Web Switching (Draft)

Giuseppe Attardi, Università di Pisa

Angelo Raffaele Meo, Politecnico di Torino

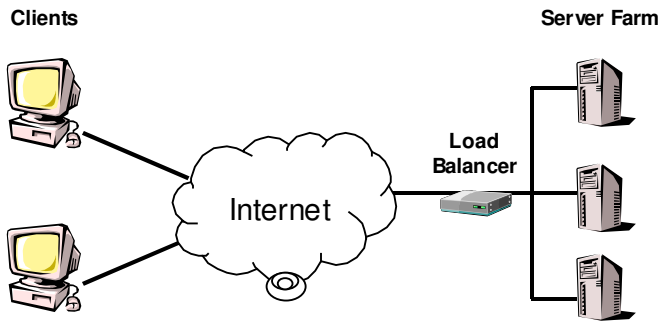
January 2003

1. The Problem

Web servers are becoming the primary interface to a number of services, that are no longer limited to browsing, but now also provide access to services through approaches like XML Web Services, and later will enable full-scale two-way applications, and eventually will evolve into grid based computing.

When serving a large number of clients (thousands concurrently and millions transactions per day) the Web server becomes a bottleneck, and hence a number of solutions have been proposed, which involve some form of replication and a mixture of techniques for performing load balancing which are neither adequate nor scalable.

The typical setting involves a farm of Web servers and a load balancer located in front of them, which redirects requests to one of them according to the pre-configured policy.



Approaches to load balancing include:

- DNS round robin
- NAT
- Direct routing
- IP Tunneling
- LocalDirector or Packet Redirector

DNS round robin is just a hack, not a real solution, even if is often used since it requires no special device or software, being handled by the DNS service itself.

Traditional load balancing methods have significant drawbacks and limitations:

- content blindness
- lack of session persistence:
  - multiple TCP connections in an application layer session (SSL session, server side scripts supported session)
  - L4 load balancer may redirect these TCP connections to different servers: this is a mistake
  - SSL session cannot be reused in multiple TCP connections: implies performance degradation.

1.1. Packet director

A packet redirector is the function to redirect IP packets according to certain policies. Redirectors can be implemented either in software or in hardware.

1.1.1. Software redirector

A software redirector is a packet filter, implemented as an application running on a machine, typically a Linux box, dedicated to the task of filtering and routing packets.

The problem with such solution is that the network stack is handled by the operating system. Each packet arriving on a network interface generates an interrupt. Since interrupt handling is performed by the OS, under heavy traffic loads, this trashes the OS, since when returning from an interrupt, it needs to service another one immediately. Packets get dropped, but with high loads the system becomes unresponsive, i.e. incapable of performing any user level task (it is even impossible to reset the OS, and rebooting the machine is necessary) or even gets into livelock, becoming unable to complete any useful work.

A possible solution is to use polling device drivers, or to limit the maximum rate at which a card can generate interrupts, exploiting features in certain hardware network interface cards, which are capable of buffering a number of requests. Experiments reported by Luigi Rizzo [personal communication], show that nonetheless improvements are limited since general purpose NIC are not well tuned: manufactures, e.g. Intel, are aware of such limitations, but not particularly interested in providing a solution. A more promising solution involves changing the scheduling policy in the OS kernel [2]. This allows the kernel to handle a number of requests with a single context switch.

1.1.2. Hardware redirector

Several products exist on the market that perform redirection in hardware.

For instance *Cisco LocalDirector* is hardware and software that intelligently load balances IP traffic across multiple servers. It is based on three elements:

- Service Manager
- Forwarding Agent
- Workload agent

The MNLB Services Manager makes the load-balancing decisions based on application availability, server capacity, and load distribution algorithms such as round robin or least connections, or the Dynamic Feedback Protocol (DFP).

The MNLB Forwarding Agent is an IOS-based packet redirector that forwards packets based on instructions received from the Services Manager

Workload Agents are value-added software components developed for specific platforms by third-party developers. Workload Agents run on server platforms or on platforms that manage server farms.

1.2. Web server functionalities

Other problems, beyond load balancing, that need to be addressed in high performance Web servers are:

1. content routing: contents is partitioned among several servers and the switch determines the most suitable server to which to route the request
2. session management, by routing requests in the same session to the same server
3. request routing: routing requests to different servers according to the originating request.

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 server machines. The request-routing decision can be based on a number of criteria, including server load, client request, or client identity. Dispatchers are typically required to perform several functions related to the routing decision:

- examine client requests to determine which server is appropriate to handle the request (i.e., content-based routing)
- identify the client to maintain affinity with a particular server for e-business applications.
- monitor server load and distribute incoming requests as to balance the load across servers

Dispatchers may be broadly categorized into two types: layer-4 dispatchers which base decisions on TCP and IP headers alone, and layer-7 dispatchers which use application layer information (e.g., HTTP headers) to direct clients. Request routing may be done primarily in hardware, completely in software, or with a hardware switch combined with control software.

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 of the client request or identity, and thus is well-suited to a layer-4 approach. Content-based routing, on the other hand, requires the dispatcher to terminate the incoming connection, examine the higher-layer headers, and either creates a new connection with the appropriate server (connection splicing), or transfer the connection to the appropriate server (connection handoff).

1.3. Session Management

Session management is critical to provide the illusion of a state-full interaction, within a stateless service like HTTP. Sessions are needed for allowing an application to progress through the interaction with the user, to maintain the authentication, etc.
The typical solution for maintaining sessions is to use cookies, that the server uploads to the client, and that the browser/client include within each successive request to the same server.
Unfortunately, for various reasons, including privacy, such cookies can only be sent to the same server that originated them. This conflicts with the goal of load balancing, which tries to assign a server to each requests based on the servers' current load. Services that deploy a number of servers to balance load, must deal with this issue, either by replicating the cookies on all servers, or by exchanging information among the servers to determine the status of the session.
This is a cumbersome operation that generates additional load and may reduce the benefits of replication.

1.4. SOAP and Web Services

As the Web evolves towards a general-purpose platform for Web Computing, new requirements arise in terms of performance and latency of request/responses. Web services rely on specific protocols like SOAP, encapsulated within normal HTTP packets. Such protocols can be made more aware of the requirements for transport and dispatching. For instance the answer to a method call instantiating a Web Service may return information, in form of a label or token, that can be inserted in later calls to the same instance, having the transport layers to handle more efficiently the routing including the dispatch of the message not just to the host but to the specific service object.

More sophisticated dispatching is hence needed, based on a variety of application information. The layer-4 approach of examining only transport-layer headers provides insufficient functionality. But layer-7 dispatchers, while more sophisticated, suffer from limitations on scalability and performance since they must perform connection termination and management for a large number of clients. Ideally, a dispatcher should provide the flexibility of layer-7 forwarding, while maintaining performance levels comparable with layer-4 hardware switches. 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-based router.

Layer4 switches using TCP/IP headers or application layer header for dispatching user requests are suitable for load-balancing and high performance but are unable to support content routing. Content routing requires terminating incoming TCP connections and examining higher-layer headers. For this reason layer7 dispatchers suffer from scalability and performance limitations.

2. Approach

One possible approach is to exploit standard MPLS switching hardware for developing a scalable, high performance content-aware Web switch.

This will entail creating MPLS enable Linux PC's by developing a software layer to handle MPLS and dispatching in the kernel.

This is a different and complementary approach to kernel-based Web server like Tux, since the approach is protocol independent: it can handle for instance SMTP, IMAP, IFS requests.

It is complementary to the Web server since it acts as a fully transparent proxy at the IP level, that handles packets, analyzing their Layer 4 content and performs part of the task that the Web or application server will have to deal.

One advantage of the approach is that the Dispatcher can perform its task just by looking at a single packet, without having to accept the whole message, going through all layers, as a normal TCP proxy will have to do.

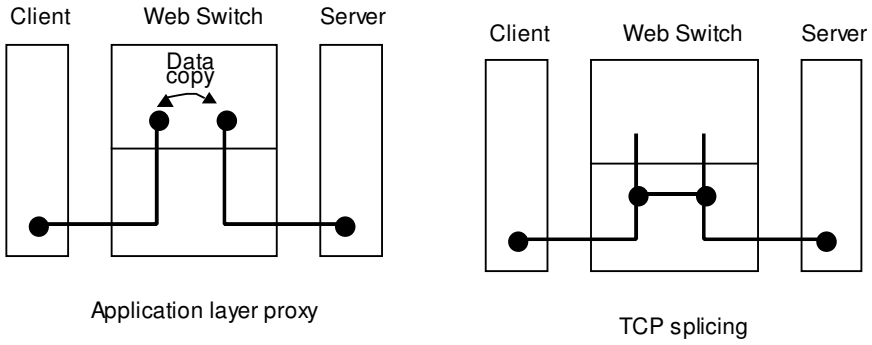
As discussed later, an approach would be to use MPLS labels as a way to identify service/session/user identification, but we will provide initially a solution based on host/port#/URL.

We propose to develop a high-performance Web Switch, which performs the following functions:

- It behaves as a transparent proxy, monitoring all traffic destined to a virtual Web server.
- It behaves as a Layer 4 switch, analyzing TCP packets, and determining the service/session/user request and performing dispatch to the appropriate server in a local server farm
- It performs content based routing, in order to offload such decisions from the Web server
- It maintains tables to perform dispatching and timeout information to recreate and reassign services in case of failure or lack of availability
- It performs load-balancing decisions based on application availability, server capacity, and load distribution algorithms such as round robin or least connections, or the Dynamic Feedback Protocol (DFP)
- It provides MPLS support, in order to exploit MPLS enable networks to perform routing.

To provide support for Web Services, we will consider the possibility of exploiting headers in the SOAP protocol in order to identify the object whose method is being invoked and speed up also message dispatching.

The architecture of the Web Switch will be based on TCP splicing at the kernel level:



Overall the Web switch will provide the following features:

- URL mapping: each HTTP request is directed to the set of machines that provide the requested resource.
- Session persistence**:** each stateful web transactions is consistently directed to same back-end server
- Multiple application sessions**:** Allows each user to maintain multiple application sessions, for example, with a web-mail system, a search system and a purchasing system. The user does not need to log out of each application before using the next.
- HTTP KeepAlive support**:** Extracts web requests from a high-performance HTTP Keepalive connection and directs each individually.
- Locality-Aware Request Distribution**:** all requests for the same resource are directed to the same back-end server, for optimum response times and cache utilization.

2.1. Cookie handling

Before accepting the next request, the Web switch needs to know where to switch, and it must maintain session persistence across multiple Web servers. However not all of contents need session persistence and user may request multiple servers, each requiring session persistence. Therefore the Web switch will have to handle cookies properly.

In order to prevent from re-fragmentation, the dispatcher will perform cookie name rewriting, changing the original “session cookie name” to a special format [5].

2.2. Content Routing

The Web Switch inspects and classifies URL according to several criteria, in particular it can use file extension to determine whether the requests is for a static page or for a CGI, PHP, ASP service, choosing and appropriate server for the task.

2.3. Label Distribution

The label distribution can be handled as follows:

- Control connection between dispatcher and proxy
  - Content based routing: URL/label mappings
  - Load balancing: set of labels, proxy round-robins client requests across labels (could also send weights+labels)
  - Affinity: set of labels, proxy assigns same label to all client requests within a given time T
  - Service differentiation: proxy receives sets of labels, one set per class (gold/…/..), assumes pre-defined service agreement
- Dispatcher populates label table at layer2

Performance goals: number of HTTP requests should scale linearly with the number of servers, without the dispatcher being a bottleneck. This must be achieved while providing the additional functionality of content-based routing, preserving sessions, etc.

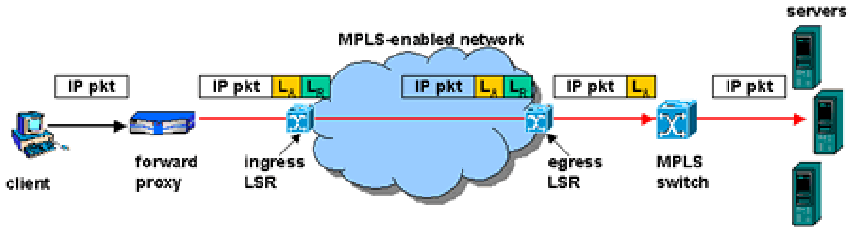
Deployment scenarios for the solution include:

- Highly visited Web sites
- ISPs offering Web hosting services
- Web hosting with special SLA requirements

2.4. MPLS

Multiprotocol Label Switching (MPLS) provides flexible routing and mechanisms for network traffic engineering. MPLS provides a circuit-switching service in a hop-by-hop routed network. 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 MPLS enabled network consists of label-switched routers (LSRs) that implement the MPLS protocols.

The proxy is responsible for mapping labels onto packets belonging to client connections. The label applied 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 the Figure below, 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_B) 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 MPLS based solution is attractive since it may exploit standard MPLS based hardware, however it has the limitation, that if proxy and dispatcher are in different MPLS domains, the solution will depend on inter-domain MPLS standards (yet to be defined).

2.5. MPLS on Linux

MPLS for Linux is a project to implement a MPLS (Multi Protocol Label Switching) stack for the Linux kernel, and portable versions of the signaling protocols associated with MPLS. http://sourceforge.net/projects/mpls-linux.

3. Benchmarking

Through benckmarking of the solution will be performed. This will require setting up a configuration consisting of:

- 8-16 Web servers, running Apache
- 1 Gigabit switch
- 4 client PCs, running Webbench

4. Benefits of the solution

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 switch, thus obviating the need for specialized, layer-7 Web switch 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.

References

[1] Cisco, Load Balancing with the MNLB Feature Set for LocalDirector, http://www.cisco.com/univercd/cc/td/doc/product/iaabu/localdir/mnlb/mnldov.htm.

[2] Luigi Rizzo, Polling versus Interrupts in network device drivers, BSD Conference Europe, 2001, http://info.iet.unipi.it/~luigi/bsdcon01/polling.

[3] A. Acharya, A. Shaikh, R. Tewari, D.C. Verma, Web Switching using MPLS. MPLS World News, 2003. http://www.mplsworld.com/archi_drafts/focus/analy-ibm.htm.
See also:
http://domino.watson.ibm.com/confrnc/nynet/nynet01/nynet.nsf/program/$File/acharya.pdf?OpenElement

[4] C. Edward Chow, Weihong Wang, “Design and Implementation of a Linux-based Content switch,” CCL document, page 11.

[5] P.T. Tsai, Y.D. Lin, Direct Routed Web Switch, with State Migration, TCP Masquerade and Cookie Rewrite. http://contest.softwareliberty.org/final-result/sys/107/final_a.PDF (or http://contest.softwareliberty.org/final-result/sys/107/brief.PDF).