Design and Implementation of a Linux-based Content switch

C. Edward Chow and Weihong Wang

Department of Computer Science

University of Colorado at Colorado Springs

1420 Austin Bluffs Parkway

Colorado Springs, CO 80933-7150

USA

Email: chow@cs.uccs.edu, wwang@cs.uccs.edu

Tel: 2-1-719-262-3110

Fax: 2-1-719-262-3369

Name of Corresponding Author: C. Edward Chow

Keywords: Internet Computing, Cluster Computing, Content Switch, Network Routing

Abstract

In this paper we present the design of a Linux-based content switch, discuss ways for improving the TCP delay binding and the lessons learnt from the implementation of the content switch. A content switch routes packets based on their headers in the upper layer protocols and the payload content. We discuss the processing overhead and the content switch rule design. Our content switch can be configured as the front end dispatcher of web server cluster and as a firewall. By implementing the http header extraction and xml tag extraction, the content switch can load balancing the request based on the file extension in the url and routes big purchase requests in XML to faster servers in e-commerce system. The rules and their content switching rule matching algorithm are implemented as a module and hence can be replaced without restarting the system. With additional smtp header extraction, it can be configured as a spam mail filter or virus detection/removal system.

1 Introduction

With the rapid increase of Internet traffic, the workload on servers is increasing dramatically. Nowadays, servers are easily overloaded, especially for a popular web server. One solution to overcome the overloading problem of the server is to build scalable servers on a cluster of servers [1, 2]. A load balancer is used to distribute incoming requests among servers in the cluster. Load balancing can be done in different network layers. A web switch is an application level (layer 7) switch, which examine the headers from layer 3 all the way to the HTTP header of the incoming request to make the routing decisions. By examining the HTTP header, a web switch can provide the higher level of control over the incoming web traffic, and make decision on how individual web pages, images, and media files get served from the web site. This level of load balancing can be very helpful if the web servers are optimized for specific functions, such as image serving, SSL (Secure Socket Layer) sessions or database transactions.

By having a generic header/content extraction module and rule matching algorithm, a web switch can be extended as a content switch [3, 4] for route packets including other application layer protocols, such as SMTP, IMAP, POP, and RTSP. By specifying different sets of rules, the content switch can be easily configured as a load balancer, firewall, policy-based network switch, a spam mail filter, or a virus detection/removal system.

1.1 Goals and motivation for Content switch

Traditional load balancers known as Layer 4 (L4) switches examine IP and TCP headers, such as IP addresses or TCP and UDP port numbers, to determine how to route packets. Since L4 switches are content blind, they cannot take the advantages of the content information in the request messages to distribute the load. For example, many e-commerce sites use secure connections for transporting private information about clients. Using SSL session IDs to maintain server persistence is the most accurate way to bind all a client's connections during an SSL session to the same server. A content switch can examine the SSL session ID of the incoming packets, if it belongs to an existing SSL session, the connection will be assigned to the same server that was involved in previous portions of the SSL session. If the connection is new, the web switch assigns it to a real server based on the configured load balancing algorithm, such as weighted least connections and round robin. Because L4 switches do not examine SSL session ID, which is in layer 5, so that they cannot get enough information of the web request to achieve persistent connections successfully.

Web switches can also achieve URL-based load balancing. URL based load-balancing looks into incoming HTTP requests and, based on the URL information, forwards the request to the appropriate server based on predefined polices and dynamic load on the server.

XML are proposed to be the language for describing the e-commerce request. A web system for e-commerce system should be able to route requests based on the values in the specific tag of a XML document. It allows the requests from a specific customer, or purchase amount to be processed differently. The capability to provide differential services is the major function provided by the web switch. Intel XML distributor is such an example, it is capable of routing the request based on the url and the XML tag sequence [3].

The content switching system can achieve better performance through load balancing the requests over a set of specialized web servers, or achieve consistent user-perceived response time through persistent connections, also called sticky connections.

1.2 Related Content switching Techniques

Application level proxies [5,6] are in many ways functionally equivalent to content switches. They classify the incoming requests and match them to different predefined classes, then make the decision whether to forward it to the original server or get the web page directly from the proxy server based on proxy server's predefined behavior policies. If the data is not cached, the proxy servers establish two TCP connections –one to the source and a separate connection to the destination. The proxy server works as a bridge between the source and destination, copying data between the two connections. Our proposed Linux-based Content Switch (LCS) is implemented in kernel IP layer. It reduces the protocol processing time and provides more flexible content switching rules and integration with load balancing algorithms.

Microsoft Windows2000 Network Load Balancing (NLB) [2] distributes incoming IP traffic to multiple copies of a TCP/IP service, such as a Web server, each running on a host within the cluster. Network Load Balancing transparently partitions the client requests among the hosts and lets the clients access the cluster using one or more "virtual" IP addresses. As enterprise traffic increases, network administrators can simply plug another server into the cluster. With Network Load Balancing, the cluster hosts concurrently respond to different client requests, even multiple requests from the same client. For example, a Web browser may obtain various images within a single Web page from different hosts in a load-balanced cluster. This speeds up processing and shortens the response time to clients. LCS provides more flexible content switching rules.

Linux Virtual Server(LVS) is a load balancing server which is built into Linux kernel [1]. In the LVS server cluster, the front-end of the real servers is a load balancer, also called virtual server, that schedules incoming requests to different real servers and make parallel services of the cluster to appear as a virtual service on a single IP address. A real server can be added or removed transparently in the cluster. The load balancer can also detect the failures of real servers and redirect the request to an active real server.

LVS is a transport level load balancer. It is built in the IP layer of Linux kernel. The incoming request first comes to the load balancer. It is then forwarded to one of the real servers based on the existing load balancing algorithm, using IP addresses and port numbers as the keyword to create a hash entry in the hash table, and save the ip address and port number of assigned real server as the value of the hash table entry. When the following packets of this connection come, load balancer will get the hash entry based on the IP addresses and port numbers, retrieved the ip address and port number of the assigned real server, and redirect the packets to it.

2 Linux-based content switch design

The Linux-based Content switch (LCS) is based on the Linux 2.2-16 kernel and the related LVS package. LVS is a Layer 4 load balancer which forwards the incoming request to the real server by examining the IP address and port number using some existing schedule algorithm. LVS source code is modified and extended with new content switching functions. LCS examines the content of the request, e.g., URL in HTTP header and XML payload, besides its IP address and port number, and forwards the request to the real servers based on the predefined content switching rules. Content switch rules are expressed in term of a set of simple if statements. These if statements include conditions expressed in terms of the fields in the protocol header or pattern in the payload and branch statements describing the routing decisions. Detailed of the content switching rules are presented in Section 4.

2.1 The Architecture and Operations of LCS

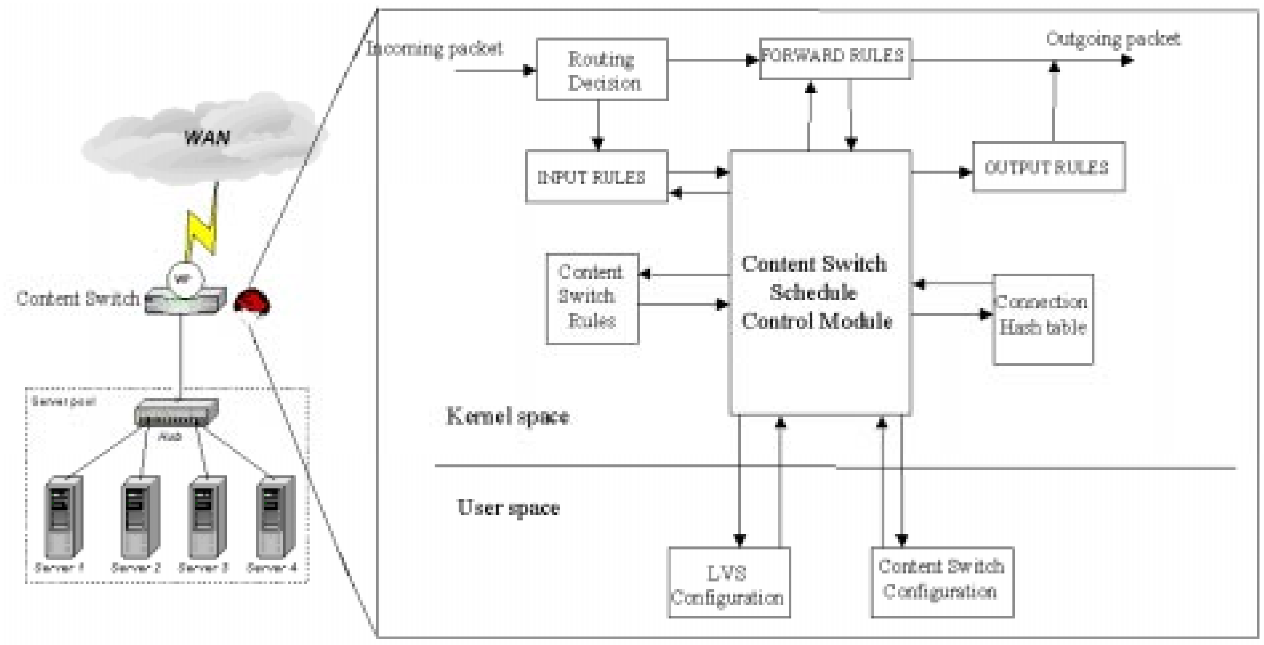

Figure 1. Architecture of the Linux-based Content Switch.

Figure 1 shows the main architecture of LCS. Here *Content Switch Schedule Control* module is the main process of the content switch which is used to manage the packet follow. *Routing Decision, INPUT rules, FORWARD rules and OUTPUT rules* are all original modules in Linux kernel. They are modified to work with Content switch Schedule Control module. *Content switch Rules* module is the redefined rule table. *Content switch schedule control* module will use this information to control the flow of the packets. *Connection Hash table* is used to hash the existing connection to speed up the forwarding process. *LVS Configuration* and *Content Switch Configuration* are user space tools used to define the Content switch server clusters and the Content switch Rules.

Figure 2 shows the main operations of the Content switch. The administrator of the content switch use a content switch (CS) rule editor to create the content switching rules. The content switch rule editor can be a simple text editor or an enhanced editor with GUI and built-in with conflict detection function for checking the conflicts within the rule set. Examples of conflicts include duplication, two rules with the same condition but different routing actions, two rules with conditions that intersect, two rules with conditions that are subset and in improper order. We have built an interative Java-based editor with rule conflict detection. The rule set will be translated into an internal function to be called by the rule matching algorithm download into the kernel as a module. The fields of headers and XML tag sequences mentioned in the rule set will be saved in an array to enable the header content extraction function to extract only those data needed for the execution of the rules.

When an incoming packet arrives, the content switching schedule control module will call the header content extraction function to extract the values of the headers or XML tags mentioned in the rule set, and put them into an array for the rule matching. The content switching schedule control module then called the rule matching algorithm to match the conditions based on the extracted values. If the condition of a rule matches with the header and content of the packet, the routing instructions in the action part of the rule, or the branch statement of the if statement, is carried out.

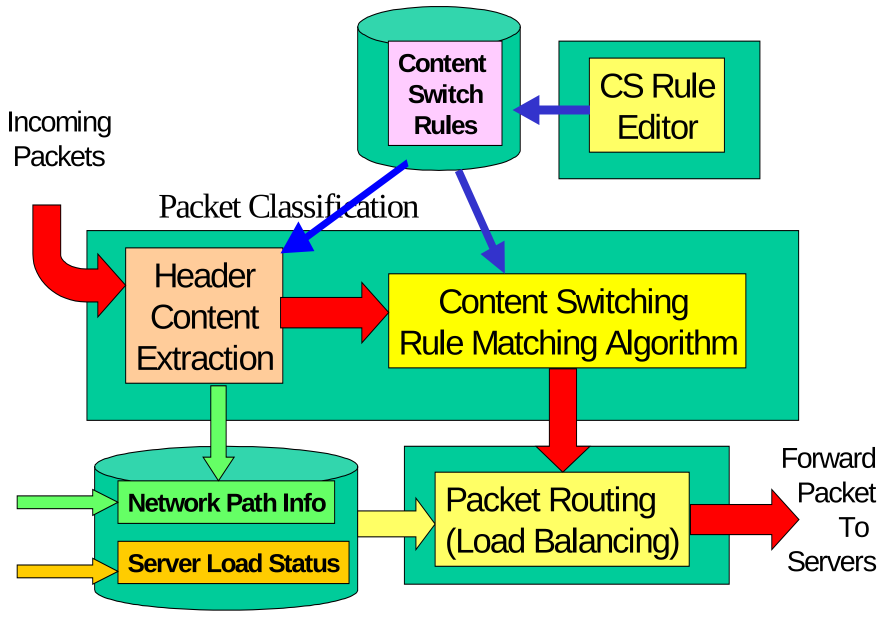
Content Switch Rules
CS Rule Editor
Incoming Packets
Packet Classification
Header Content Extraction
Content Switching Rule Matching Algorithm
Network Path Info
Server Load Status
Packet Routing (Load Balancing)
Forward Packet To Servers

Figure 2. Main Operations of Content Switch.

Content switch rule has a simple syntax is *if (condition) { action1} [else { action2}.]*

One example of a content switch rule is

```
if (match(url,”process.pl”) && xml.purchase/totalAmount > 5000) {
     routeTo(FastServers, NONSTICKY);
}
```

Here only the packet with the url field or the absolute path field of the HTTP request end with process.pl, its payload containing the XML document, and the value of its tag sequence purchase/totalAmount greater than 5000, will be routed to one of FastServers, selected based on some load balancing decision and the connection will be NOSTICKY, i.e., future request from the same connection will be going through the content switch rule matching again. If

For load balancing performance purpose, the network path information module collects network bandwidth information between the client subnet and those of the servers. The server load status collect information from servers about their current pending queues and processing speed. If the routing decision is to load balancing among a set of servers, the load balancing algorithm can retrieve the network path and server load information and make smart decisions.

Figure 3 shows the input output processing of the content switch in IP layer of Linux Network Software.

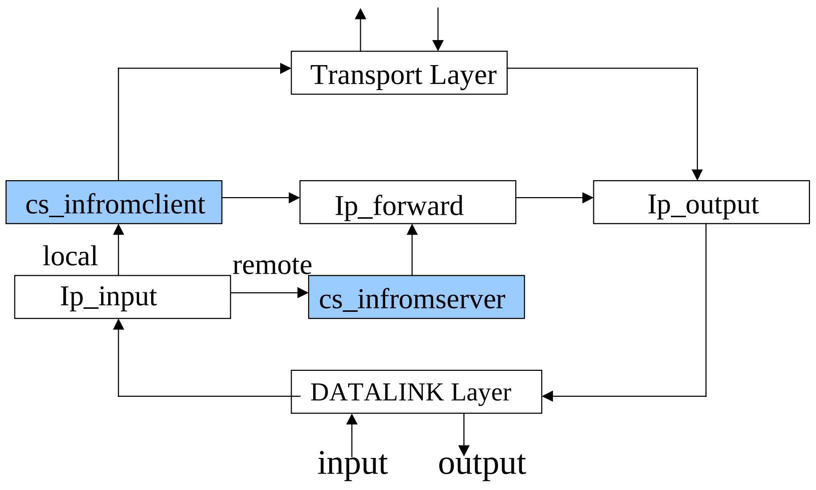

Figure 3. Content switch functions added to the IP layer of the Linux networking software.

cs_infromclient manages the packet from the client to the content switch; **cs_infromserver** handles the packet from the server to the client.

3 TCP Delayed Binding

Many upper layer protocols utilize TCP protocol for reliable orderly message delivery. The TCP connection will be established via a three way handshake and the client will not deliver the upper layer information until the three way handshake is complete. The content switch then selects the real server, establishes the three way handshake with the real server, and serve a bridge that relays packets between the two TCP connections. This is called *TCP delayed binding*.

To hide the complexity behind the clustering system, the client only deals with a virtual IP address, VIP. Therefore all subsequent packets from the client will go through the content switch. It will be more efficient if the return packets from the server to the client can be by-passed the content switch and go directly to the client. However, since the sequence number committed by the content switch and the sequence number committed by the real server are different by the nature of TCP protocol, every subsequent packets between the client and the server require sequence number modification. Due to the changes in the sequence number field, the header checksum also needs to be recomputed. This imposes significant processing overhead in the content switch processing. We will discuss detail message sequences and suggest way for improving the performance.

3.1 The message exchange sequence in TCP Delayed Binding

Because the client established the connection with the content switch, it accepts the sequence number chosen by the content switch and when the packets come from real server to client, content switch must change their sequence numbers to the ones that client expects. Similarly, the packets from client to server are also changed by content switch. By doing the packet rewriting, the Content switch "fools" both the client and real server, and they communicate with each other without knowing the

Content switch is playing the middleman.

Detailed sequence number rewriting process is shown below in Figure3.

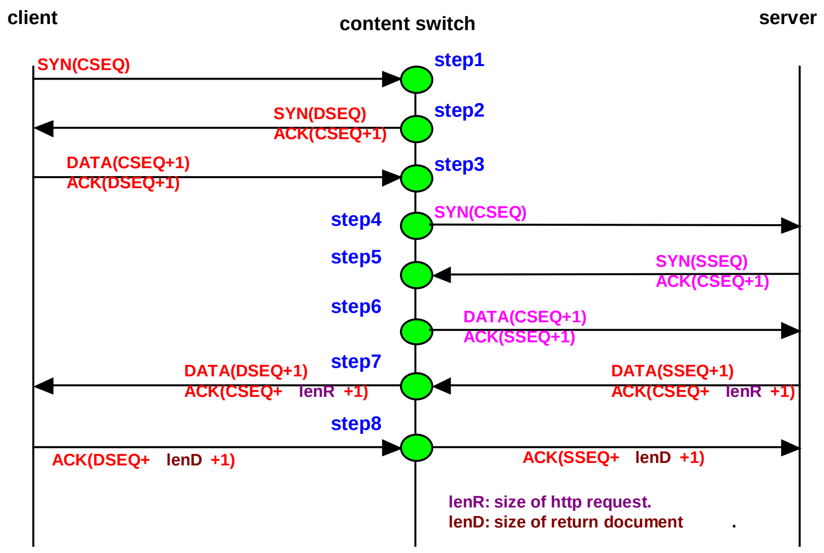

Figure 3. Message Exchange Sequence in TCP Delay Binding.

Step1-Step3: The process is the standard TCP three way handshake between the client and the content switch. Actually there is an ACK message preceding the DATA message sent by the client. The content DATA message contains the upper layer information. At Step 3, the content switch perform the content switch rule matching and select the real server.

Step4: The Content switch forwards the original SYN request to the chosen server.

Step5: The server replies its SYN/ACK including the server's initial sequence number (SSEQ).

Step6: The DATA message is forwarded from the Content switch to the server. The original sequence number is kept, the ACK sequence number is changed from acknowledge number of the content switch (DSEQ+1) to acknowledge number of the server (SSEQ+1).

Step7: For the return data from the server to the client, the sequence number needs to be changed to that associated that of the content switch. For a large document, several packets are needed. Push flags in the TCP header are typically set on the follow up packet, so the client TCP process will deliver immediately to the upper layer process.

Step8: For the ACK packet from the client to the content switch, the ACK sequence number is changed from the one acknowledging the content switch to that acknowledging the server.

Delayed Binding is the major technique used in content switch design. To maintain correct connection between the client and the server, the content switch must adjust the sequence number for each side because of delayed binding. This requests that all the subsequent packets must go through the content switch to get their sequence number changed. As many other existing content switch products, the content switch design presented in this paper uses NAT (Network Address Translation) approach.

TCP delayed binding can be improved by allowing the content switch to guess the real server assignment based on the history information, and IP address and port number in the TCP SYN packet. If the guess is right, all subsequent packets does not require sequence number modification. The sequence number modification process can also be moved from the content switch to a platform closer to the real server or as a process running on the same platform of the real server. It will enable the return document to be routed directly to the client. The content switch processing can also be improved by having several connections pre-established between the content switch and the real servers. This cuts down two steps in the message exchange sequence.

3.2 Handle Multiple Requests in a Keep-Alive Session

Most browsers and web servers support the keep-alive TCP connection. It allows a web browser to request documents referred by the embedded references or hyper links of the original web page through this existing keep alive connection without going through long three way handshake. It is a concern that different requests from the same TCP connection are routed to different web servers based on their content. The challenge here is how the content switch merges the multiple responses from different web servers to the client transparently using the keep alive TCP connection.

Figure 4 shows the situation where different requests from one TCP connection go to different web servers through the content switch.

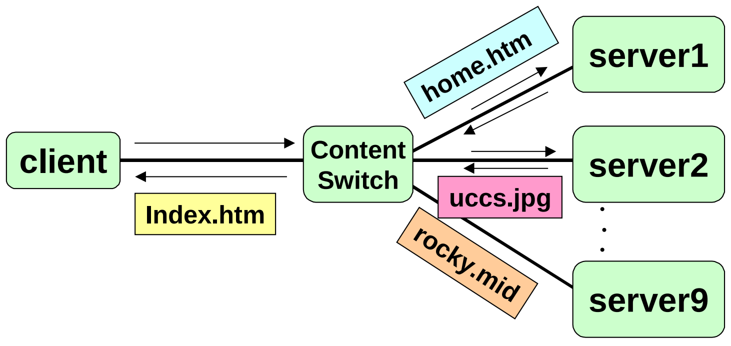

Figure 4. Multiplexing Return Document into a Keep-Alive Connection

If the client sends http requests with one TCP connection to the content switch without waiting for the return document of the previous request, then the content switch can routes these requests to three different web servers based on their contents and it is possible the return documents of those request will arrive at the content switch out of order. The content switch must be able to handle this situation.

The brute force solution will be to discard the early requests. One possible solution is to buffer the responses at the content switch so that they return in the same order as their corresponding requests. The drawback is that it significant increases the memory requirement of the content switch. The other solution is to calculate the size of the return documents and adjust the sequence number accordingly. It avoids the buffer requirement and the later requests will be sent with the starting sequence number that leaves space for those slow return documents. The drawback here is that the content switch needs to have the directory information of the server and how they map the request into the actual path of the file system.

Fortunately, after investigating the usage of keep-alive connections in Netscape browser (version 4.75) and Microsoft IE browser (IE version 5.01), the above keep-alive session multiplexing problem will not appear in those two browsers. Both browsers only send one request at a time over a keep-alive connection. The response must be received before another request can be sent. The Netscape browser creates separate keep-alive TCP connections for each embedded references in a web page. These keep-alive TCP connections are then used in a round robin fashion for the subsequence requests or their embedded references. The IE5.01 we used only open up two keep-alive connections. Several embedded references of a web page may be sent over a single keep-alive connection.

4 The Content Switching Rule Design

Content switching rules are typically expressed in terms of content pattern or conditions that cover the class of packets to be matched, and its associated action. In the existing content switch products, there are two basic ways that rules are specified:

1. The rules are entered using the command line interface. The syntax are typically similar to that of CICSO ACL (access control list)convention [7,8,9].

2. Using a special language to specify the pattern and describe the action of the service. The rule set is then translated and downloaded into the content switch [10].

An example of approach 1 can be seen in Cisco Content Engine 2.20 [7] (CE). For example, Cisco CE can support HTTP and HTTPS proxy server with the rule

*rule no-cache url-regex\.*cgi-bin.**

This rule specifies that the incoming packets with the url matching the regular expression pattern "*cgi-bin*" will not be forwarded to the cache servers. The Foundry ServIron Installation and Configuration Guide [9] provides an excellent collection of rules and web switch configuration examples.

An example of approach 2 is Intel IX-API SDK[10],. It uses network classification language (NCL) to classify the incoming traffic and describe the action of the packet. The rule syntax is presented as

```
Rule <name of the rule> {predicate} {action_method()}
```

The predicate part of the rule is a Boolean expression that describes the conditions. A packet must have the specified action performed. The action part of the rule is a name of an action function to be executed when the predicate is true, and performs some actions upon the incoming packet. For example, `Rule check_src {IP.src==10.10.10.30} {action_A()}`

The meaning of this rule is that if source IP address is 10.10.10.30, then the action function action_A() is executed

4.1 LCS Content Switch Rule

LCS follows an approach similar to Approach 2. The rules are defined using C functions. The syntax of the defined rules is as follows:

RuleLabel: if (condition) { action1} [else { action2}].

```
R1: if (xml.purchase/totalAmount > 52000) { routeTo(server1, STICKY_IP_PORT); }

R2: if (strcmp(xml.purchase/customerName, "CCL") = = 0) {
        routeTo(server2, NONSTICKY); }

R3: if (strcmp(url, "gif$") = = 0) { routeTo(server3, NONSTICKY); }

R4: if (srcip = = “128.198.60.1” && dstip = = “128.198.192.192” &&
        dstport = = 80) { routeTo(LBServerGroup, STICKY_ON_IP_PORT); }

R5: if (match(url, “xmlprocess.pl”)) { goto R6; }

R6: if(xml.purchase/totalAmount > 5000){routeTo(hsServers, NONSTICKY);}
    else {routTo(defaultServers, NONSTICKY); }
```

The rule label allows the use of goto and make referencing easier. We have implemented match() function for regular expression matching and xmlContentExtract() for XML tag sequence extraction in content switching rule module. These rules can be specified by a text editor. The rule set file will be translated by a ruleTranslate program into a data structure contains the XML tag sequences and a function with translated if statements of the rule set. The data structure and the rule function form the basis of a rule module. To update to a new rule set, rmmod is called and the content switch schedule control module will call a default function NO_CS(). The rule module file is then replaced and insmod is called. The content switch is then switched to use the new rule set.

4.2 The Rule action and syntax of the sticky (non-sticky) connections.

In LCS, there are three different options related to the sticky connections. These rules are based on the different content of the packets.

1. Option for sticky connection based on the source IP address.

   Example: *If(source_ip==128.198.192.194)   { routeTo(server2, STICKY_ON_IP);}*

   The condition of this rule is source IP address. The action inside *routeTo()* will assign the real server2 to the connection, and add this connection to the sticky connection database. When the new connection comes, the rule matching process will look for the data entry with the same IP address in sticky database first, if the data entry is found, the connection will be routed to the same server directly without carrying out the rule matching.

2. Option for sticky connection based on source IP address and TCP port number.

   Example: *If((source_ip==128.198.192.194)&&(source_port==9872)) {*

   routeTo(server4, STICKY_ON_IP_PORT);}

   The condition of this rule includes source IP and port number. This rule is for multiple requests in one TCP keep alive connection. The action process will add this entry to the keep alive connection hash table using IP address and port number with the hash key. If the new request arrives from the same connection, the request will be routed to the same server without rule matching.

3. The Rule for non-sticky connection.

   Example: *If (URL==”*jpg”)   { RouteTto(imageServer, NON_STICKY);}*

   This rule specifies the connection to be non-sticky connection. So either the request from the same connection or the new connection all need to process the rule matching to choice the real server.

4.3 Content switch Rule Matching Algorithm

Rule matching algorithm directly affects the performance of the Content switch. It is related to the packet classification techniques [11,12.13] In a layer 4 switching, the switch only exams the IP address and port number of the packet which are in the fixed field. So the rule matching process can be speed up by easily using a hash function. In Content switch, higher layer content information is needed. These information such as URL, HTTP header or XML tag are not from the fixed fields and have different length, so it is hard to build a hash data structure to speed the searching process. In our prototype, we have observed 3 order magnitude of packet processing time difference. It is therefore crucial to improve the performance of the rule matching algorithm, or to emphasize the differential treatment of packets and the flexibility to add other functions such as spam mail filtering.

The rule-matching algorithm should deal with following issues:

1. How to speed up the rule matching process?

2. Are there any specific orders where subsets of rules should be searched? Can some of the rules be skipped in some conditions?

3. The rules may contradict each other. One packet can matche more than one rule. How to detect the conflicts among the rules?

Our LCS prototype currently use the brute forced sequential execution of if-then-else statements representing the rule set.

5 References

[1] "Linux Virtual Server", http://www.linuxvirtualserver.org

[2] "Windows 2000 clustering Technologies: Cluster Service Architecture", Microsoft White Paper, 2000. http://www.microsoft.com.

[3] George Apostolopoulos, David Aubespin, Vinod Peris, Prashant Pradhan, Debanjan Saha, " Design, Implementation and Performance of a Content-Based Switch**",** Proc. Infocom2000, Tel Aviv, March 26 - 30, 2000, http://www.ieee-infocom.org/2000/papers/440.ps

[4] Gregory Yerxa and James Hutchinson, "Web Content Switching", http://www.networkcomputing.com.

[5] M. Leech and D. Koblas. SOCKS Protocol Version 5. IETF Request for Comments 1928, April 1996.

[6] D. Maltz and P. Bhagwat. Application Layer Proxy Performance Using TCP Splice. IBM Technical Report RC-21139, March 1998.

[7] "Release Notes for Cisco Content Engine Software". http://www.cisco.com".

[8] "Network-Based Application Recognition Enhancements". http://www.cisco.com.

[9] "Foundry ServIron Installation and Configuration Guide," May 2000. http://www.foundrynetworks.com/techdocs/SI/index.html

[10] "Intel IXA API SDK 4.0 for Intel PA 100," http://www.intel.com/design/network/products/software/ixapi.htm and http://www.intel.com/design/ixa/whitepapers/ixa.htm#IXA_SDK.

[11] Anja Feldmann S. Muthukrishnan "Tradeoffs for Packet Classification**",** Proceedings of Gigabit Networking Workshop GBN 2000, 26 March 2000 - Tel Aviv, Israel **http://www.comsoc.org/socstr/techcom/tcgn/conference/gbn2000/anja-paper.pdf**

[12] Pankaj Gupta and Nick McKcown, "Packet Classification on Multiple Fields", Proc. Sigcomm, September 1999, Harvard University. http://www-cs-students.Stanford.edu/~pankaj/paps/sig99.pdf

[13] V. Srinivasan S. Suri G. Varghese, "Packet Classification using Tuple Space Search", Proc. Sigcomm99, August 30 - September 3, 1999, Cambridge United States, Pages 135 - 146 http://www.acm.org/pubs/articles/proceedings/comm/316188/p135-srinivasan/p135-srinivasan.pdf