Design and Implementation of a Linux-based Content switch

C. Edward Chow, Dept. of CS, Univ. of Colorado at Colorado Springs, Colorado Springs, CO 80933-7150, USA, chow@cs.uccs.edu
Weihong Wang, Dept. of CS, Univ. of Colorado at Colorado Springs, Colorado Springs, CO 80933-7150, USA, wwang@cs.uccs.edu

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 systems. 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. With IMAP/POP header extraction, it can be configured as a load balancer for email retrieval systems.

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

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. NLB transparently partitions the client requests among the hosts and lets the clients access the cluster using one or more "virtual" IP addresses. With NLB, the cluster hosts concurrently respond to different client requests, even multiple requests from the same client.

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.

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.

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 predefined 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.

LCS uses Linux Network Address Translation (NAT) approach to achieve content switch. When an incoming packet arrives at IP layer, *Routing Decision* function is called to check if the packet is destined to local or remote host. If the packet is for the local host, *INPUT RULES* function is called to deliver the packet to *Content Switch Schedule*

Control module. Otherwise *FORWARD RULES* function is called to pass the packet to *Content Switch Schedule Control* module. *Content Switch Schedule Control* module will check *Connection Hash Table* to see if it belongs to an existing connection. If the packet is a new request, *Content Switch Schedule Control* module will extract the header/content of the request and apply *Content Switch Rules* on it to choose a real server for this request. Also a new hash table for this connection is created, and then *FORWARD RULES* function is called to forward this request to the chosen server. If the packet is from an existing connection, *Content Switch Schedule Control* module will call *FORWARD RULES* function to forward the packet based on the information in hash table.

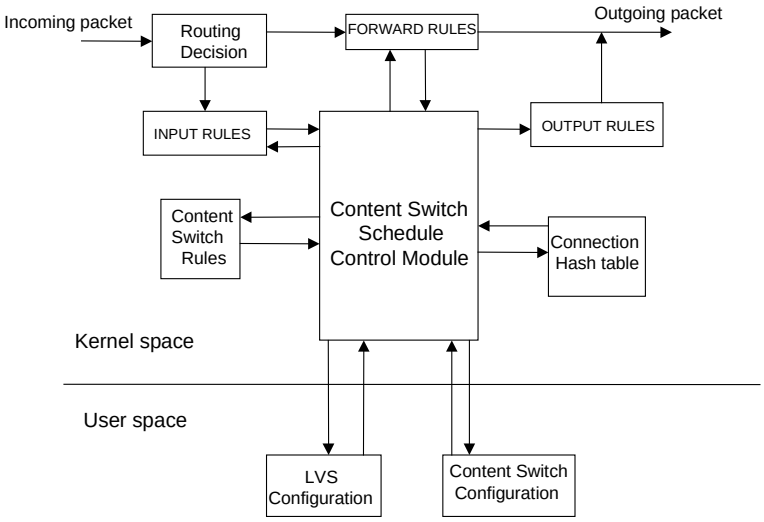


Figure 1 LCS Architecture

Figure 2 shows the input output processing of the content switch in IP layer of Linux Network Software. **cs_infromclient** manages the packet from the client to the content switch; **cs_infromserver** handles the packet from the server back to the client.

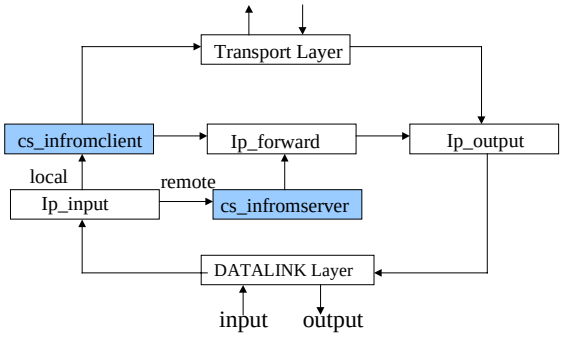


Figure 2 Content switch functions added to IP layer in Linux network software

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*.

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.

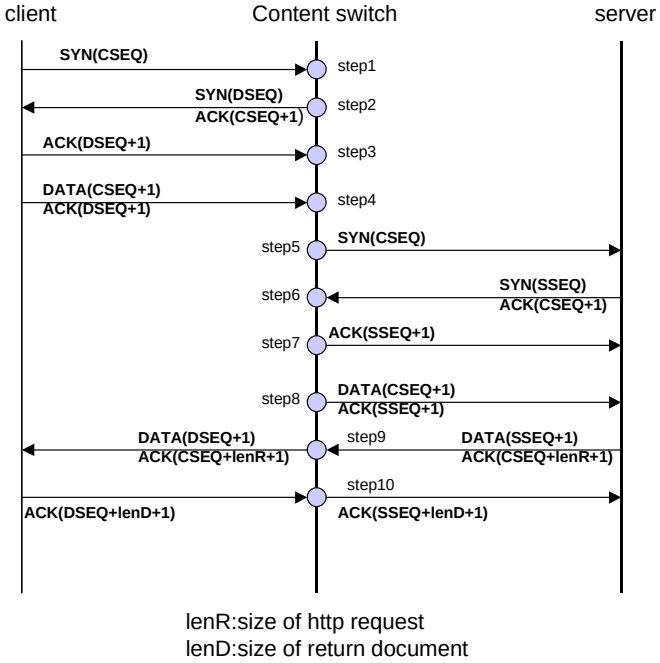


Figure 3. Message Exchange Sequence in TCP Delay Binding.

Step1-Step3: The process is the standard TCP three way handshake between the client and the content switch. The client and content switch commit their initial sequence number as CSEQ and DSEQ.

Step4: The client sends its request data to the content switch. The content switch chooses a real server based on the request data to server this request. The request data may contain more than one IP packets, so the content switch need to get all the IP packets before processing rule matching algorithm.

Step5-Step7: The Content switch establishes TCP connection with the real server. The content switch forwards the SYN request from the client to the server using its original initial sequence number CSEQ, the server commits its own initial sequence number SSEQ.

Step8: 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).

Step9: 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.

Step10: 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. 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.

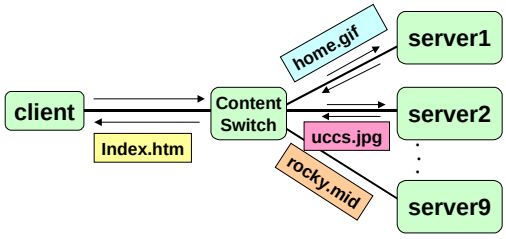


Figure 4. Multiplexing Return Document into a Keep-Alive Connection

If the client sends http requests within one TCP connection to the content switch, then the content switch can route 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 of the later request 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.

We investigated the usage of keep-alive connections in Netscape browser (version 4.75) and Microsoft IE browser (IE version 5.01). Both browsers set up multiple TCP connections for the embedded references in a web page, and each TCP connection contains one or more HTTP requests. The browser may send out these multiple HTTP requests in one TCP keep-alive connection at the same time, without waiting for the response data of the earlier requests. So it is necessary for the content switch to handle the responses from the different real servers.

4 The Content Switching Rule Design

4.1 LCS Content Switch Rule

LCS rules are defined using C functions. The syntax of the rules is as follow:

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

Examples:

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_IP); }

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 makes referencing easier. We have implemented match() function for regular expression matching and xmlContentExtract() for XML tag sequence extraction in content switching rule module. The rule is designed as a kernel dynamic module. So it can be edited at kernel running time without recompiling kernel. To update to a new rule set, “rmmod” is called to removed the current rule set, and the content switch schedule control module will call a default function NO_CS() to schedule the requests using round robin algorithm (or weighted connection, least connection…). The content switch uses “insmod” to insert the new rule module.

4.2 The Rule actions of the sticky (non-sticky) connections.

In LCS, there are three different options related to the sticky connections. These options are *STICKY_IP_PORT*, *STICKY_IP* and *NONSTICKY*. With the option *STICKY_IP_PORT*, if the condition is true, all the following packets with the same IP addresses and TCP port numbers will be routed to the same server directly without carrying out the rule matching process. This option will route all the requests in one TCP keep-alive connection to the same server. And the option *STICKY_IP* will stick all the packets with the same IP addresses to the same server. This option will route all the request from the same client to the same server. The option *NONSTICKY*, 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.

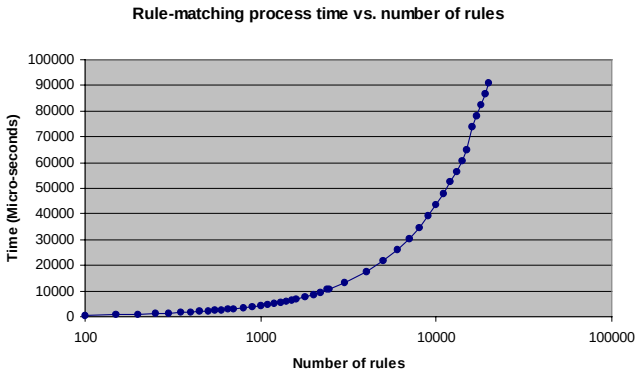
4.3 Content switch Rule Matching Algorithm

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. Also the order of the rules and rule conflicting will affect the content switch performance. Our LCS prototype currently use the brute forced sequential execution of if-then-else statements representing the rule set.

5 LCS Performance Result

Since the content switch examines the content of the request before it forwards the request data to the real server, it may have more overhead than the layer 4 load balancing method. For all the tests we have performed, we would like to find out what parameters affect the performance of the content switch. To set up our LCS testbed, HP Vectra workstation with 240 MHz Pentium Pro Processor and 128 MB memory is used as the content switch. Four real servers are connected with the content switch.

Figure 5a shows the measured response time for the rule matching process inside the content switch kernel when a different number of rules are used. As shown in Figure 5a, the more rules the content switch has, the longer the process time. This is because the larger number of rules, it will take longer time to perform rule matching. The more efficient searching algorithms will help to improve the rule matching performance.



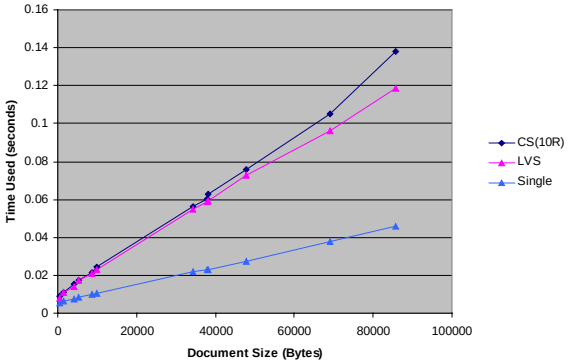


Figure 5a. Rule process time vs. number of rules. Figure 5b. Rule process time vs. XML file size

Figure 5b compares the impact of XML fi processing overheads among LCS, LVS, and a single web server. Here we are interested in finding the impact of TCP delay binding overhead on LCS, and therefore we did not configure LCS to match any rule. It indicates the TCP delay binding incurred very small amount of additional processing compared to the overall processing time. The difference between LVS and that of a single server is the redirect routing processing time.

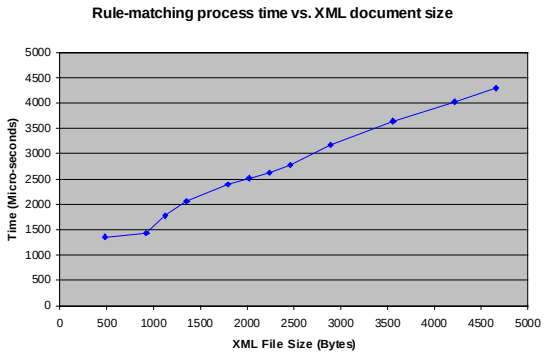


Figure 5c. Rule process time vs. XML file size.

Figure 5c shows the impact of XML document size on LCS processing time. If the request contains an XML document and the rules contain conditions related to the XML tag value, the process time varies with the XML

document size. If rules are defined to choose the real server based on XML tag values, the rule-matching process will need to parse an XML request to find the XML tag value. The XML parsing process uses recursive algorithm, so the process time increases dramatically with the size of the XML document.

6 Conclusion

We have presented the design and implementation of a Linux LVS-based content switch. The impacts of the number of rules, the document size, and the TCP delay binding on the LCS performance are analyzed. The rule set is implemented as a set of simple if statements with labels and is compiled into a Linux kernel module. The rule module can be dynamically loaded into the LCS kernel. We also studied the impact of multiple requests of a keep-alive connection on the content switch processing and analyzed as a set of solutions. The software provides a foundation for studying the network and protocol related issues in content switches and cluster systems.

7 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