6] "Foundry ServerIron," http://www.foundrynet.com/products/webswitches/serveriron.

[7] "BIG-IP controller," http://www.f5.com/f5products/bigip/.

[8] Azer Bestavros, Mark Crovella, Jun Liu, and David Martin, "Distributed packet rewriting and its application to scalable server architectures," in *Proceedings of IEEE International Conference on Network Protocols*, Austin, TX, October 1998.

[9] Mohit Aron, Darren Sanders, Peter Druschel, and Willy Zwaenepoel, "Scalable content-aware request distribution in cluster-based network servers," in *Proc. of USENIX Annual Technical Conference (USENIX 2000)*, San Diego, CA, June 2000.

[10] Ariel Cohen, Sampath Ragarajan, and Hamilton Slye, "On the performance of TCP splicing for URL-aware redirection," in *Proc. of Symposium on Internet Technologies and Systems (USITS '99)*, Boulder, CO, October 1999.

[11] David Maltz and Pravin Bhagwat, "TCP splicing for application layer proxy performance," Tech. Rep. RC 21139, IBM TJ Watson Research Center, 1998.

[12] Oliver Spatscheck, Jorgen S. Hansen, John H. Hartman, and Larry L. Peterson, "Optimizing TCP forwarder performance," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, pp. 146–157, April 2000.

[13] Marcel Rosu and Daniela Rosu, "Evaluation of TCP splice benefits in Web proxy servers," Tech. Rep. RC 22159, IBM Research Report, August 2001.

[14] George Apostolopoulos, David Aubespin, Vinod Peris, Prashant Pradhan, and Debanjan Saha, "Design, implementation and performance of a content-based switch," in *Proceedings of IEEE INFOCOM*, March 2000.

[15] Vivek S. Pai, Mohit Aron, Gaurav Banga, Michael Svendsen, Peter Druschel, Willy Zwaenepoel, and Erich M. Nahum, "Locality-aware request distribution in cluster-based network servers," in *Proc. of Architectural Support for Programming Languages and Operating Systems (ASPLOS VII)*, October 1998.

[16] Denise Pappalardo, "UUNET's massive MPLS deployment underway," *Network World Fusion*, July 1999, http://www.nwfusion.com/news/1999/0721mpl.html.

[17] David Rohde, "MPLS finds its way deeper into access services," *Network World Fusion*, February 2001, http://www.nwfusion.com/edge/columnists/2001/0212edge1.html.

[18] Eric C. Rosen, Arun Viswanathan, and Ross Callon, "Multiprotocol label switching architecture," Internet Request for Comments (RFC 3031), January 2001.

[19] The MPLS Resource Center, "MPLS Vendor Information," http://mplsrc.com/vendor.shtml, 2001.

[20] Peter Ashwood-Smith et al., "Generalized MPLS – signaling functional description," Internet Draft (work in progress), March 2001.

[21] Teleglobe, "Teleglobe partners with innovatia to offer managed cpe router program," Corporate news release, November 2001, http://www.teleglobe.com.

[22] CoSine Communications, "MPLS VPNs," http://www.cosinecom.com/solutions/mpls_vpns.html.