Content Switch Rules and their Conflict Detection

C. Edward Chow, Ganesh Kumar Godavari, Jianhua Xie

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,gkgodava@cs.ucs.edu, jxie@cs.uccs.edu

Tel: 2-1-719-262-3110

Fax: 2-1-719-262-3369

Name of Corresponding Author: C. Edward Chow

Keywords: Content Switch, Policy-based Networking, Rule Conflict Detection

Abstract

Content switch can be configured as load balancer, firewall, spam mail filter, and virus detection and removal system, by specifying a set of rules. In this paper we present our content switch rule design for a Linux-based content switch, and show how they are translated and downloaded into the switch for packet processing. One common problem in specifying rules for content switches or policy-based networks is the conflict detection problem, where rules may contradict or duplicate each other. We explore the algorithm issues related to conflict detection problem and present some performance results. An algorithm for detecting conflict in rules with regular expression matching is presented and its complexity analyzed. An interactive Java-based content switch rule editor was designed for specifying the rules and detecting the conflicts among rules. Its conflict detection algorithm and performance results are also presented.

1 Introduction

With the rapid increase of Internet traffic, we have found load balancing clusters are used for improving the performance of server farm. The content switches or web switches [1,2] are new class of network devices that can be served as a front end for such server cluster. They provide load balancing service by distributing the requests based on the headers and content of the upper layers (3-7). They provide high availability by checking the server status through monitoring messages and re-routing the requests from failed servers to active servers. By rejecting packets based on their source or destination ip addresses and port numbers, they can be configured as firewall. Since they can check the upper layer information, they greatly extend the firewall coverage cases and flexibility. By putting the packets on different queues after examining their headers and content, they can be configured as policy-network devices that control the bandwidth/resource allocation.

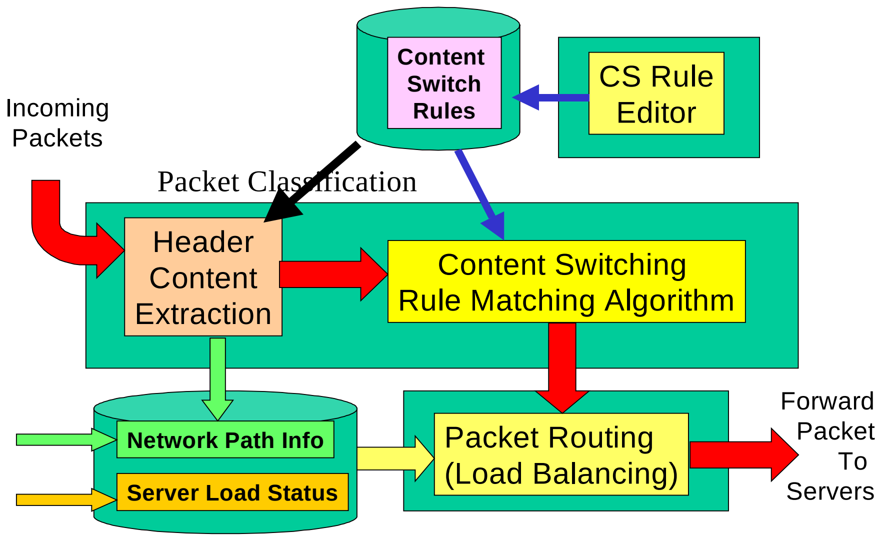

Content Switch Rules
CS Rule Editor
Incoming Packets
Packet Classification
Header Content Extraction
Content Switching Rule Matching Algorithm
Network Path Info
Server Load Status
Packet Routing (Load Balancing)
Forward Packet To Servers

Figure 1. Content Switch Rule and Content Switch

The central mechanism of the content switch is the content switch rule. The rule specifies what class of packets it tries to match and what action should be applied to the packet. Each incoming packet is checked against a set of content switching rules. The set of content switch rules specifies the behavior of the content switch. Figure 1 shows the content switch rule and its relationship with other modules in a 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.

This paper is organized as follows: Section 2 presents the content switch rule design. Section 3 describes how the rules can translated into an internal function to be called by the rule matching algorithm, and downloaded into the switch as a module. Section 4 discusses the conflict detection problems for rules with regular expression. Section 5 shows the design of an interactive Java-based rule editor with built-in rule conflict detection. The performance results of its rule conflict detection algorithm were presented. Section 6 is the conclusion.

2 Content Swtich 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 [3,4,5].

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 [6].

An example of approach 1 can be seen in Cisco Content Engine 2.20 [3] (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 [5] provides an excellent collection of rules and web switch configuration examples.

An example of approach 2 is Intel IX-API SDK[7],. 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

2.1 Content Switch Rule

Our content switch rule 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}].

Here are as a set of legal content switching rules. We use them to explain the content switch rule design.

```
R1: if (match(url, “purchase.pl”) && xml.purchase/totalAmount > 50000) {
      routeTo(highSpeedServers, STICKY_ON_IP_PORT); }

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

R3: if (match(url, "mid$") = = 0) { routeTo(midiServer, NONSTICKY); }

R4: if (match(smtp.from, “spam.com”)) { discard(STICKY_ON_IP); }
R5: if (match(smtp.to, “chow@cs.uccs.edu”)) { routTo(mailServer1, STICKY_ON_IP_PORT); }
```

```
R6: if (match(imap.login "chow")) { routeTo(mailServer1, STICKY_ON_IP_PORT); }

R7: if (inSubnet(srcip, "128.198.60.0/24") && dstip = =  s2ip("128.198.192.192") &&
        dstport = = 80) { routeTo(LBServerGroup, STICKY_ON_IP_PORT); }

R8: if (match(url, "xmlprocess.pl")) { goto R9; }

R9: if (xml.purchase/totalAmount > 5000){routeTo(hsServers, NONSTICKY);}
    else {routeTo(defaultServers, NONSTICKY); }
```

The rule label allows the use of goto for branching, and make referencing of rules easier. The condition of the rule are expressed in terms of Boolean expressions with relational expressions, or Boolean functions such as match(string, regexPatter) and inSubnet(), as basic terms. The relational expressions are expressed in terms of simple variable <relational operator> value form. Here the variable can be fields of headers, substring in the content, XML tag sequences, or a supported string functions such as strcmp() function.

In Rule R1, we have two terms in its condition. One is match(url, "purchase.pl"), which covers all packets that are to be processed by purchase.pl server side CGI script. We have implemented match(string, regexPattern) function for regular expression matching in content switching rule module. It return true when the string contains the regular expression pattern specified in the second parameter. url is a reserved word for the absolute path parameter which appears right after the http request command. The other term is xml.purchase/totalAmount > 50000, which covers the packets that contain an purchase request in XML and contain totalAmount tag under the root tag purchase. The string value wrapped around by the <totalAmount> tag denotes the xml.puchase/totalAmount. Note that here we expect the string value is a legal numeric value and can be converted. If the value is not numeric, an error message will return to the sender.

Rule R1 contains routeTo (highSpeedServers, STICKY_ON_IP_PORT). HighSpeedServers is the name representing a cluster of fast servers. The user can specify what servers constitute the highSpeedServers cluster and what load balancing algorithm is used to select one of them for serving the request. STICY_ON_IP_PORT option specifies the subsequent packet with the same source IP address and source port number will be forwarded to the same selected server. The connection is called a sticky connection and is to be sticky on IP address and port number.

R2: if (strcmp(xml.purchase/customerName, "CCL") = = 0) {

    routeTo(specialServer, NONSTICKY); }

Here we use strcmp() to compare the string value of XML tag sequence purchase/customerName with that of “CCL”. Any packet with purchase/customerName tag value equals to “CCL” will be forwarded to specialServer without regarding its url or other header values.

R3: if (match(url, "mid$") = = 0) { routeTo(midiServer, NONSTICKY); }

Here any HTTP request packet with file extension mid will be routed to midiServer. The $ indicates the mid must appear at the end of the url string. This rule illustrates that we can spread the server load based on the media type of their request documents.

R4: if (match(smtp.from, “spam.com”)) { discard(STICKY_ON_IP); }

Here we demonstrate the rule for configuring the content switch to discard packet from a specific domain and actually any subsequent request from the same node will be discarded. Smtp is the reserved word for the Simple Mail Transfer Protocol. From is a reserved word for the from header field of the email. Any packet with email address containing “spam.com” will covered by this rule. If we would still like the spam node to access others of our services, we can change the option to STICKY_ON_IP_PORT.

R5: if (match(smtp.to, “chow”)) { routTo(mailServer1, STICKY_ON_IP_PORT); }

R6: if (match(imap.login “chow”)) { routeTo(mailServer1, STICKY_ON_IP_PORT); }

Here we demonstrate how to route the mails of a user to a specific mail server. IMAP protocol is used to retrieve emails. Login is a reserved word for the login field in the IMAP request.

With a slight modification, the following six rules will spread the load of mail services to three servers.

R5a: if (match(smtp.to, “^[A-I,a-i]”)) { routTo(mailServer1, STICKY_ON_IP_PORT); }

R6a: if (match(imap.login, “^[A-I,a-i]”)) { routeTo(mailServer1, STICKY_ON_IP_PORT); }

R5j: if (match(smtp.to, “^[J-R,j-r]”)) { routTo(mailServer2, STICKY_ON_IP_PORT); }

R6j: if (match(imap.login, “^[J-R,j-r]”)) { routeTo(mailServer2, STICKY_ON_IP_PORT); }

R5s: if (match(smtp.to, “^[S-Z,s-z]”)) { routTo(mailServer3, STICKY_ON_IP_PORT); }

R6s: if (match(imap.login, “^[S-Z,s-z]”)) { routeTo(mailServer3, STICKY_ON_IP_PORT); }

R7: if (inSubnet(srcip, “128.198.60.0/24”) && dstip = = s2ip(“128.198.192.192”) &&
    dstport = = 80) { routeTo(LBServerGroup, STICKY_ON_IP_PORT); }

Srcip, dstip, srcport, dstport are reserved words representing the fields in the headers of protocols. Since we want to cover packets from a subnet, an Boolean function inSubnet() for checking if the net address of the IP address of the packet falls is a specific one. A s2ip() function is used to convert IP address in dot notation to the equivalent 32 bit value.

R8: if (match(url, “xmlprocess.pl”)) { goto R9; }

R9: if (xml.purchase/totalAmount > 5000){routeTo(hsServers, NONSTICKY);}
  else {routeTo(defaultServers, NONSTICKY); }

Rule R8 demonstrates the use of goto. Rule R9 demonstrate the use of else branches.

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

In our Linux-based Content Switch prototype (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.

3 Translation and Update of the Content Switch Rule Set

These rules can be specified by a text editor. The rule set file will then 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.

3.1 Rule Translation

A program called ruleTranslate takes as input a file containing the rule set created by the rule editor and convert the rule set into a function to be called by the content switch module.

For the following rule set,

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

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

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

R4: if (inSubnet(srcip, “128.198.60.0/24”) && dstip = = s2ip(“128.198.192.192”) &&
        dstport = = 80) { routeTo(LBServerGroup, STICKY_ON_IP_PORT); }

R5: if (strcmp(url, “xmlprocess.pl”)= =0) { goto R6; }

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

generates the following rule.c flie as output

```
#include ...  some header files include

char *xmlTagSequence[] = {

            "purchase/totalAmount",

            "purchase/customerName",

};

int noOfxmlTagSequences = n;
/* here n will be 2, the number of xml tag sequences found in the rule set */

char *xmlTagValue[n];    /* this char * array will be filled with string
```

```
containing the value of the tag in a packet with XML content */

int ruleMatch() {

xmlContentExtract(xmlTagSequence, xmlTagValue);
/* this function performs xml parsing, matches the tag sequences in the  xmlTagSequence array,  and assign the string
value of the tag to xmlTagValue array.

R1: if (atoi(xmlTagValue[0]) > 52000){
      routeTo(server1, STICKY_ON_IP_PORT); return(0);}

R2: if (strcmp(xmlTagValue[1], "CCL")==0) {
      routeTo(server2, NONSTICKY); return(0); }

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

R4: if (inSubnet(srcip, “128.198.60.0/24”) &&
       dstip = = s2ip(“128.198.192.192”) &&
       dstport = = 80) {
      routeTo(LBServerGroup, STICKY); return(0)}

R5: if (strcmp(url, “xmlprocess.pl”)= =0) { goto R6; }

R6: if(atoi(xml.purchase/totalAmount)> 5000){
      routeTo(hsServers, STICKY); return(0);
   } else {
      routTo(defaultServers, STICKY); return(0); }
```

ruleTranslate basically scans through the rule set, and implements following pseudo code

```
for each xml tag sequence, xml.<tagseq> it encounters,
```

. append a string <tagseq> to the xmlTagSequence[] in rule.c file

. change the xml.<tagseq> in the if statement to xmlTagValue[n] and

If relational operator should be interpreted as numeric operator due to the numeric value,

then if statement becomes

```
if (atoi(xmlTagValue[n]) rop value) { action}
```

If the value is a string and rop is = =.

if statement becomes

```
if (strcmp(xmlTagValue[n],value) ==0) {action }
```

. add return at the end of action statement, unless the action statement end with goto <Rx> statement.

4 *Conflict Detection Problem for Rules with Regular Expressions*

When we define the content switch rules, we may unintentionally define some conflict rules. For example, if there are two rules in the rule set:

if (match(url, "^*.gif$) {routeTo(Server1); return;}

if (match(url, "^a.gif$) {routeTo(server2); return;}*

The second rule will never be executed, because the second regular express is a subset of the first one. A useful content switch rule editor should detect this kind of errors and prompt the administrator with instructive messages. To detect this kind of conflict, we need to know the relationship between two regular expressions. There are five potential relations between two regular expressions, say, reg1 and reg2:

reg1 = reg2	*reg1 equals to reg2*

reg1 $\supset$ *reg2*	*reg1 contains reg2*
reg1 $\subset$ *reg2*	*reg2 contains reg1*
(reg1 $\cap$ *reg2)* $\neq \phi$	*reg1 intersects reg2*
(reg1 $\cap$ *reg2)* $= \phi$	*reg1 disjoins reg2*

For the use of content switch rule conflict detection, we need only detect the first three cases. Because “x contains y” is symmetric to “y contains x” and “x equals y” is the same as “x contains y and y contains x”, so we need only design an algorithm to detect if x includes y, then all the problems solved [7].

4.1 Regular Expression Definition

In our discussion, we restrict the meta-characters to

*	matches zero or more characters or meta-characters
[x-y]	matches any single character in the range of x to y inclusive
?	matches any single character

We rule out the two markers (^, the beginning marker, and $, the end marker) because this definition has the same expression power as that with the markers and it can simplify our algorithm. For those who are used to regular expressions with markers, we can use the following simple algorithm to transfer between the two definitions.

Regular expression with marker to regular expression without marker:

```
if (reg begin with ^) strip ^;
else reg = ‘*’+reg;
if (reg ends with $) strip $;
```

else reg = reg + '';*

The following is a truth table of regular expressions. The regular expressions in the 2nd and 3rd columns are equivalent and so do those in 4th and 5th columns.

	with markers	*Without markers*	*with markers*	*without markers*
	^abc.gif$	*Abc.gif*	*abc.gif*	**abc.gif**
abc.gif	*True*	*True*	*True*	*True*
dabc.gif1	*False*	*False*	*True*	*True*

4.2 Algorithm for detecting "x contains y"

Definition 1: *The relation "assignable to" between two characters x and y is defined as:*

c assignable to c	*for any character or meta-character c*
c assignable to ?	*for any character or range character([x-y]) c*
*c assignable to **	*for any character or meta-character or empty string c*
c assignable to [x-y]	*for any character or the range character within the range from x to y*

If x is assignable to y, we write it as: $x \rightarrow y$.

Definition 2: *Single size character: all characters and meta-characters except **

With the definitions above, we have the following algorithm.

4.3 Algorithm: Detecting if one regular expression contains another

Input: *regular expression reg1 and reg2*

Output: *true if reg1⊃reg2, false otherwise*

Method:

1. *p1=0, p2=length of reg1, q1=0, q2=length of reg2*
2. *if q1equals q2 and p1 equals p2, finish and return true.*
3. *If reg1[p1] is a single size character and reg2[q1] → reg1[p1],*

   then let p1 = p1+1 and q1 = q1+1, goto step 2
4. *for t from q1+1 to q2, execute the algorithm with p1+1, p2, t, q2*

The pseudo-code of the algorithm is as following:

```
contains(string reg1, string reg2, int p1, int p2,int  q1,int  q2){
   if (q1 == q2) {
        if (p1 == p2)  return true;
       else return false
   }
   if (X[p1] is a single size character) {
       if (! Y[q1] → X[p1])  return false;
       else return contains(p1+1, p2, q1+1, q2);
   }
```

```
    else {
        for(t=q1; t<=q2; t++)
            if (contains(p1+1, p2, t, q2)) return true;
        return false;
    }
}
```

4.4 Complexity analysis

In the above algorithm, the recursion occurs only when the character to be compared is asterisk. So in the worst situation, i.e., half of the characters in the reg1 are asterisk, the complexity is $o(n^n)$, where n is the length of the regular expression. But in the use of the content switch rule editing, the number of asterisks in a regular expression is quite little, normally less than 3, and the length of regular expression is also very limited, normally less than 15 characters. For example, if there are 2 asterisks in the regular expression, then the complexity is $o(n^3)$, which is good enough for where n is less than 15.

5 Java-based Content Switch Rule Editor

We have implemented an interactive Java-based Content Switch Rule Editor, RuleEdit, for editing the content switch rule set and detecting the conflicts between the rule being specified and the existing rule set. Figure 1 shows the screendump of RuleEdit. It can open a text file of existing rule set, and load the rule set into its internal structure. It currently only implements conditions with one basic term. The user can enter the variable, the relational operator, and the value of a relational expression, or a Boolean expression. On the right hand side is a set of buttons for inserting or appending the rule to the rule set based on the line number, and for deleting certain rule. The text window below shows the rules in the existing rule set.

The operator List box contains various options as shown in Fig 3. The logical operators are used for numeric conditions statements. When the contents of term2 are not numeric the rule editor automatically converts the condition into a strcmp statement. The options “Contains” and “!contains”are used for handling regular expressions, e.g.

```
    if (term1 contains term2) {sequence of actions to be performed}
```

Here term2 can be a regular expression like “*.gif”.

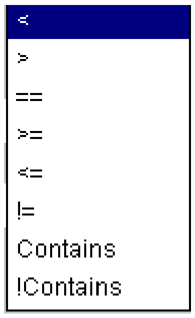
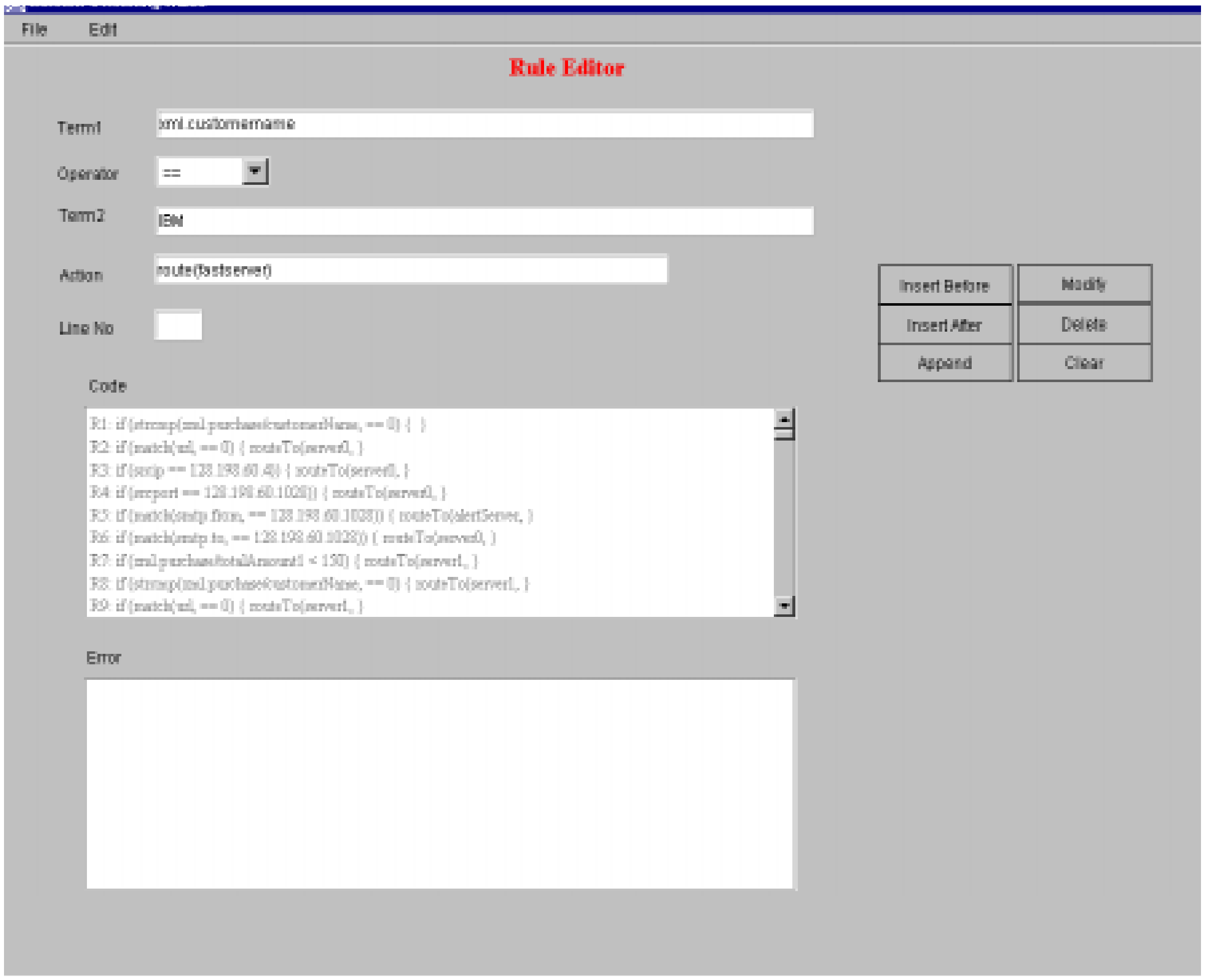


Figure 3: options in Operator ListBox



Term1: xml.customername

Operator: ==

Term2: IBM

Action: route(fastserver)

```
R1: if (strcmp(xml.purchase/customerName, == 0) {  }
R2: if (match(url, == 0) { routeTo(server0, }
R3: if (srcip == 128.198.60.45) { routeTo(server0, }
R4: if (srcport == 128.198.60.1028)) { routeTo(server0, }
R5: if (match(smtp.from, == 128.198.60.1028)) { routeTo(alertServer, }
R6: if (match(smtp.to, == 128.198.60.1028)) { routeTo(server0, }
R7: if (xml.purchase/totalAmount1 < 130) { routeTo(server1, }
R8: if (strcmp(xml.purchase/customerName, == 0) { routeTo(server1, }
R9: if (match(url, == 0) { routeTo(server1, }
```

Figure 2. Java-based Interactive Content Switch Rule Editor

When the rules are entered in the rule editor. The rule editor outputs a flag whenever a potential conflict is detected. Potential conflicts can be

1) Duplication of the condition i.e. for same condition specifying same/different actions.

2) Numerical comparison errors, these are easily committed and difficult to detect manually.

For example,

Rule111: (xml.purchase/totalAmount > 5000) { routeTo(miniServer, NONSTICKY);}

Rule122: if (xml.purchase/totalAmount > 20000) { routeTo(crayServer, NOSTICKY);}

When rule matching occurs and xml.purchase/totalAmount = 30000,

Rule111 will be executed first rather than Rule122, which may not be the intention of the user.

Similar numerical comparison conflict can occur with “<“ opertator. The Rule Editor will help detect these potential conflicts. Duplication Errors can easily be detected by checking the condition with the existing conditions Numeric Comparison Errors are detected using the following algorithm.

5.1 Conflict Rule Detection Algorithm

When a new rule is being entered,

1) Check the logical operator, if it is of not “>”, “<” goto step 6.

2) Check with the existing rules whether the logical operator and term1 match, if no match is found goto step 6.

3) If the logical operator is “>”,

   Check whether match.term2 < term2 and match.ruleposition < ruleposition If true flag the user of potential conflict and goto step 6.

4) If the logical operator is “<” ,

   Check whether match.term2 > term2 and match.ruleposition < ruleposition

   If true flag the user of potential conflict and goto step 6.

5) Add rule to existing rules.

6) Exit.

5.2 Performance Results of the Rule Conflict Detection Algorithm

To understand the performance, we use a Perl script to create rule set of varying sizes and load them in RuleEdit and can compute the time it takes for the RuleEdit to finish the conflict detection. Figure 4 shows the performance result. It takes about 1.6 minutes to finish the comparison between one rule and an existing 8000 rules. Below 2000 rules, the user did not see noticeable delay.

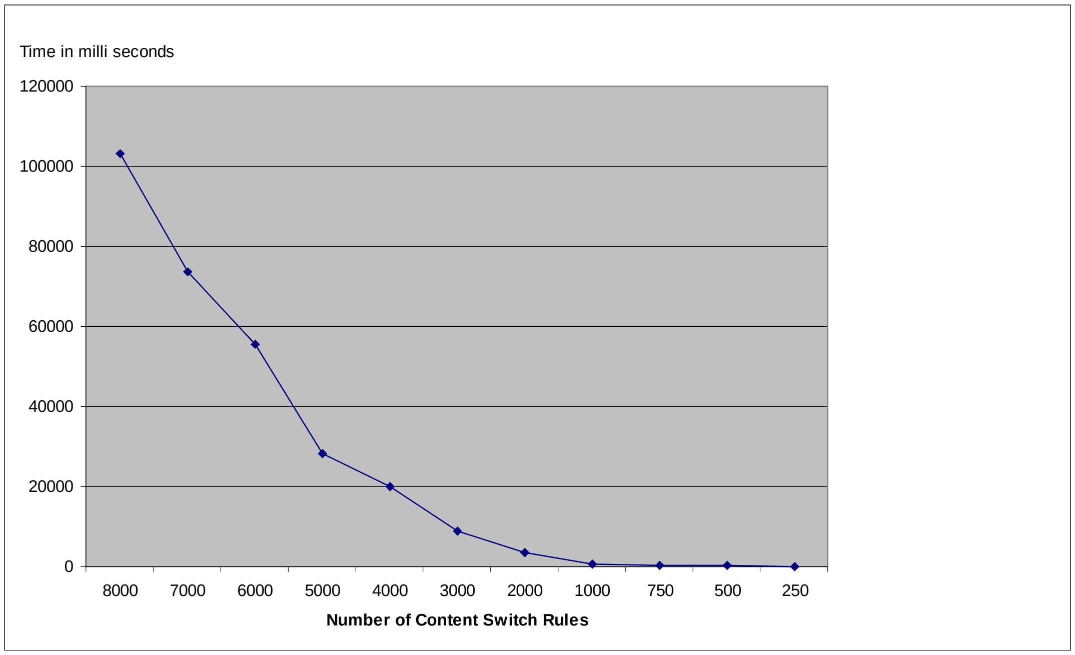


Figure 4. Time Performance of Rule Conflict Dection algorithm.

5.3 Features and Current Limitations of RuleEdit

The RuleEdit can inform the end user of a potential conflict in if statements, which are tiring to debug when written manually. It automatically converts string data type into strcmp format, the native form of c string handling. It also handles regular expression comparisons.

Currently it handles only one terms in the condition. We are working on the multiple term cases. The conflict detection algorithm can be improved.

6 Conclusion

We have presented a flexible content switch rule design for specifying the operation of content switches and demonstrate rules for various configurations. The detailed steps for translating and downloading the rule set to be executed by the content switch are presented. The rule conflict detection problem and their algorithms are presented together with the performance and design of an interactive Java-based Rule Editor.

7 References

[1] 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

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

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

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

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

[6] "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.

[7] Introduction to Language and the Theory of Computation, by John C. Martin, published by McMraw-Hill, 1997.