Intelligent Content-Based Routing for Enhanced Internet Services

By

Suresh Jayaraman

A Thesis

Submitted to the Faculty of Graduate Studies at University of Manitoba
in Partial fulfillment of the Requirements
for the Degree of

MASTER OF SCIENCE

Department of Computer Science
University of Manitoba
Winnipeg, Manitoba

0-612-76965-8

INTELLIGENT CONTENT-BASED ROUTING FOR ENHANCED INTERNET SERVICES

BY

Suresh Jayaraman

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University

of Manitoba in partial fulfillment of the requirements of the degree

of

Master of Science

To Mom and Dad with love...

Abstract

An Intelligent content-based router should be designed to analyze data and find a suitable server for processing a client's request quickly and efficiently. Current content routers examine only the HTTP based URL request and routes the request to the "*best*" server for processing. These routers fail to examine different types of TCP-based user requests. The content router developed in this thesis examines all type of TCP-based requests. The content router is a core router that simply forwards packets to the edge routers for delivery after performing its content based processing. This router can be replicated to achieve higher performance in large networks. Moreover, by adopting a formal design approach, which is subject to mechanical evaluation using the Z-EVES tool, the correctness of the design is ascertained.

The objectives of this thesis are to:

- Provide an object-oriented design of an intelligent content-based router (a network device that routes packets based on their contents) for e-commerce applications using the UML paradigm.
- Model the design using the Z specification language to guarantee correctness and prove the reliability of the design. In particular, Z notation will provide the capability to capture both dynamic and static features and operations of the proposed content-based router.
- Provide a prototype implementation of the design as a proof of concept.

Acknowledgements

First I would like to thank my supervisor Dr. Sylvanus Ehikioya for generating the initial ideas for this thesis and for guiding me through out my thesis and being there when I needed him.

My special thanks to Dr. Jose A. Rueda of TR*Labs* for providing me with a great research environment. Thank you also for providing me moral and technical support through out my research period at TR*Labs*.

I would also like to thank Dr. Muthucumaru Maheswaran for making his lab (Advanced Network Research Lab) available for implementation of the concepts in this thesis and providing initial network programming assistance.

I also thank Dr. Peter Graham for providing useful comments and excellent editing of the initial thesis proposal.

A special thanks to Dr. Ruppa Thulasiraman for attending all my mock presentations and providing a feedback for the work I had done. I also thank him for reading my thesis and providing me his comments.

I would also like to thank the members of my thesis committee Dr. Sylvanus Ehikioya, Dr. Muthucumaru Maheswaran and Dr. Jose A. Rueda for accepting to examine this thesis.

Finally, a thanks to TR*Labs* as an organization for providing the necessary financial support.

I thank everyone at the offices of the Department of Computer Science at the U of M, particularly Ms. Lynne and Ms. Susan for all the administrative support they provided during my study.

I thank my friends (Chintu, Rajesh M, Rajesh R, Arvind V, Arvind S, Shony, Kumaran, Ganesh and Gayathri) for their friendship and support in completing this thesis. I also thank Deepa for proofreading my thesis module by module. A big "*thank you*" to everyone for being good friends and supporting me in many different ways.

This section would be incomplete if I do not thank my family. I thank my brother (Prasad), mom and dad for all the support, patience, perseverance, affection, confidence and prayers without which I would not have made it this far.

I finally thank GOD for everything he has given me.

Contents

List of Figures

Chapter 1

Introduction

As the number of Internet users and sites continues to increase rapidly, demands on network transmission bandwidths keep growing and the networks connected to the Internet often become heavily loaded. As a result, locating and accessing relevant information in large distributed systems is sometimes difficult and slow. This limits the practical applicability of wide area distributed systems. To address this problem, efforts must be made to use the available bandwidth more effectively.

1.1 Motivation

Transmission links alone do not make a network. Other components such as switches, routers, etc. (and the software that run them) are also parts of a network. Of particular interest to this thesis is the router. A router is a device that is used for forwarding packets from one network to another. Every packet must pass through, typically, many routers. The increase in demand for network bandwidth also places a huge demand on network routers [GLM98] and router saturation has an impact on the performance of many distributed computing applications, including Electronic Commerce. One way to overcome this problem is to develop innovative new router architectures that do routing based on packet content in an effort to minimize wasted bandwidth. The design and prototyping of such router architecture is the focus of this thesis.

Current routers do not examine packet data; rather they blindly forward packets based solely on their destination address (which is contained in each packet header). While this minimizes router processing and thereby increases potential router throughput, it also limits routing flexibility. With content-based routing, it is possible to optimize routing based on application characteristics. This is not possible with conventional routers. Such optimizations can be applied to increase the efficiency of bandwidth use in the Internet.

1.2 Goal of the Study

The main goal of the thesis is to develop an intelligent content-based router that examines the data in a packet, and then routes the packet to a destination where it can be most quickly, cheaply, and efficiently processed. Before forwarding packets to their respective destinations, the router examines the data in each packet and based on the data itself as well as the network state, will determine a suitable destination address that can optimize processing of the packet. Thus, a packet may be redirected to a different destination address than was originally specified. This can be used to improve network bandwidth utilization by replicating network services (e.g. web servers) and doing in-network selection of the "optimal" replica to use for a particular packet/request.

The routing mechanism proposed in this thesis uses a set of metrics (including such network state information as the cost, speed, and traffic over various links as well as server proximity and workload) in making decisions about which destination to forward packets to. The job size is not considered as a metric in this thesis. The transmission cost of each packet depends upon various factors like network bandwidth, general health of the network (i.e.) status of participating servers and size of each packet. This routing mechanism, which we refer to, as Intelligent Content-based Routing will also be useful for any distributed system which, can offer the required data at different network locations. It is also extendable to other optimizations based on packet content. Providing fast response, scalability, and consistent operational behaviour will be the key challenges in my router design.

1.3 Contributions

This thesis proposes a new design for an intelligent content-based router. This design addresses the various problems, such as network traffic, load on different servers, replication of data on different servers and implements a new solution to overcome these problems.

The main contributions of this thesis are:

- Provides a new architecture for an Intelligent Content-Based Router.
- Provides various network designs where the newly designed content router can be used effectively and efficiently.
- Provides an Object Model for the newly designed content-based router.
- Provides a Formal Specification of Intelligent Content-based router using the Z specification language to prove the correctness and reliability of the design.
- Provides a prototype implementation of the proposed design.

The content router proposed in this thesis consists of three major components embedded within the content router. They are the Packet Inspector, the Resource Inspector and the Scheduler Unit. We developed new algorithms for implementing the Resource Inspector and the Scheduler. The complete details of each component are discussed in Chapter 3 (*System Design*). In this thesis, we utilize much of the application information from the participating servers and from their status. The designed router is capable of finding the load and resource information on each sever dynamically and provides the collected information to other components of the router in order to process the user's request. It must be noted that the implementation of this thesis has some assumptions, which is described in Chapter 5 (*Object Model and Functionality*). Finally a user scenario is provided with some screenshots to explain how the newly designed content router can be utilized in real time E-Commerce applications.

1.4 Benefits

The proposed architecture provides a verified, content-based routing technology that can be used to build application-specific intelligent software routing environments. Such environments can be exploited to create more efficient geographically distributed databases and other similar applications [E00].

Intelligent content-based routing can provide the following key services: (i) content-based routing, (ii) traffic optimization, (iii) economically scalable services that provide appropriate response to varying processing loads, and (iv) the ability to track content requests and respond with appropriate content.

Of particular current interest, content-based routing can be used to deliver optimized Web response time, which is critical to the success of e-commerce applications. That is, content routing enables the transparent selection of the best site and server for processing/delivering the requested content thereby, providing an enabling technology for more efficient distributed Web site processing.

It is expected that this thesis will also lead to other application-level content routing applications and, potentially, to the development of a hardware intelligent content - based router.

1.5 General Assumptions

To develop an efficient Intelligent Content-Based router, our architecture makes the following assumptions.

Cost of Transmission: In this thesis transmission cost was not considered while designing the content router. We assumed that the transmission cost is minimal and equal for every packet that is being processed. But, in general, in order to consider transmission cost various factors like packet size and network bandwidth have to be considered also.

Nature of applications: Since this thesis focuses on issues in E-Commerce, our architecture was exclusively designed for different applications of E-Commerce. Our architecture is made flexible to support other distributed applications.

Job size: The size of the job and time taken to transmit the user request is not considered as a metric in this thesis. The main reason for not considering these two metrics is they will increase the processing time of each packet. Therefore to avoid the processing delay job size was ignored.

MPLS: The concept of Multiprotocol Label Switching is used in this thesis to label a processed packet. This approach avoids multiple processing of the same packet.

Health of the network: In general the health of the network is evaluated based on the status of the participating servers. In this thesis, status of each participating server is obtained before routing the user request to the appropriate server.

Data compression and encryption: In this thesis, security issues are ignored because they are not crucial to the central objective of the router design. So a user's request is not compressed or encrypted for transmission over the network.

1.6 Organization of Thesis

The rest of the thesis is organized as follows. Chapter 2 (*Related Work and Background Literature*) provides some necessary background and related work in the area of content routing design and architecture. Chapter 3 (*System Design*) discusses the various network designs and new architecture for the content router. A formal specification for the proposed content routing architecture is presented in Chapter 4 (*Specification of Intelligent Content-Based Router*). This chapter provides a formal design approach where the designed system is mathematically proved to be correct, which helps in developing a robust and fail-safe system. Chapter 5 (*Object Model and Functionality*) presents various object models for the designed content router and explains the functionality of newly designed content router. Chapter 6 (*Implementation*) contains the implementation details and some screenshots explained with a user scenario. In Chapter 7 (*Conclusion and Future Work*) we present a summary of our work and contributions. We conclude Chapter 7, and this dissertation by providing an outline for future research directions.

Chapter 2

Related Work and Background Literature

This chapter introduces the reader to the necessary background and related research work in content-based routing. Several mechanisms to improve the speed of information retrieval and message delivery and to enhance the efficiency of network applications have been proposed. After surveying the issues in conventional router design, in Section 2.1 we discuss several such mechanisms. Some of the approaches incorporate router technology while others do not. Section 2.2 gives a summary of the different products that were developed and Section 2.3 concludes this chapter with a summary.

2.1 Existing Mechanisms

Sheldon [DS94] discusses content routing using content tags / labels for documents in a Wide Area Information Service (WAIS) server using a semantic file system, and a source and a catalog file. A query, posed as a predicate, is used to identify keywords in a document. The source file contains the details of host name, host address, database name, port number, and a short description of the database. The catalog file contains a list of short headlines for each file in the database. The architecture described in this paper is similar to the one in [SDW+94]. The content routing system has a collection of documents and each document has a content label associated with it. Each content label contains a brief abstract of the documents related to that particular collection. Each query predicate contains a field name and the value to be searched. The mechanism of the design is that the user tries to refine the query as much as possible and then forwards it to the remote servers to find the result. This architecture uses the brute-force searching technique. This architecture is however not efficient and slow. In addition, the implementation cost is high because it requires to maintain large number of files.

Keshav and Sharma [KS98] discuss general design issues for routers including speed, scalability, consistency, cost, configuration and bandwidth. The primary design issues are speed and reliability. Reliability is attained using techniques such as: "hot spares, dual power supplies and duplicate data paths through the routers"[KS98]. The time taken to do lookups in the routing table typically has the greatest effect on the performance of a router. Decreasing the time required to lookup the destination address can increase the speed of the router. As the packet size decreases the number and hence cost of route lookups increases. Gupta, *et al* [GLM98], Srinivasan, *et al* [SV97], and Waldvogel, *et al* [WVT+97] are all examples of work addressing efficient routing table lookups. To increase the speed of packet forwarding (including route lookup), architecture with multiple parallel forwarding engines can also be used. A detailed scheme for load balancing parallel forwarding processing is discussed in [GLH95].

Another consideration in designing a router is the scheduling of incoming packets. A simple method is First Come First Serve (FCFS). This method, however, is not an efficient one because the chances of losing packets are high. Design of a fair queuing method by Demers et.al. [DKS89], however, resolves these problems at a somewhat higher implementation cost.

Another method used in increasing router performance is differentiated processing based on packet type. The increase in performance is achieved by using different schemes for the buffering and forwarding, filtering and classifying, and queuing and scheduling of different packets. These mechanisms can be applied at various levels. The mechanisms proposed in this thesis will differentiate between packets based on their data content and will allocate resources and customize processing accordingly.

Partridge, *et al* [P+98], Asthana, *et al* [ADJ+92], and Konstantinidou [K94] discussed hardware design issues related to very high performance (multi-Gigabit) routers.

To provide better performance, service and security in the face of increased demand for

Internet bandwidth, network providers are turning to "differentiated services". Kumar, *et al* [KLS98] concluded that the current Internet architecture is not meeting market demands and proposed the use of packet classification, packet scheduling, and buffer management tools to provide enhanced performance. They discussed router-based mechanisms for providing such differentiated services.

Challenger, *et al* [CID+00] survey various techniques for improving the performance of highly accessed web sites including the use of multiple processors, the caching of dynamic data, and efficient web site design. To reduce traffic to a web server, multiple servers running on different machines may be used to share the load. Such systems are, however, still addressed at a single location. Some sites also use replication to create copies of entire web *sites* (which may be geographically distributed). Unfortunately, if a replicated site fails, it cannot route incoming requests to other sites. A key issue with such systems is locating the sites. One method is to use Round Robin Domain Name Service (RR-DNS) [26, 27], which allows a single domain name to be associated with multiple IP addresses (one per site). But this technique has drawbacks including possible load imbalance and lost requests if a server fails because the client and name server cannot detect this. To avoid these problems, a TCP (Transmission Control Protocol) router can be used. The function of a TCP router is to accept requests from clients and forward them to the corresponding servers in a round robin fashion (possibly taking server load into account). Servers then respond directly to clients without router involvement. When a server node fails the TCP router can re-direct requests to other web servers. Another technique is the use of web-server accelerators. A web accelerator caches web documents and has a TCP router running on it. When a request from a client arrives, the accelerator first looks in its cache. If the requested object is found it is returned to the client, otherwise the router selects a server node to process the request. Various modifications have been made to these basic ideas.

Hunt, *et al* [HGK+98] discuss a TCP router, called a "Network Dispatcher", which supports load sharing over several TCP servers. The dispatcher is placed between the front-end clients and the back-end server and forwards requests from the clients to the server nodes. Responses from servers are returned directly, bypassing the network dispatcher. Though the performance of the "router" is good, it does not analyze the packet data but merely forwards packets to the most lightly loaded server node. Cardellini, *et al* [CCY00] discuss a similar system for geographic load balancing for scalable distributed web systems.

Andresen and McCune [AM98] discuss a model for hierarchical scheduling of Distributed World Wide Web Server clusters, which process the data dynamically. This model has a group of clusters, servers and clients. The server nodes in the clusters are aware of one another's existence. The system maintains information about the load and cache characteristics of all the clusters that are connected through the cluster server as well as network bandwidth information. Each server node in the cluster runs a scheduler algorithm (e.g. Crovella, *et al* [CFH99]) and one of the processes is responsible for linking these schedulers in a hierarchical way. A client's request is routed to the closest server for processing. If one node fails the system can dynamically change the connection process to any of the other nodes or other clusters using the cluster server.

Pai, et al [PAB+98] discuss a simple strategy, Locality-Aware Request Distribution (LARD), which is a content-based request distribution system. LARD focuses on static content. One of the advantages of this strategy/method over normal cluster-based network servers is that it offers enhanced performance due to its high cache hit rates. The architecture of LARD consists of back-end nodes and a front-end. The front-end is responsible for forwarding requests to the back-end nodes, which constitute the server. In routing a request, this strategy focuses on the content requested and the load on the back-end nodes. LARD uses hashing techniques to locate the requested data. Based on the load on each node, the front-end decides which node should process the given request. When a request arrives, it sends the request to a lightly loaded node, which caches the needed data. If the requested node is fully loaded it will send the request to a new node, which is not heavily loaded.

Song, *et al* [GC00] describe an architecture for a scalable and highly available web server accelerator based on caching data from frequently visited sites. These caches are also known as HTTP (HyperText Transfer Protocol) accelerators. The web server accelerators use multiple processors to provide more cache memory and higher throughput. The system works as follows: First the client sends a request into the network. A TCP router receives the request and passes it on to a nearby caching site. If the first site is not the owner of the requested object, it determines the owner and sends the request to the owner along with the TCP connection details. The owner fetches the object from its cache or from the back-end server if it is not in the cache. Finally the primary owner returns the requested object either directly or indirectly (through caching sites) to the client.

Song, *et al* [SLI+00] also provide an alternative design to [SLI+99] that includes a load balancer as a separate node, which may also choose to route the requests using content-based information. The load balancer has information about the availability and load details of each caching site. When the load balancer acts as a content router, it analyzes the content and directly routes the requests to the owner site, which will fetch the requested object either from its cache or from the back-end server.

Genova and Christensen [GC00] describe a layer 5 switch for implementing distributed web sites. A distributed web site consists of multiple local sites and the switch acts as a front-end for each local site. Each local site has one or more servers and caches information about the load on, and content available from, the server nodes. When a client makes a request, the switch consults the cache to see if the requested object is available in that local site and what the load information is for the server node. If the node is fully loaded and the request data is not available, the request is passed on to the next closest switch. After processing, the requested object is sent back to the client. The routing depends mainly on the data stored in the cache. In a globally distributed site, one can have any number of local sites. Each local site can have any number of server nodes. So, every time a new local site is created or a new server node is added a new cache should be created or the cache size should be increased.

2.2 Product Survey

Commercial systems for improving web access times are now becoming available. Cisco [C00] for example, discusses various protocols, such as Dynamic Feedback Protocol (DFP), Director Response Protocol (DRP), Web Cache Communication Protocol (WCCP), and Boomerang Control Protocol (BCP) that can be exploited for content routing. The DFP dynamically provides statistical information about the load on and availability of a server. The DRP gives information about the distance between a client and a server and it determines the server that is best capable of processing requested data. The WCCP redirects data to other servers based on information present in the cache. The BCP uses agents to provide network information for routing. The Cisco content router uses information supplied by these protocols to carry out its processing.

IntelliDNS [ID] provides a solution for Internet traffic management. The design acts as a global load balancer with intelligence for managing Internet traffic and for content redirection. The set of metrics used for managing the traffic and content redirection are network performance, clients proximity and server status to choose the optimal site to serve clients requests. IntelliDNS supports both DNS based and HTTP based traffic redirection. If the request is a DNS based request from the client the IntelliDNS gives its own alternate IP address and redirects the client to the best and efficient content server based on the set of metrics listed above. It also supports protocol re-mapping from HTTP to Hypertext Transfer Protocol Security (HTTPS), Real - Time Streaming Protocol (RTSP) and Microsoft Media Server (MMS). The main drawbacks are the design supports only DNS and HTTP based request and it uses a large database to store the client's geographical location and the server location.

Arrowpoint's [AP] Web Network Services (WebNS) provides a solution for URL and cookie based intelligent switching. WebNS is designed for name based switching. It uses the full URL and cookie to select the best server or site for the user's request. The WebNS switch knows the full information about the client from the cookie and it also knows the user's request and the server to process the client's request based on network information and server status. The Web switch parses the URL to identify the client's request. Based on the request the switch finds a suitable server or site. The Web switch periodically checks for the status of the servers. The client is switched to the new server or site that is selected for processing the request. The requested data is sent back to the client through the shortest path.

2.3 Summary

This chapter discussed the history of the work done related to content routers. Section 2.1 outlines various existing mechanisms and Section 2.2 discusses various product surveys. We observe that the past research work do not address the entire range of problems that were identified. We used the material discussed in this chapter as one of our motivation for our work.

Chapter 3

System Design

This chapter describes our architecture for the content-based router and proposes various network designs where the designed content router can be used efficiently. Section 3.1 gives a brief description of content router and discusses the various components of the newly designed content router briefly. Section 3.2 proposes the various network designs. Section 3.3 explains in detail the architecture of our content router and Section 3.4 concludes with a summary of the chapter.

3.1 Content Router: Description

The existing content routers fail to deliver correct information to the right people in appropriate time. So a need for an intelligent-content based router arises. A content router analyzes the data present in a packet before forwarding the packet to the appropriate server. The main reason for developing a new intelligent content-based router is to reduce network traffic and to optimize routing cost, which in turn could potentially increase the performance and decrease the latency of the content router. The different components present in our content router are Packet Inspector, Resource Inspector and Scheduler unit. The Scheduler has the load and distance information between the content router and the server. Based on these informations a user's requests are forwarded to the appropriate server. The above mentioned components are embedded in the Intelligent Content router that I have implemented. Even though some existing content routers possess these components, they fail to examine different types of TCP-based request. The content router implemented in this thesis examines all types of TCP-based user requests. These new features make this design unique when compared with current content-based routers.

3.2 Network Design

The content router proposed in this thesis can be used in various network design models. Each design has its own advantages. The various network designs proposed in this thesis are:

1. Intelligent content routing for metropolitan type of networks - Option A, B and Option C.
2. Intelligent content routing for wide area networks.

The network designs mentioned above are discussed in detail in the following section.

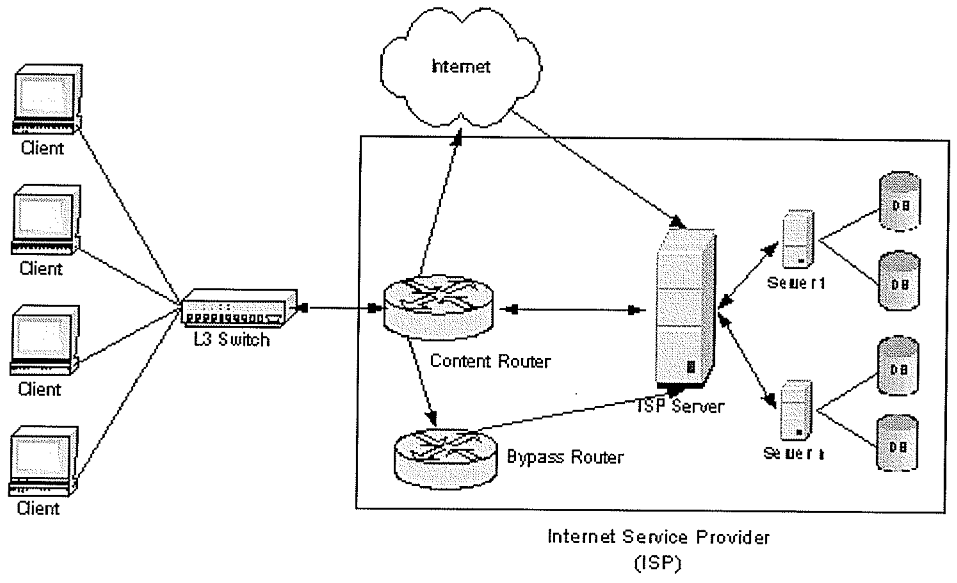


Figure 3.1 *Design for Metropolitan type of Network - Option A*

3.2.1 Intelligent Content Router for Metropolitan Networks - Option A.

Figure 3.1 shows one design for metropolitan networks. The components present in Option A are: the different clients connected to a switch. The Internet Service Provider (ISP) network has a content router connected to an ISP server. The Layer 3 switch, which is outside the ISP network, is connected to the content router. A bypass router is connected to the content router. The ISP server may have many differentiated servers connected to it, which offers different services. Each server has different databases on it. The content router is also connected to the Internet. This model is specifically designed for registered services with the ISP. The registered services can be a single company with different branches or it can be different companies with a single major server.

3.2.1.1 Functionality

This section discusses the functionality of the design for metropolitan network - option A. The clients send requests into the network. The Layer 3 switch captures the user request in a packet format and forwards the packets to the content router present in the ISP network. The main function of a Layer 3 switch is to collect all user requests on a queue basis. The content router reads the header and tokenizes the data. If the request is a URL based request the content router sends the request to the Internet and continues to process the next request. If it is a registered service request, the content router finds a suitable server for processing the request based on the information given by the ISP server. The client's request is forwarded to the best appropriate server through the bypass router connected to the content router. The ISP server sends the processed request back to the client via the content router. The response is sent back using different queuing strategies. The three different queuing strategies are

1. High Priority Queuing (HPQ).
2. Low Priority Queuing (LPQ).
3. Unprocessed Queuing (UQ).

The requests for registered services and their responses are sent through the HP Queue. The ISP server sends the response to the content router, which sends it back to the Layer 3 switch, which forwards the response to the client. The URL response from the Internet to the content router is stored in the LP Queue. The LP Queue is processed only when the HP Queue is empty. The remaining requests and responses are sent to the Unprocessed Queue. The Unprocessed Queue is processed when the HP and LP Queues are empty.
The next design discussed in detail is Metropolitan Network - Option B.

3.2.2 Intelligent Content Router for Metropolitan Networks - Option B.

Figure 3.2 shows another design for metropolitan network - Option B. The various components present in Option B network are: different clients connected to the Layer 3 switch. The Layer 3 switch is connected to the content router present in the Internet Service Provider network. The content router is connected to an ISP router as well as to the Bypass router. The ISP router is connected to the ISP server. The ISP router is also connected to Internet and to other network routers. The ISP server has many differentiated servers connected to it, which offer different services. Each server has different databases on it. The next section discusses the functionality of this network.

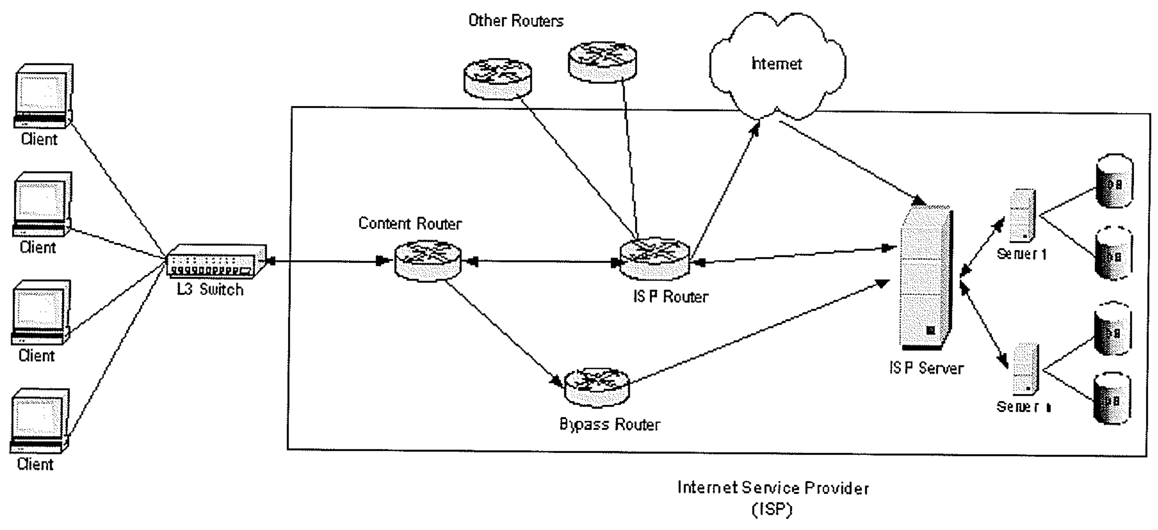


Figure 3.2 *Design for Metropolitan type of Network - Option B*

3.2.2.1 Functionality

The clients send request into the network. The Layer 3 switch captures the user request in a packet format and forwards the packet to the content router inside the ISP network. The main function of a Layer 3 switch is to collect all user requests from different clients on a queue basis. The content router reads the header and tokenizes the data. If the client's request is an URL request, the content router forwards the packet to the ISP router. The ISP router forwards the request to the Internet and waits for the response. The ISP router also forwards the requests to their respective destination, which comes from other routers that are connected to it. Once a response is obtained from Internet the ISP router forwards the response back to the content router. If the request is a registered service requests the content router finds a suitable server for processing the request based on the information given by the ISP server. The client's request is forwarded to the best appropriate server through the bypass router connected to content router. The processed request is sent back to the client via the content router. The response is sent back to the client using different queuing strategies discussed in Option A network. The next section discusses Option C network in detail.

3.2.3 Intelligent Content Router for Metropolitan Networks - Option C.

Figure 3.3 shows another design for metropolitan network-option C. The different components present in Option C network are: clients connected to a network, the ISP has a content router, which is connected to a Layer 3 switch as well as to the Internet. The Layer 3 switch has some content routers connected to it. The content routers present in the ISP network are connected to the ISP network's gateway. The ISP server has many registered servers connected to it. Each server has some data of interest in it.

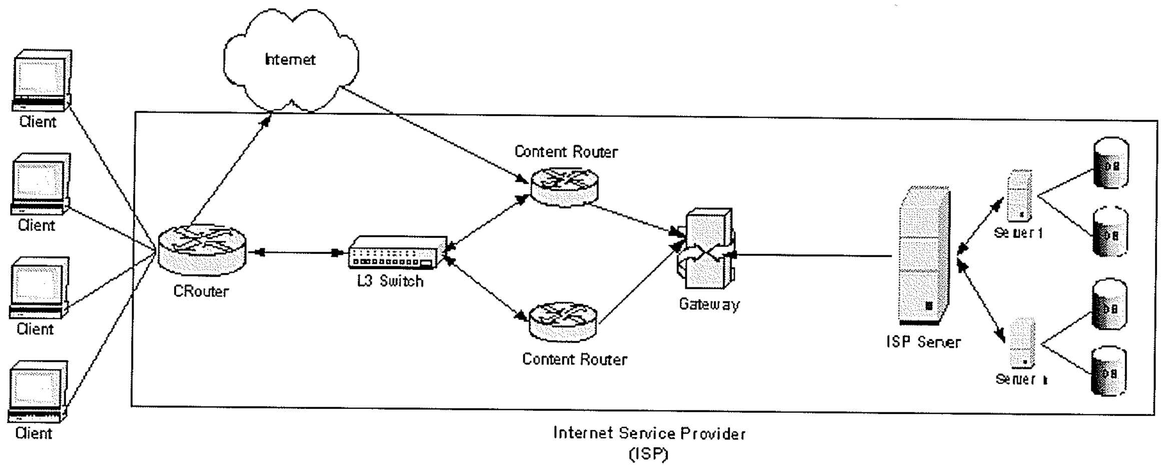


Figure 3.3 *Design for Metropolitan type of Network - Option C*

3.2.3.1 Functionality

Clients send in their request and the content router present at the entrance of the ISP network captures the user request in the packet format. The content router reads the header of the captured packets and tokenizes the data present in the packet. If the request is an URL request the content router forwards the packet to the Internet for further processing. If the request is for a registered service the content router forwards the packet to the Layer 3 switch. The main function of the Layer 3 switch is to collect all user requests from the content router and forwards them to different content routers that are connected to the gateway of the ISP network. The Layer 3 switch forwards the user request to the content routers in a weighted round robin fashion. The length of the router queue is the weight used for forwarding the user request. Once the content router captures the user request the content router finds a suitable server for processing the request based on the information given by the ISP server. The client's request is forwarded to the best appropriate server through the gateway of the ISP network.

3.2.4 Advantages

1. The different designs discussed above are efficient because of the Content Router present inside the ISP Network.
2. Routing is cheap, quicker and efficient for the registered servers within an ISP Network.

3.2.5 Intelligent Content Routing for Wide Area Networks.

Figure 3.4 shows the design for wide area networks. The various components present in this design are clients, a client side content router, a server side content router and servers with different databases on them. The client side content router is connected to the Internet. A Server side content router has different servers connected to it. Each server has different databases on it. In addition to the two routers there is a Gigabit Network connected to the server side content router and the client side content router. This design is well suited for a big company with many branches around the globe. Section 3.2.5.1 discusses the functionality of this design.

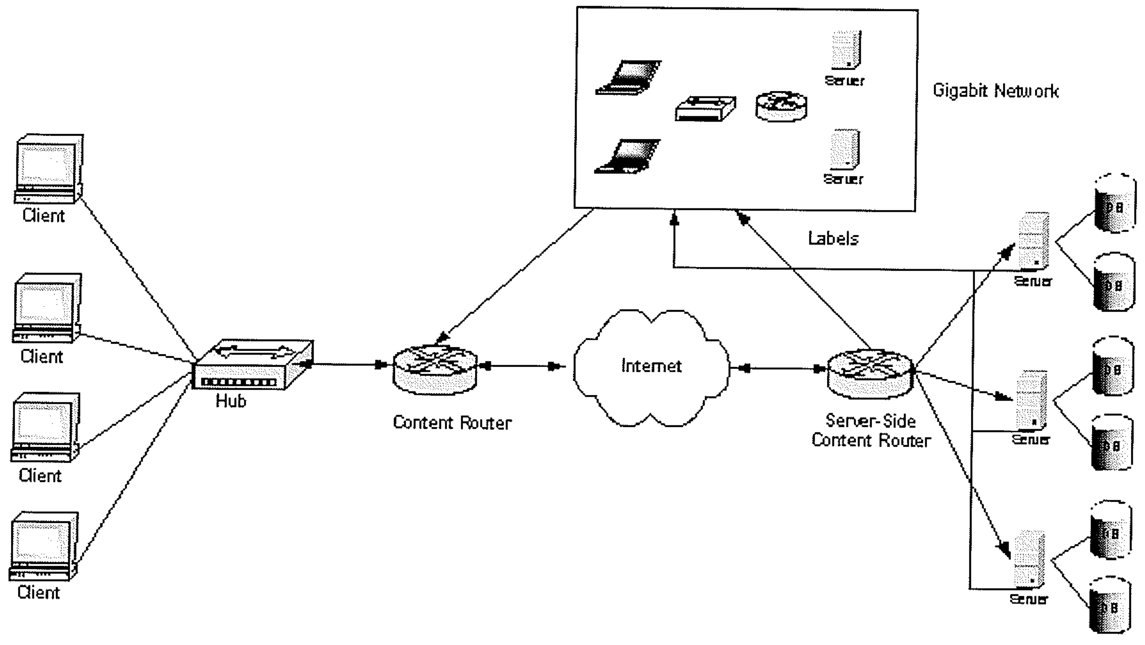


Figure 3.4 *Intelligent Content Routing - Wide Area Network*

3.2.5.1 Functionality

Clients send in their request and the content router captures the request in the form of packets. The data present in the packet is analyzed and tokenized. The tokenized data is sent to the server-side content router through the Internet to find an efficient server for processing the client's request. The content router forwards the packet with the tokenized data to the server-side router. The tokenized data sent by the client-side content router is read by the server-side router and finds an efficient Server based on a set of metrics, like system resources, proximity of the client and the server and the status of the server. Based on the metrics the server router selects a server and forwards the client request to the appropriate server. After processing the request the server sends the response back to the server-side content router. The server-side router captures the processed packet.

While sending the response back to the client-side content router the server-side router labels the processed packet and forwards them to the Gigabit network for a quicker response from the server. The Gigabit network captures the labeled packet and forwards the packet back to the client-side content router. The content router captures the response and looks for a label in the packet. If the packet is labeled the content router forwards the packet back to the client without processing the packet. If there is no label the content router starts the processing of packet and forwards the packet to the server router.

The labeling of the packet is done through the Multiprotocol Label Switching (MPLS). The main advantage of using this system is to avoid heavy traffic on the Internet and process requests in an efficient and fast approach. The content router starts processing the packets without knowing the status of the packet that is processed or unprocessed. To avoid multiple processing the processed packets are labeled. So when the content router captures a packet it looks for the label and forwards the packet to the client, thereby enhancing processing time. The main functionality of using MPLS in this thesis is to label the processed packets.

3.2.5.2 Advantages

1. This design is efficient and fast because the response from the server is sent through a different path instead of the same forwarding path.
2. Traffic is reduced and time taken for processing each packet is minimized.

3.2.6 Global Network Structure

Figure 3.5 shows the design of Global Network Structure. This design is an extension of The Wide Area Network design with replication of intelligent content routers in different areas. The different components present in this design are four different networks, which are interconnected through edge routers. Each network has different clients connected to a switch, and a content router connected to different servers. Each server has different databases on it. The edge routers act as the communication media between these areas.

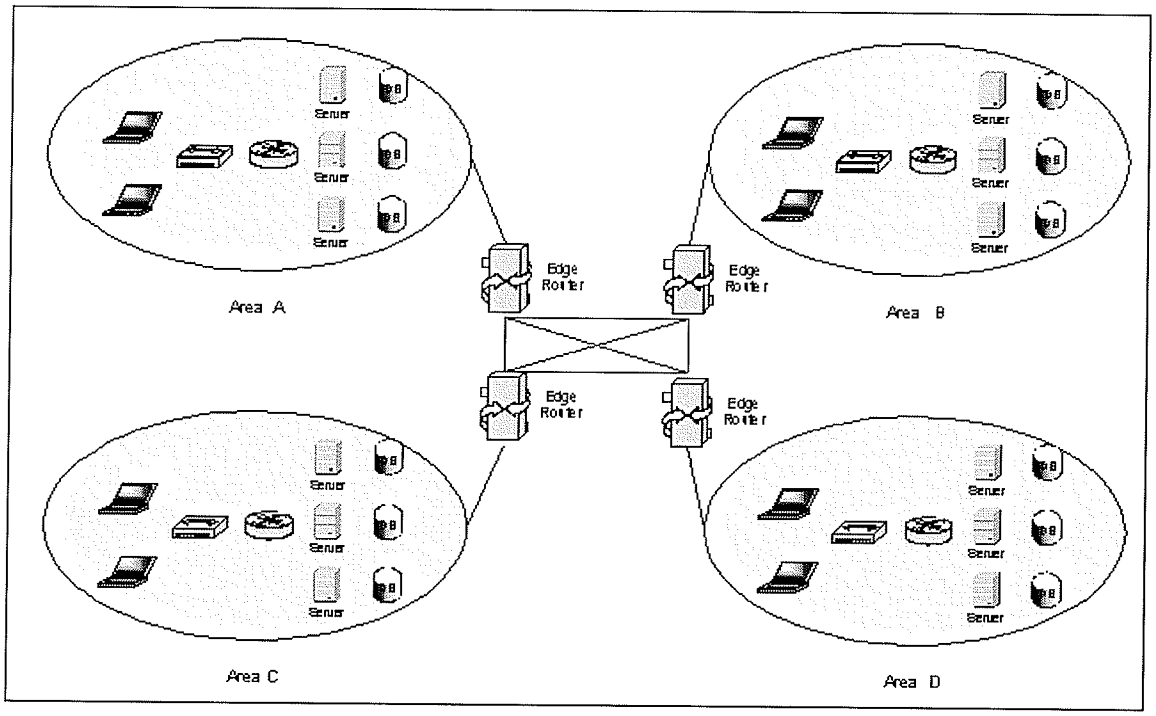


Figure 3.5 *Global Network Structure*

3.2.6.1 Functionality

The main functionality of this design is sharing of resources between locations.
Each location has a Resource Agent. These agents are mobile i.e., they are capable of moving from one place to another. The resource agents move from place to place and collect all the available resource's information update the resource table present in each local area. When the clients send requests into the network the content router reads the header and analyzes the data and finds a suitable server for processing the request. If the requested data in unavailable in the local area it finds a suitable server in remote a location from the resource table maintained by the resource agent. Once a remote server is selected the user request is forwarded to the appropriate server through the edge routers. If there is any change in resources, all the resource tables are updated by the resource agents. The update operation can also be performed by sending a broadcast message to all location. But the main disadvantage of sending a broadcast message is that the local agent does not get any acknowledgement from other resource agents. So the message can even be lost during the data transmission if the network connection is bad.

3.3 System Architecture

The high-level system architecture of the proposed intelligent content-based router is shown in Figure 3.6. Each component is briefly described below.

The *Packet Capture* and *Packet Data Extraction / Analysis* module enables the unit to capture and extract the data in each packet of a user's request. This data is the content that is routed to the appropriate server at that moment based on a set of metrics. This component of the system intercepts the user's request data stream in the form of packets and then extracts the data content (i.e., the payload) it contains for routing.

A core component of the system is the *Resource Manager*. The main job of the Resource Manager is to assemble vital information about the resources available in the system for ease of access and fast decision-making. The resources for e-commerce and other Internet applications are often stored in databases (at the participating servers). The Resource Manager collects resource information about the number of databases available in the system, the addresses of these databases, and permission data (such as who can obtain the database addresses) and stores the data collected in a resource table. This resource table is used to feed the load-balancing unit (discussed below). To implement this component, we adopted intelligent mobile agent technology. Mobile agents are suitable because they enable us to seamlessly and transparently assess servers (at remote locations) and retrieve appropriate data of interest. The agents only need to know the address (IP address or full domain name) of the resource and a known set of database types. The agents can retrieve the metadata of each database, such as the name of the schemas, the description of the schemas, and table definitions, etc. This information is necessary to make informed judgements on where to find the available resources for the application. The databases are transparent to the system.

The *Load Balancing System*, a major part of system, uses the information assembled by the Resource Manager to facilitate content-based routing. It is responsible for scheduling and allocating transactions to the various servers for execution based on the current processing / work load information of each server. This unit answers questions such as: (i) How busy is each server? (ii) Which server can process the request in the shortest time? I used existing queuing and scheduling algorithms (as in operating systems and other distributed systems) to realize an efficient and robust system.

Finally, the *Content Routing Unit* is responsible for the actual redirection of the user payload based on the contents of the packets. Using the assembled data of the Resource Manager and the recommended scheduling plans of the Load Balancing System (routing tables, network nodes, application resources, etc), the Content Routing Unit selects the specific destination to route the user payload to. The decision about where to go is based on the accumulated and cached information from the Resource Manager and the Load Balancing System.

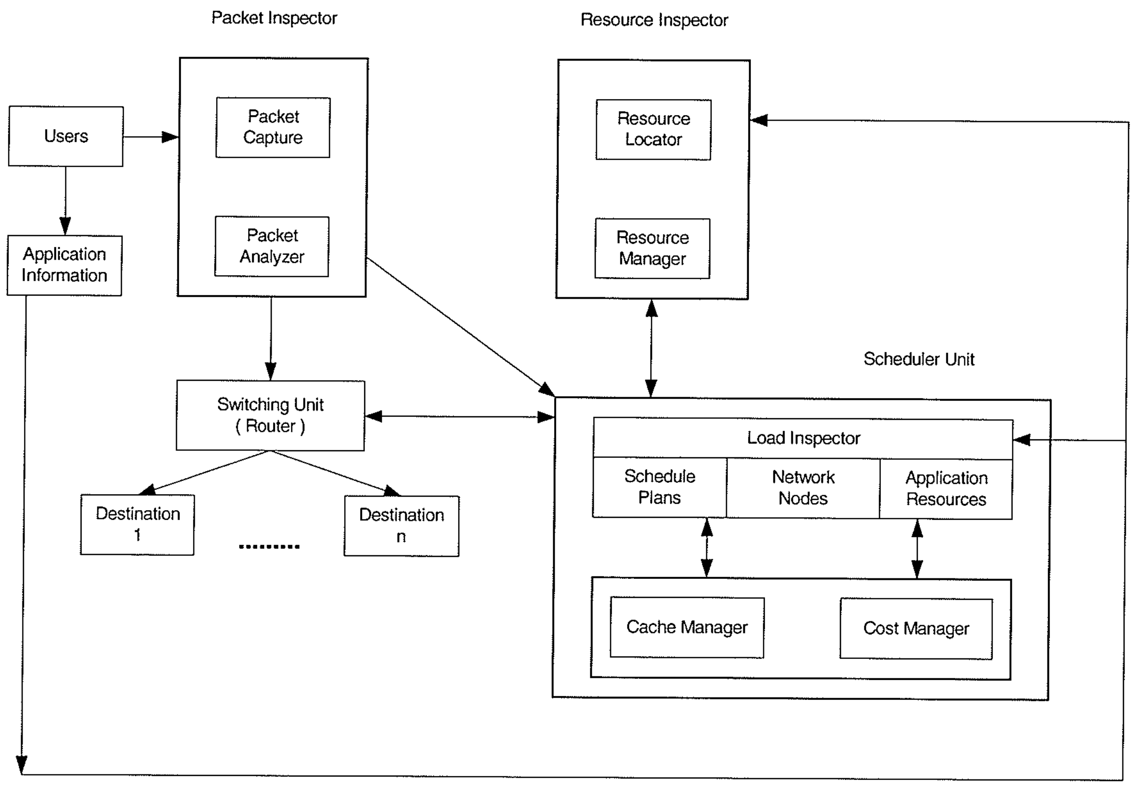


Figure 3.6. *Intelligent Content - Based Routing Architecture*

3.4 Summary

This chapter began with the discussion of a short description of content router and mentioned the different components of the newly designed content router. Section 3.2 presented the different network designs and their functionality. The advantages of the different designs are also presented. Section 3.3 presented the architecture of our content router. Chapter 4 presents a formal specification for the designed content-based router.

Chapter 4

Specification of Intelligent Content-Based Routing

This chapter presents a formal specification for the newly designed content-based router. To develop a robust and fail safe system, formal specification is one of the approaches that can be used. This section discusses formal specification of Intelligent Content-Based Routing for E-Commerce Applications. The specification describes the requirements and functionality of the system and controls the software complexity and enhances the quality and reliability of the system. A formal specification is usually written using a formal specification language. This language has a well defined syntax and semantics. The formal specification language used in this thesis is Z. The main reason for using Z is it has tool support for typechecking the syntax and semantics of Z - based specifications. Section 4.1 discusses the problem model and description of the system. Section 4.2 gives the Z-specification, before it ends with a summary.

4.1 Problem Model and Description

This specification mainly deals with one of the important functionalities of networking concepts i.e. routing. The specification is done for Content-Based Routing for E-Commerce Applications. Router is a device, which forwards packets from one machine i.e. source to the other machine i.e. destination. In normal IP routing the router checks the destination address of the packet and forwards the packet to its corresponding destination. But a Content-based router analyzes the data in the packet before forwarding it to a destination where the data can be reached. This concept of routing is specified using Z-specification language. The different operations that are performed are: defining the structure of a packet, creating a packet, creating a user list, adding new users, logging into the system, list for logged users, sending a request. Section 4.2 gives the specification of the routing system.

4.2 Specification

This section gives the specification for Content-Based Routing. The basic set types that are used in this specification are defined below. The first few set types upto *DATA* are the different fields present in an IP packet. Each set type is explained in the *PacketDef* schema.

[*IPHEADERLEN*, *TYPEOFSERVICE*, *FLAGS*, *FRAGOFFSET*, *IDENTIFICATION*, *TIMETOLIVE*, *PROTOCOL*, *HEADERCHECKSUM*, *TOTALLENGTH*, *OPTIONS*, *DATA*, *VERSION*]

The name and passwd types are used to store the registered users list and password.

[*NAME*, *PASSWD*, *SERVERADDRESS*, *RESOURCENAME*]

The *CPUAvail*, *MEMAvail* and *QueueLEN* are the load details of different servers and the *DISTANCE* is the distance between the server and the client.

$CPUAvail == \mathbb{N}$

$MEMAvail == \mathbb{N}$

$QueueLEN == \mathbb{N}$

$DISTANCE == \mathbb{N}$

The serverstatus type gives the status of the participating server.

$SERVERSTATUS ::= Active \mid Down$

$BOOLEAN ::= True \mid False$

A *RESPONSE* is a message or a result given by the system after each operation performed on it. The different responses given by the content router specify the network administrator about the router's performance. The different responses given by the system are defined below.

RESPONSE ::= *PacketDefined*
| *PacketCreated*
| *NewUserAdded*
| *LoggedInSuccessfully*
| *RequestSent*
| *ResourceTableUpdated*
| *ServerAddressFound*
| *SystemStatusObtained*
| *DistanceObtained*
| *ScheduleTableFormed*
| *DestAddressChanged*
| *CacheUpdated*

The first aspect of the system is to describe its state space. Each operation in the system is defined within a schema. A schema has two parts, the declaration part and the predicate part. The parts are separated by a central line. The part above the central line is the declaration and below the central line is the predicate. The predicate part specifies the requirements of the values of the variables defined in the declaration part. The *PacketDef* schema defined below gives the structure of an Internet Protocol (IP) packet. Each packet contains the version of IP currently used, IP header length indicates the header length, Type of Service, Total length of the IP packet, Identification indicates the current packet, Flags, Fragment Offset, Time-to-Live is a counter which gradually decrements down to zero, and the packet is discarded. The Protocol indicates the next level protocol of packet such as TCP, UDP etc. Header checksum ensures IP header integrity, Source Address specifies where the packet is coming from, Dest Address specifies the packet's destination address, Options provides additional security and finally the packet has the Data. The result for this schema is *"PacketDefined"*.

PacketDef

$ver: VERSION$
$ipheaderlen: IPHEADERLEN$
$tos: TYPEOFSERVICE$
$tl: TOTALLENGTH$
$id: IDENTIFICATION$
$flg: FLAGS$
$frgoff: FRAGOFFSET$
$tol: TIMETOLIVE$
$proto: PROTOCOL$
$hc: HEADERCHECKSUM$
$sourceip: SERVERADDRESS$
$destip: SERVERADDRESS$
$op: \mathbb{P}\ OPTIONS$
$data: \mathbb{P}\ DATA$
$Re!: RESPONSE$

$ver \notin \emptyset$
$ipheaderlen \notin \emptyset$
$tos \notin \emptyset$
$tl \notin \emptyset$
$id \notin \emptyset$
$flg \notin \emptyset$
$frgoff \notin \emptyset$
$tol \notin \emptyset$
$proto \notin \emptyset$
$hc \notin \emptyset$
$sourceip \notin \emptyset$
$destip \notin \emptyset$
$Re! = PacketDefined$

The next schema operation is *PacketCreation*. The *PacketCreation* schema captures the inputs needed for creating the packet. The fields discussed in the previous schema cannot be empty except the op (options) and data fields. A packet can be an empty packet without any data or it can carry some data for transmission. Once all the fields are filled up the packet is created and it is ready for transmission. The result for this schema operation is "*PacketCreated*".

```
PacketCreation
ΔPacketDef
vers?: VERSION
iph?: IPHEADERLEN
typos?: TYPEOFSERVICE
totlen?: TOTALLENGTH
identi?: IDENTIFICATION
flag?: FLAGS
fragoff?: FRAGOFFSET
timetol?: TIMETOLIVE
prototype?: PROTOCOL
hcheck?: HEADERCHECKSUM
sip?: SERVERADDRESS
dip?: SERVERADDRESS
opt?: ℙ OPTIONS
req?: ℙ DATA
Re!: RESPONSE
──────────────
ver = vers?
ipheaderlen = iph?
tos = typos?
tl = totlen?
id = identi?
flg = flag?
frgoff = fragoff?
tol = timetol?
proto = prototype?
hc = hcheck?
sourceip = sip?
destip = dip?
op = opt?
data = req?
Re! = PacketCreated
```

The next schema operation is maintaining a user list and a login list for those people who login to the system. Each user has a username and a password to login. The main reason for maintaining a user list is that in all E-Commerce applications only registered users are allowed to perform some of the core transactions. In order to commit the transactions a user list is maintained and verified. Each time a user logs in his/her password is verified before committing a transaction. The next set of schemas describes the maintenance of registered user list.

UserList

$users: NAME \rightarrowtail PASSWD$

$loggedusers: \mathbb{P}\ NAME$

The Initial User List schema contains the initial value of the users list and login list. Initially there are no users. So the two fields are empty.

InitialUserList

$UserList$

$users = \varnothing$

$loggedusers = \varnothing$

The AddUser schema captures the operation of adding a new user to the system. This operation has a change in the class *UserList*. When a new user is added there are two inputs name and password and Re! is the result obtained for this schema.

AddUser

$\Delta UserList$

$name?: NAME$

$passwd?: PASSWD$

$Re!: RESPONSE$

$name? \notin \mathrm{dom}\ users$

$users' = users \cup \{(name? \mapsto passwd?)\}$

$Re! = NewUserAdded$

The name that is given by the user must not be in the User List. If it exists the user has to give a new name for registering. The name and password field should not be empty. Once the user registers by supplying the name and password it is added to the users list. The result obtained is *NewUserAdded.*

The next schema is the Login operation. All the registered users can try to login to the system. The inputs given are name and password and the output Re! is the result.

Login

$\Xi UserList$
$name?: NAME$
$passwd?: PASSWD$
$Re!: RESPONSE$

$name? \in \mathrm{dom}\ users$
$passwd? \in \mathrm{ran}\ \{(name? \mapsto passwd?)\}$
$loggedusers' = loggedusers \cup \{name?\}$
$Re! = LoggedInSuccessfully$

The name given by the user is checked in the users list for the registered user. If it is a registered user the name is checked for its corresponding password which is mapped to the user name. If both are valid, the user name is added to the loggedin users list and the result obtained is "*LoggedInSuccessfully*".

The User Request schema discusses sending a user request to the network. The input supplied for this operation are, the user name and the data to send. Re! is the result obtained.

UserRequest

$\Xi UserList$
$name?: NAME$
$request?: DATA$
$Re!: RESPONSE$

$name? \in loggedusers$
$Re! = RequestSent$

The name given by the user is checked in the loggedin users list. If the user name is not present in the loggedin user list the user has to login first. If the user is loggedin the request is sent to the network. The result obtained is "*RequestSent*".

The next schema operation is to maintain a server list, which has the list of all the registered servers.

ServerAddressList

$serverlist: \mathbb{P}\ SERVERADDRESS$

The Resource Table schema maintains a list of resource name and its corresponding server address.

ResourceTable

$resourcelist: RESOURCENAME \nrightarrow SERVERADDRESS$

The Initial Resource Table list contains the initial value of the resource list.

InitialResourceTable

$ResourceTable$

$resourcelist = \varnothing$

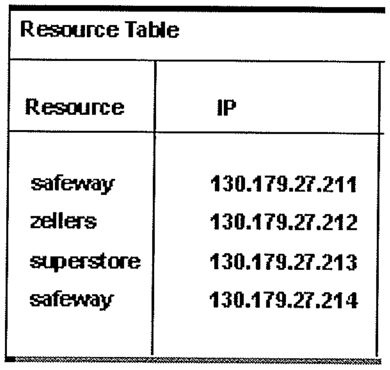
The AddEntries schema describes adding new resources to the system. This operation affects the ResourceTable. When a new resource is added, two inputs and a response are obtained.

AddEntries

$\Delta ResourceTable$
$resourcename?: RESOURCENAME$
$loc?: SERVERADDRESS$
$Re!: RESPONSE$

$loc? \notin \mathrm{ran}\ resourcelist$
$resourcelist' = resourcelist \cup \{(resourcename? \mapsto loc?)\}$
$Re! = ResourceTableUpdated$

The two inputs are resource name and server address. The condition to add the resources to the table is that the server address should not be in the resource list. If the server address exists the corresponding resource name is checked. If the resource name is different, the resource and the address are added else they are discarded. If the resource name exists in the list the corresponding server address is checked with the input server address. If both the addresses are different the resource name and the server address are added to the list else the resource is discarded. The result obtained is *"ResourceTableUpdated"*. Figure 4.1 shows the structure of the Resource Table.



Resource Table

Resource	IP
safeway	130.179.27.211
zellers	130.179.27.212
superstore	130.179.27.213
safeway	130.179.27.214

Figure 4.1 *Resource Table*

The Data Location Table schema has two components; *matchedentries* and the *dltserverlist*. The *matchedentries* maintains a list of all instances of resources and server address from *ResourceTable* based on users request. The *dltserverlist* maintains a separate list for all the server address stored in the *matchedentires*.

DataLocationTable

matchedentries: RESOURCENAME ⇸ *SERVERADDRESS*
dltserverlist: ℙ *SERVERADDRESS*

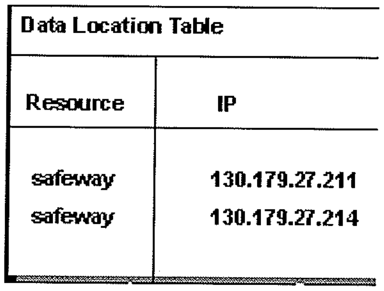
The Initial Data Location table has zero entries when the system is activated.

InitialDLTable

DataLocationTable

matchedentries = ∅
dltserverlist = ∅

Each entry in the Data Location table has a resource name and its corresponding server address. Figure 4.2 shows the structure of the Data Location Table.



Data Location Table

Resource	IP
safeway	130.179.27.211
safeway	130.179.27.214

Figure 4.2 *Data Location Table*

The Find Server Address schema describes finding a server address from the Resource table list for the tokenized data. The input for this schema is tokenized data and the output is server address.

FindServerAddress

$\Delta DataLocationTable$
$\Xi ResourceTable$
tokenizeddata?: RESOURCENAME
loc!: SERVERADDRESS
Re!: RESPONSE

$tokenizeddata? \in \mathrm{dom}\ resourcelist$
$loc! = resourcelist\ (tokenizeddata?)$
$matchedentries' = matchedentries \cup \{(tokenizeddata? \mapsto loc!)\}$
$dltserverlist' = dltserverlist \cup \{loc!\}$
$Re! = ServerAddressFound$

The input is checked in the resource list maintained by the resource table. If the tokenized data is not in the list, the packet is routed to the original destination address present in the packet. If the tokenized data exists in the list the corresponding server address is obtained. Both the data and the server address are stored in the data location table and the server address is also stored in a separate server list maintained by the Data Location Table. The result for this schema is *"ServerAddressFound"*.

The next schema gives the structure of the System Status Table. It has the server address and the status of the server i.e. active or down.

SystemStatusTable

$statusentries: SERVERADDRESS \rightarrowtail SERVERSTATUS$

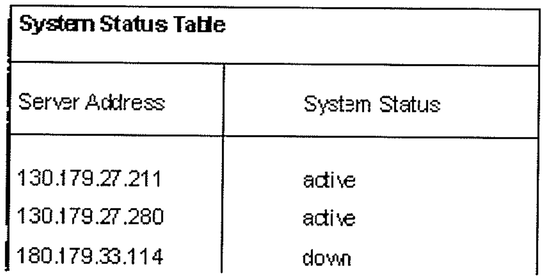
The Initial System status list is empty.

InitialSST

SystemStatusTable

$statusentries = \varnothing$

Figure 4.3 shows the structure of the System Status Table.



System Status Table	
Server Address	System Status
130.179.27.211	active
130.179.27.280	active
180.179.33.114	down

Figure 4.3 *System Status Table*

The Ping function defined below is used to find the status of a server.

$$ping: SERVERADDRESS \nrightarrow SERVERSTATUS$$

The Find System Status schema gives the status of the system. This schema takes the serverip as the input and gives the server status as output. The response is stored in Re!.

FindSystemStatus

$\Delta SystemStatusTable$
$\Xi ServerAddressList$
$serverip?: SERVERADDRESS$
$servstatus!: SERVERSTATUS$
$Re!: RESPONSE$

$serverip? \in serverlist$
$servstatus! = ping(serverip?)$
$statusentries' = statusentries \cup \{(serverip? \mapsto servstatus!)\}$
$Re! = SystemStatusObtained$

The input serverip is checked in the server list maintained by the ServerAddressList schema. If the serverip is found, the ping function is applied on the server to find the server's status. The status is stored in *servstatus!*. The final status with its corresponding server address is stored in the system status table. The result otained is *"SystemStatusObtained"*.

The ProximityTable schema defines the structure of the Proximity table. It has two columns server address and distance.

ProximityTable

SERVERADDRESS: SERVERADDRESS
DISTANCE: DISTANCE
distentries: SERVERADDRESS ⇸ DISTANCE

Initially the Proximity table is empty.

InitialPT

ProximityTable

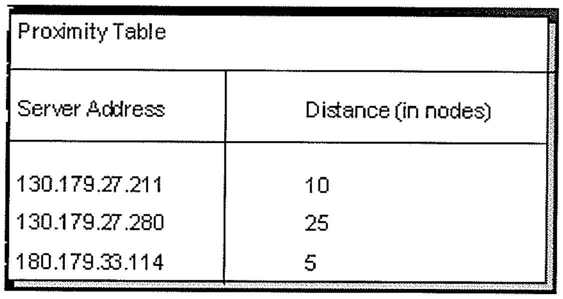
---

distentries = ∅

Traceroute is the function used to find the distance between the content router and the server.

| *traceroute: SERVERADDRESS ⇸ DISTANCE*

Figure 4.4 shows the structure of the Proximity Table.



Proximity Table

Server Address	Distance (in nodes)
130.179.27.211	10
130.179.27.280	25
180.179.33.114	5

Figure 4.4 *Proximity Table*

The FindDistance schema gives the distance between the content router and the server. It takes one input (i.e. serverip?) and produces one output (i.e. distance!) and the response is stored in Re!.

FindDistance

$\Delta ProximityTable$
$\Xi ServerAddressList$
$serverip?: SERVERADDRESS$
$distance!: DISTANCE$
$Re!: RESPONSE$

$serverip? \in serverlist$
$serverip? \in \mathrm{dom}\ traceroute$
$distance! = traceroute(serverip?)$
$distentries' = distentries \cup \{(serverip? \mapsto distance!)\}$
$Re! = DistanceObtained$

The input serverip is checked in the server list to find whether the input serverip is valid. If it exists in the server list the traceroute function is applied to the input serverip and the distance is stored in the output variable. Once the distance is obtained the Proximity table is updated with the distance and the corresponding server address. The response obtained is *"DistanceObtained"*.

The *LoadDetails* schema encapsulates the structure of the load details. The different components that are necessary for obtaining the load details are: percentage of free CPU available (*CPUAvail*), percentage of free memory available (*MEMAvail*), processor queue length (*QueueLEN*), and the distance between the router and the server (*DISTANCE*). This encapsulated structure is used by the *loadinfolist* function defined in *ScheduleTable* schema.

LoadDetails

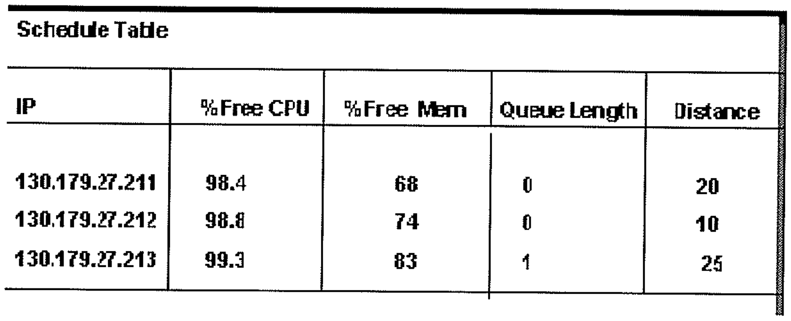
$CpuInfo: CPUAvail$
$MemInfo: MEMAvail$
$QueueLen: QueueLEN$
$Dist: DISTANCE$

| *loadinfo: SERVERADDRESS ⇸ LoadDetails*

The Schedule Table schema gives the structure of the schedule table. The different fields present are serveraddress, percentage of CPU avialable, percentage of memory available, length of the processor queue and the distance between the router and the server.

ScheduleTable
loadinfolist: SERVERADDRESS ⇸ LoadDetails

Figure 4.5 shows the structure of the Schedule Table.



Schedule Table

IP	%Free CPU	%Free Mem	Queue Length	Distance
130.179.27.211	98.4	68	0	20
130.179.27.212	98.8	74	0	10
130.179.27.213	99.3	83	1	25

Figure 4.5 *Schedule Table*

The next schema, *FormScheduleTable* describes forming the schedule table. The input for this schema is the server address and the output is the load details discussed above. The input is checked in the server list maintained in the data location table. If the server address exists in the data location table, the status of the server is checked in the system status table. The precondition for finding the load details is that the server status should be active. If the server status is down the corresponding server address is discarded and the next server address is processed. Once the server status is active, the load details of the input server are obtained by applying the *loadinfo* function, which is defined above. After obtaining the load details the schedule table is updated with the load information with the corresponding server address mapped to it. The result obtained for this schema is *"ScheduleTableFormed"*.

FormScheduleTable

$\Delta ScheduleTable$
$\Xi DataLocationTable$
$\Xi ProximityTable$
$\Xi SystemStatusTable$
serverip?: SERVERADDRESS
cpuinfo!: CPUAvail
meminfo!: MEMAvail
qlen!: QueueLEN
dist!: DISTANCE
serverstatus!: SERVERSTATUS
ld: LoadDetails
Re!: RESPONSE

$serverip? \in dltserverlist$
$serverstatus! = statusentries\ (serverip?)$
$serverstatus! = Active$
$ld = loadinfo(serverip?)$
$cpuinfo! = ld\ .\ CpuInfo$
$meminfo! = ld\ .\ MemInfo$
$qlen! = ld\ .\ QueueLen$
$dist! = ld\ .\ Dist$
$loadinfolist' = loadinfolist \cup \{(serverip? \mapsto ld)\}$
$Re! = ScheduleTableFormed$

Different functions used to find the best destination address are: *getLoadDetails*, *isBetter*, and *theBestIP*. The *getLoadDetails* returns load details for the corresponding server address present in the *ScheduleTable.*

$getLoadDetails: SERVERADDRESS \rightarrowtail\!\!\!\!\!\!\rightarrow LoadDetails$

The *isBetter* function returns the better server address between two different servers based on the load information obtained from the *ScheduleTable.* The different load details used for comparison are percentage of *CPUAvailable*, percentage of free *MEMAvail*, length of the processor queue i.e. *QueueLEN* and the *DISTANCE* between the content router and the server.

```
$isBetter: LoadDetails \times LoadDetails \rightarrow\!\!\!\!\!+ BOOLEAN$
--------------------------------------------------
$\forall ld1, ld2: LoadDetails \mid (ld1, ld2) \in \mathrm{dom}\ isBetter$
  $\cdot\ isBetter\ (ld1, ld2) = True$
    $\Rightarrow ld1 . CpuInfo > ld2 . CpuInfo$
     $\vee ld1 . Dist < ld2 . Dist$
     $\vee ld1 . QueueLen < ld2 . QueueLen$
     $\vee ld1 . MemInfo > ld2 . MemInfo$
     $\vee ld1 . LoadDetails = ld2 . LoadDetails$
```

The next function, *theBestIP*, uses the *isBetter* function to find the best destination server for processing the user request. The inputs supplied for this function are two server addresses and the output obtained is the best server address.

```
$theBestIP: \mathbb{P}\ SERVERADDRESS \rightarrow\!\!\!\!\!+ SERVERADDRESS$
--------------------------------------------------
$\forall sa: \mathbb{P}\ SERVERADDRESS$
  $\cdot\ \exists Tbip: SERVERADDRESS \mid Tbip \in sa$
      $\cdot\ theBestIP\ (sa) = Tbip$
        $\Rightarrow (\forall ip: SERVERADDRESS \mid ip \in sa$
            $\cdot\ isBetter\ ((getLoadDetails\ (Tbip)), (getLoadDetails(\ ip)))$
              $= True)$
```

The next schema operation is *RewriteIPHeader*. The main function of the *RewriteIPHeader* schema is to rewrite the original packet's destination address with the new server address. The inputs for this operation are *newdestip?* and packet id (i.e. *pid?)*. The original packet's id is checked with the input pid?. If both ids are equal the packet's destination address is changed to the new server address. The result for this schema is *"DestAddressChanged"*.

RewriteIPHeader

$\Delta PacketCreation$
$\Xi ScheduleTable$
newdestip?: SERVERADDRESS
pid?: IDENTIFICATION
Re!: RESPONSE

pid? = id
newdestip? = theBestIP (dom *loadinfolist*)
destip = newdestip?
Re! = DestAddressChanged

The *CacheManager* schema maintains a list in the cache. The list has the resource name and best-selected server address.

CacheManager

cachelist: RESOURCENAME $\rightarrow\!\!\!\!\!\shortmid$ *SERVERADDRESS*

The initial list of the *CacheManager* is empty.

InitialCL

CacheManager

cachelist $= \varnothing$

The *UpdateCache* schema updates the *CacheManager's* list by adding the best server address and its resource name. The input supplied for this operation is *serverip?*. The *theBestIP* function is applied to select the best server address from the list maintained by the ScheduleTable. The resource name and the server address are updated in the *CacheManager's* list. The response from this operation is *"CacheUpdated"*.

UpdateCache

$\Delta CacheManager$
$\Xi ScheduleTable$
$data!: RESOURCENAME$
$serverip?: SERVERADDRESS$
$Re!: RESPONSE$

$\mathrm{dom}\ loadinfolist \in \mathrm{dom}\ theBestIP$
$serverip? = theBestIP\ (\mathrm{dom}\ loadinfolist)$
$cachelist' = cachelist \cup \{(data! \mapsto serverip?)\}$
$Re! = CacheUpdated$

Using the different operations defined in the system, the Content Router can be defined as follows.

$$\begin{aligned} ContentRouter \mathrel{\widehat{=}} & ((ResourceTable \wedge SystemStatusTable \wedge ProximityTable)\ ; \\ & UserRequest\ ;\ FindServerAddress\ ;\ FindSystemStatus \\ & FindDistance\ ;\ LoadDetails\ ;\ FormScheduleTable\ ; \\ & RewriteIPHeader\ ;\ UpdateCache) \end{aligned}$$

While some of the operations mentioned above are executed sequentially others are executed in parallel. When the system is started the *ResourceTable*, *SystemStatusTable*, and *ProximityTable* operations are executed in parallel. These three operations are executed continuously until the system is stopped. The rest of the operations are executed sequentially and are done based on the *UserRequest*.

4.3 Summary

This chapter presented a formal specification for the designed content-based router. Section 4.1 discussed the problem model and presented a short description. In Section 4.2 we presented the Z-specification and explained the functionality of each schema, and gave us examples of the different tables that were used. Chapter 5 presents an object model and explains the functionality of each of the components present in the newly designed content router.

Chapter 5

Object Model and Functionality

This chapter presents an Object Model for the designed content router and explains the different functionalities of the design. Developing a software system is becoming complex and expensive due to the change from single-tier to multi-tier architecture and distributed systems. To develop sophisticated software system one requires creativity, ability to learn and analyze the problem and should have knowledge or experience in different programming languages. To avoid the complexity and to maintain the quality and reliability of the system the concept of object orientation comes into existence. The object models in this thesis are developed using Unified Modeling Language (UML). The UML has many object-oriented notations, which is used to analyze and design sophisticated applications. The main reason for using UML for developing the object models is, it has many specialized notational elements, which supports complex applications. The different types of UML diagrams I have used in this thesis are: class diagram, activity diagram, sequence diagram and deployment diagram. Figure 5.1 shows the class diagram for content-based router.

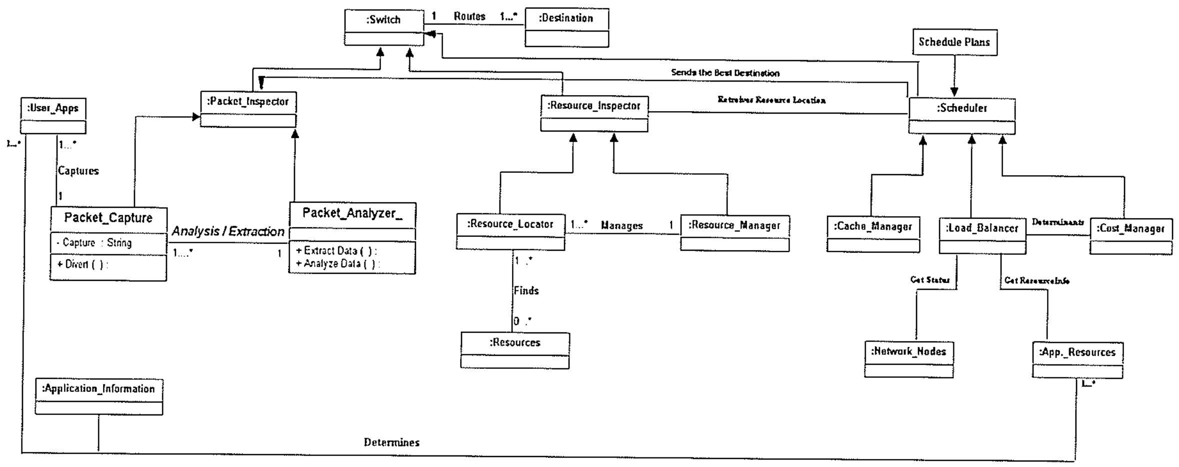


Figure 5.1 *Class diagram for Content-Based Router*

The class diagram in Figure 5.1 shows the different classes present in the application. It also specifies the relationship between different classes. While creating a large complex system, the application is divided into different modules. The different modules present in this thesis are Packet Inspector, Resource Inspector and Scheduler. Each module is further divided into sub-modules. Each module has it's own class diagram.

Figure 5.2 shows the activity diagram for content-based router. The activity diagram shows the different activities and flows of data or decisions between the activities. Activity diagram is used in workflow analysis. It is also called as flowchart. Activity diagram shows different activities handled by different objects. It can support parallel execution. Activity diagrams are used for detailed specification of complex systems with respect to implementation. Figure 5.3 shows the sequence diagram of the system. The sequence diagram shows the relationship between two different objects. Each object is represented as vertical lines and shows how messages are sent between two objects. The sequence diagram is also known as interaction diagram. The messages that are sent between two objects are also called as events. An event takes place only when the target object replies back to its message.

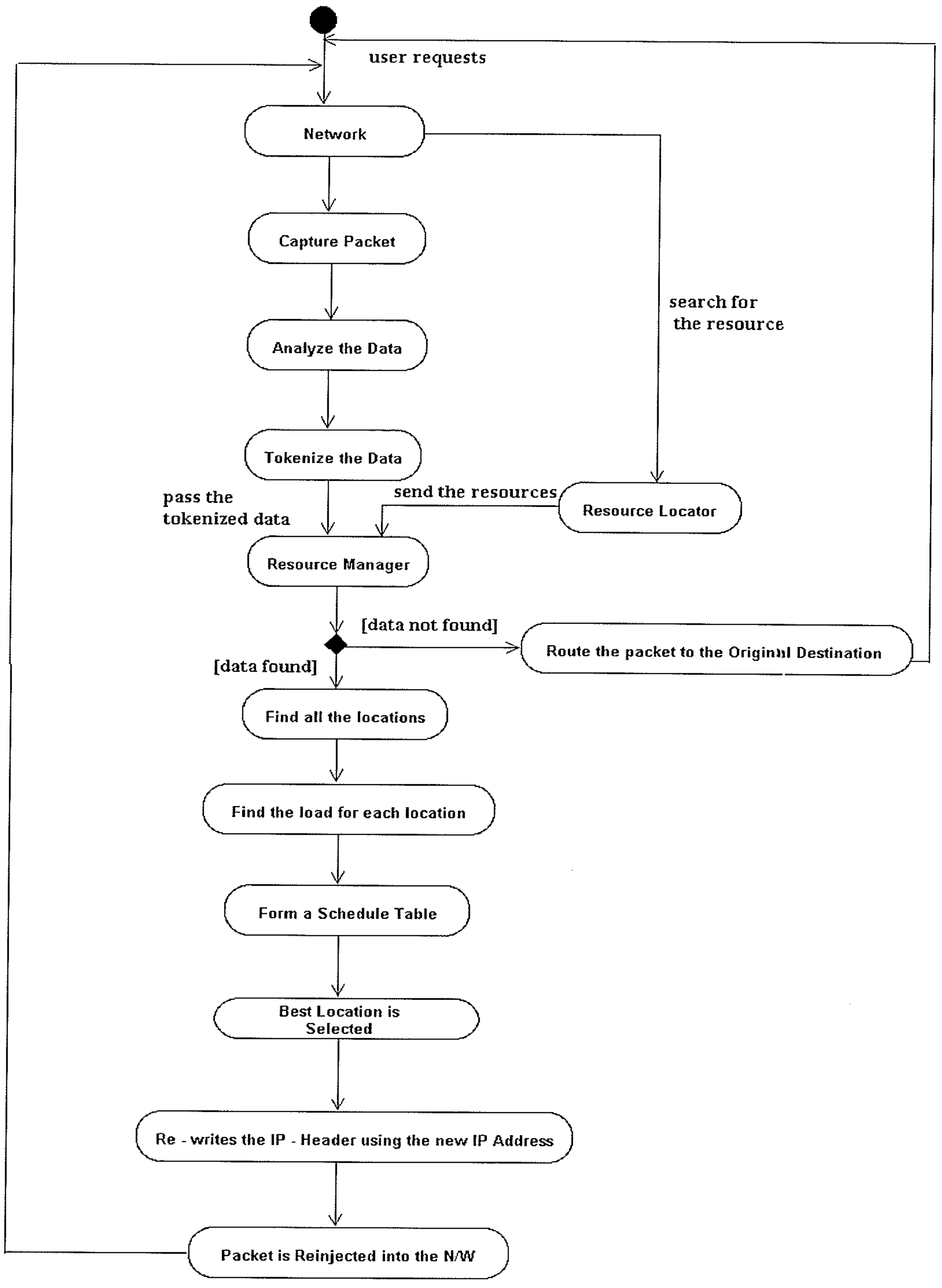


Figure 5.2 *Activity diagram for Content-Based Router*

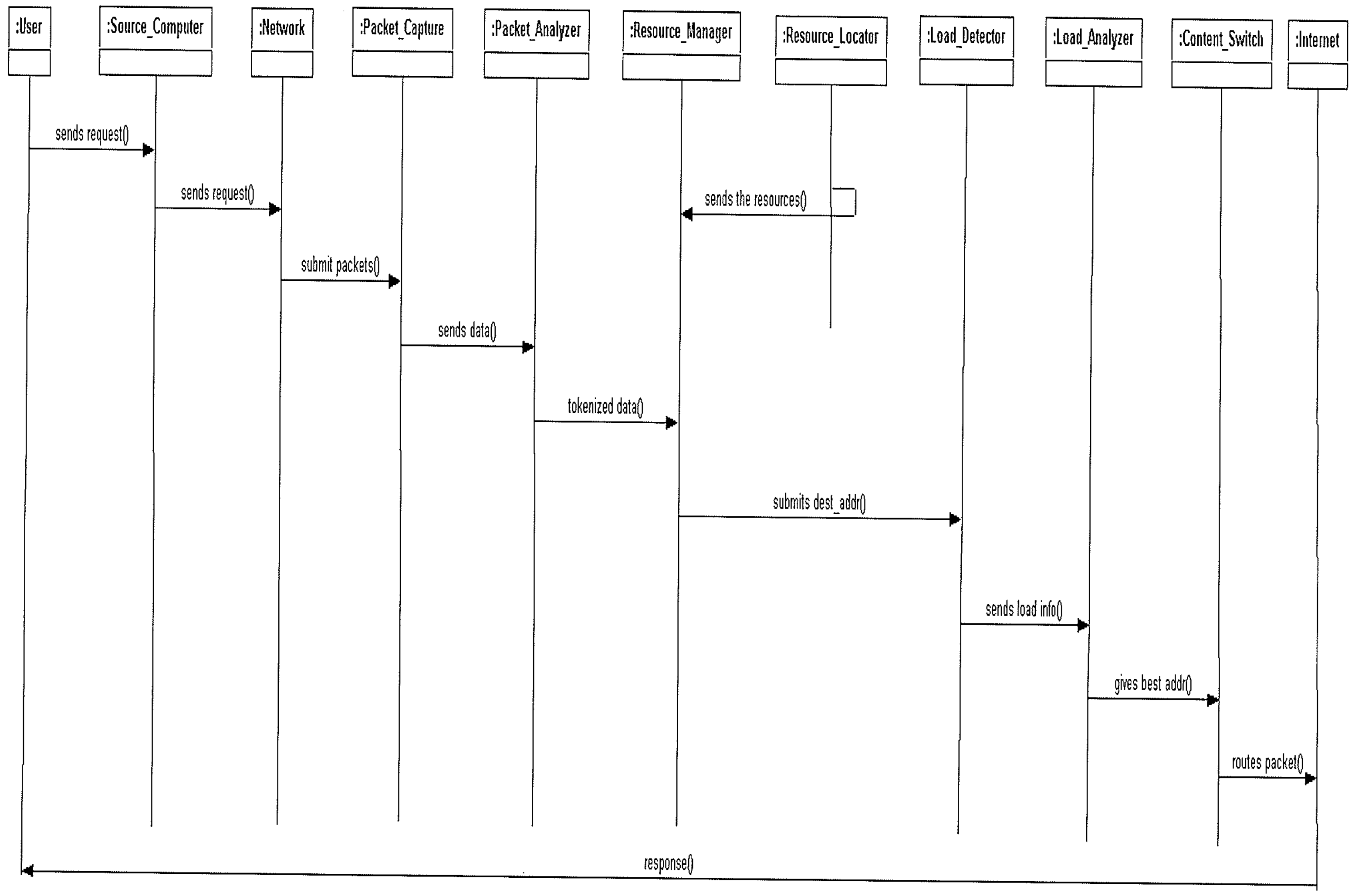


Figure 5.3 *Sequence diagram for Content-Based Router*

Figure 5.4 shows the deployment diagram for content-based router. The deployment diagrams are used to describe the architecture of the system. A three-dimensional box represents each node in deployment diagram. Each node represents different components of the system. The different nodes present in this system are the different clients, a network hub, which connects different computers together, a content router and different servers with different databases on it.

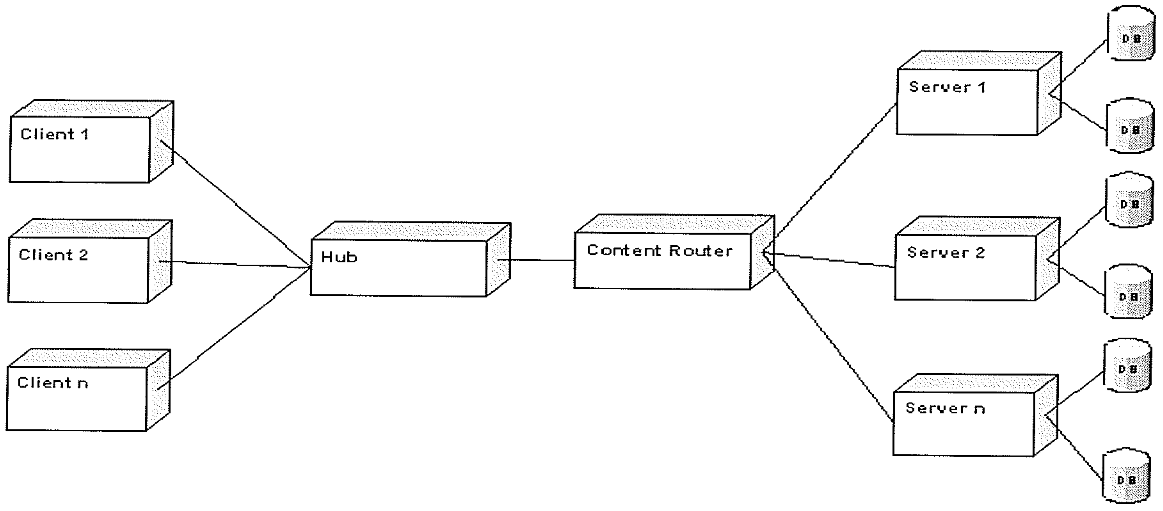


Figure 5.4 *Deployment diagram for Content-Based Router*

5.1 Packet Inspector

The *Packet Inspector* module enables the router to capture and extract the data in each packet of a user's request. This data is the content that is routed to the appropriate server at that moment based on a set of metrics. This component of the system intercepts the user's request data stream in the form of packets and then extracts the data content (i.e., the payload) it contains for routing. Figure 5.5 shows the class diagram for packet inspector.

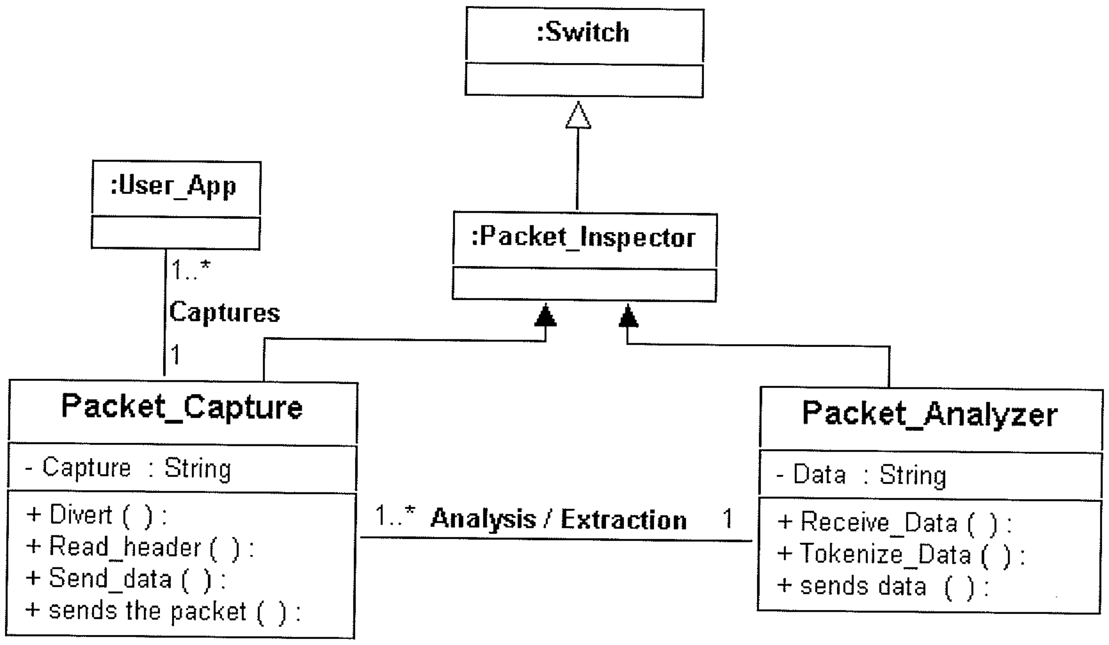


Figure 5.5 *Packet Inspector - Class Diagram*

5.1.1 Functionality

The Packet Capture and Packet Data Extraction / Analysis are the two sub-components of the Packet Inspector. The Packet Inspector unit captures and extracts the data in each packet of a user request. This extracted data is used for routing the packet to the appropriate server. Figure 5.6 gives the Sequence diagram for the Packet Inspector. The Packet Capture component takes care of capturing the packet and sending the data to the Packet Analyzer. The Packet Capture component opens a socket connection and listens for the packet that flows in the network.

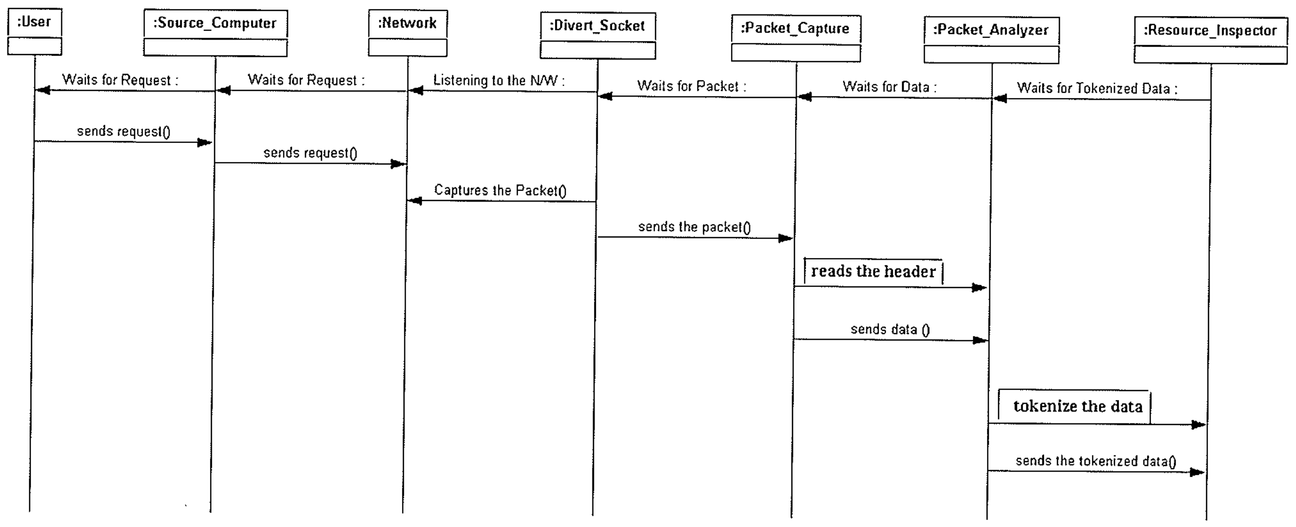


Figure 5.6 *Packet Inspector - Sequence Diagram*

When the user sends in a request the socket grabs or captures the packet, and stops the packet flow from the current node or hop to the next node. The Packet Capture collects the captured packet, scans the header and the data field. By scanning the header and data field the Packet Capture finds the source address, destination address and the data in the packet. If the data field is empty the packet is discarded without any further processing. If the packet contains data, it is forwarded to the Packet Analyzer for processing. The Packet Analyzer converts the extracted data from the machine code to readable string format. The converted data is tokenized and a keyword or set of keywords is selected, which is sent to the next component of the system, the Resource Inspector. Algorithm 5.1 and Figure 5.7 gives the pseudo code and Activity diagram for the Packet Inspector. Thus, the Packet Inspector intercepts the users request data stream in the form of packets and then extracts the data content, which is used for routing.

Algorithm 5.1 *Packet Inspector*

INPUT: User Request;

OUTPUT: Tokenized data in string format;

```
    WHILE (Network is active) DO
```

```
            Open a socket connection S;
            IF (S = -1) THEN
                    Socket open error;
                    Exit the system;
            IF (S >= 0) THEN
                    Open a divert socket;
                    Listen to a port for receiving the packets;
            FOREACH packet DO
            SWITCH (ether_type) IN

            CASE IP Packet:
                    Divert the packet to the user level;
                    Read the header and data;
                    IF (data = null) THEN
                            Discard the packet;
                    ELSE convert the data to string format;
                            Tokenize the data;
            CASE ARP Packet:
                    Read the header;
                    Forward the packet to the original destination;

            CASE RARP Packet:
                    Read the header;
                    Forward the packet to the original destination;

            OTHERWISE:
                    IF (unknown packet type) THEN
                            Forward the packet to the original destination;

            END {SWITCH};
    END {WHILE};

End of Algorithm;
```

5.1.2 Implementation Strategies

The Packet Inspector component is implemented in C and Java. The components implemented in C are integrated into the other parts using Java's Native Interface facility. The protocol used for capturing the packets is the divert socket. The libpcap library file in C was used to capture the packets. The drawback in using libpcap is, it just gives a copy of the packet and forwards the packet to the next node. This drawback is avoided in divert sockets, because it actually grabs the packet from the network. The content of the packet is converted and analyzed using Java because it supports many classes and methods than any other language.

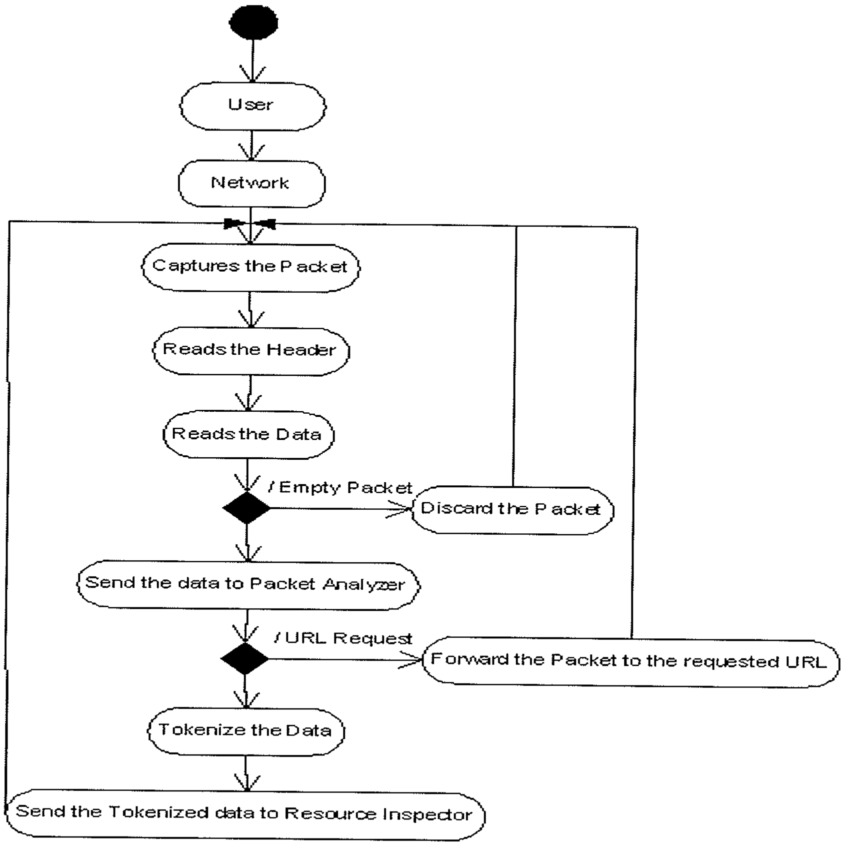


Figure 5.7 *Packet Inspector - Activity Diagram*

5.2 Resource Inspector

A core component of the system is the *Resource Inspector*. The main job of the Resource Inspector is to assemble vital information about the resources available in the system for ease of access and fast decision-making. To implement this component, we adopted intelligent mobile agent technology. Mobile agents are suitable because they enable us to seamlessly and transparently assess servers (at remote locations) and retrieve appropriate data of interest. The agents only need to know the address (IP address or full domain name) of the resource and a known set of database types. The agents can retrieve the metadata of each database, such as the name of the schemas, the description of the schemas, and table definitions, etc. This information is necessary to make informed judgements on where to find the available resources for the application. The databases are transparent to the system. Figure 5.8 shows the class diagram for resource inspector.

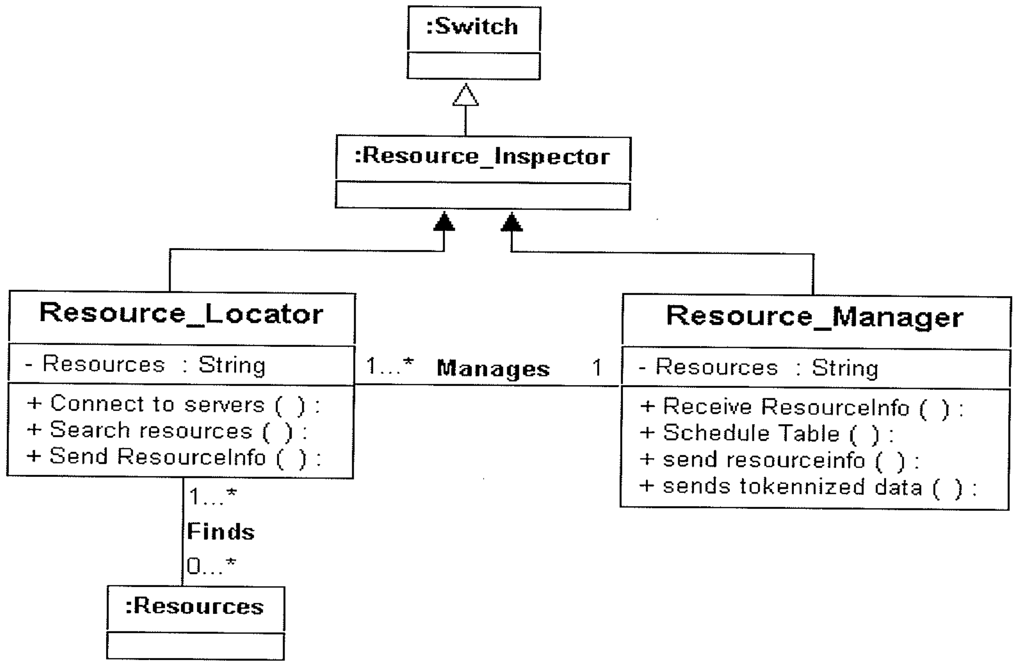


Figure 5.8 *Resource Inspector - Class Diagram*

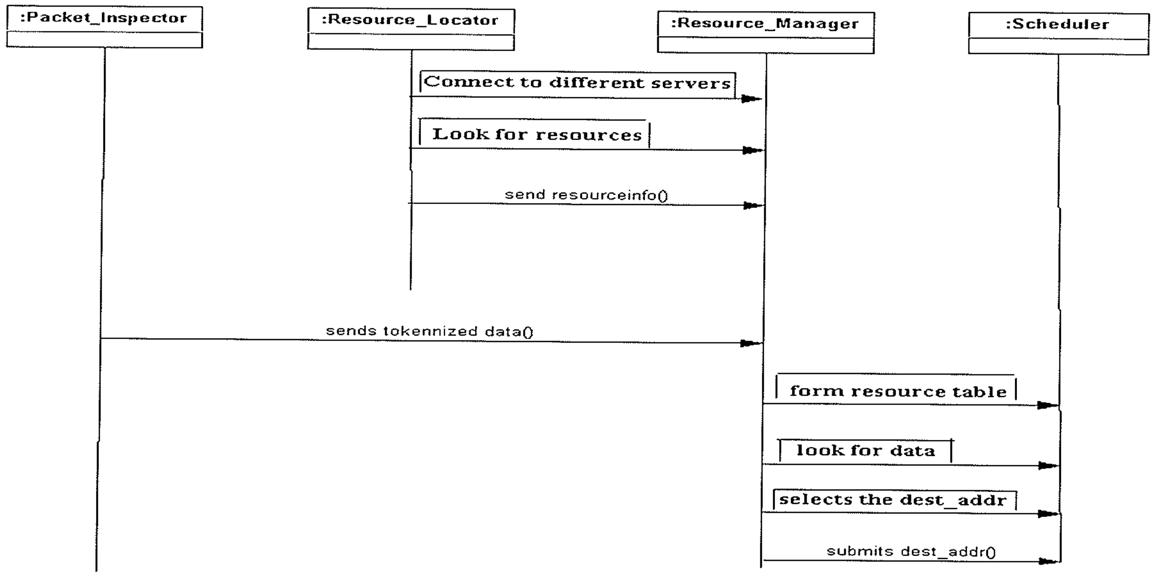


Figure 5.9 *Resource Inspector - Sequence Diagram*

5.2.1 Functionality

The Resource Locator and Resource Manager are the two sub components of the Resource Inspector. The main job of the Resource Inspector is to assemble vital information about the resources available in the system for ease of access and fast decision making. Figure 5.9 gives the Sequence diagram for the Resource Inspector. The Resource Locator collects the resource information. When the switching unit is started, the Resource Locator creates resource agents. These agents are capable of moving from one location to other location. Because of their mobile property, these agents are called Mobile Agents. The Mobile Agents are sent to different machines to look for resources. The resources for E-Commerce applications are often stored in databases at participating servers. The resources are heterogeneous because they are build using different database systems (e.g., Microsoft Access, Oracle, SQL Server, DB2, Sybase, etc). The Resource Agents enter the appropriate designated server and retrieves the data of interest. The agents extract the metadata of each database, such as the name of the schemas, the description of the schemas, and table definitions etc. These informations are given to the Resource Manager to make informed judgements on where to find the available resources for the application. Based on the metadata information and the server address, the Resource Manager collects resource information about the number of databases available in the system, the address of these databases, and permissions on the databases and stores the collected data in a resource table. Algorithm 5.2 and 5.3 gives the pseudo code for Resource Locator and Resource Manager. Figure 5.11 gives the Activity diagram for the Resource Inspector.

Algorithm 5.2 *Resource Locator*

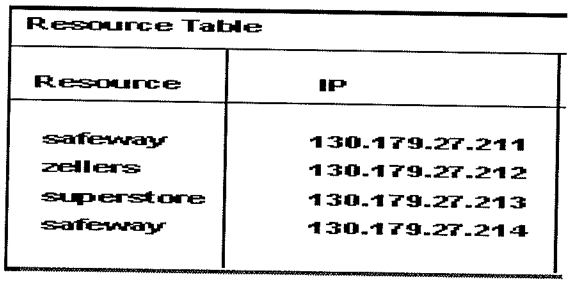
```
INPUT: Server addresses;
OUTPUT: Resource information of various servers;

// Abbreviations used and there corresponding meaning.

        RM: Resource Manager;

FOREACH server DO
        Create resource agents;
WHILE (network is active) DO
        Open a connection with all servers;
        IF (server is active) THEN
                Send the resource agents to the assigned server;
                Collect the resource information for each server;
                Exit the system;
        ELSE wait for active connection with the server;
        END {IF};
        Send all the collected resource informations to RM;
END {WHILE};
End of Algorithm;
```

Algorithm 5.3 *Resource Manager*

```
INPUT: Tokenized data from Packet Inspector;
          Resource Informations from Resource Locator;
OUTPUT: Server address or addresses for the tokenized data;

// Abbreviations used and there corresponding meaning.

        RT: Resource table;
        SA: Server address or addresses;
        TD: Tokenized data;
        DL: Data Location;

RT is formed using the resource informations;

FOREACH tokenized data DO
        Look for SA;
WHILE (network is active) DO
        Search for TD in RT;
        IF (TD not found in RT) THEN
                Forward the packet to the original destination;
        ELSEIF (TD found in RT) THEN
                Find SA;
                Form a DL table using the SA;
                Send the DL table to the Scheduler unit;
        END {IF};
END {WHILE};
```

End of Algorithm;

While collecting the resources in the resource table the resource informations are also copied into a file as backup information. The advantage of following this process is, even when the system is down or switched off all the informations are stored, which can be used as soon as the system is recovered. The resource table has tow columns and n - number of rows. The Resource table is shown in Figure 5.10.



Resource Table

Resource	IP
safeway	130.179.27.211
zellers	130.179.27.212
superstore	130.179.27.213
safeway	130.179.27.214

Figure 5.10 *Resource Table*

The two columns in the resource table are the server address and the resources available in the server. The resource table is scanned for the tokenized data obtained from Packet Inspector to find the appropriate server or servers for processing the user request. The obtained server address or addresses are stored in a Data Location table. The data location table is shown in Figure 5.12. The Data Location table is sent to the Scheduler unit for further processing.

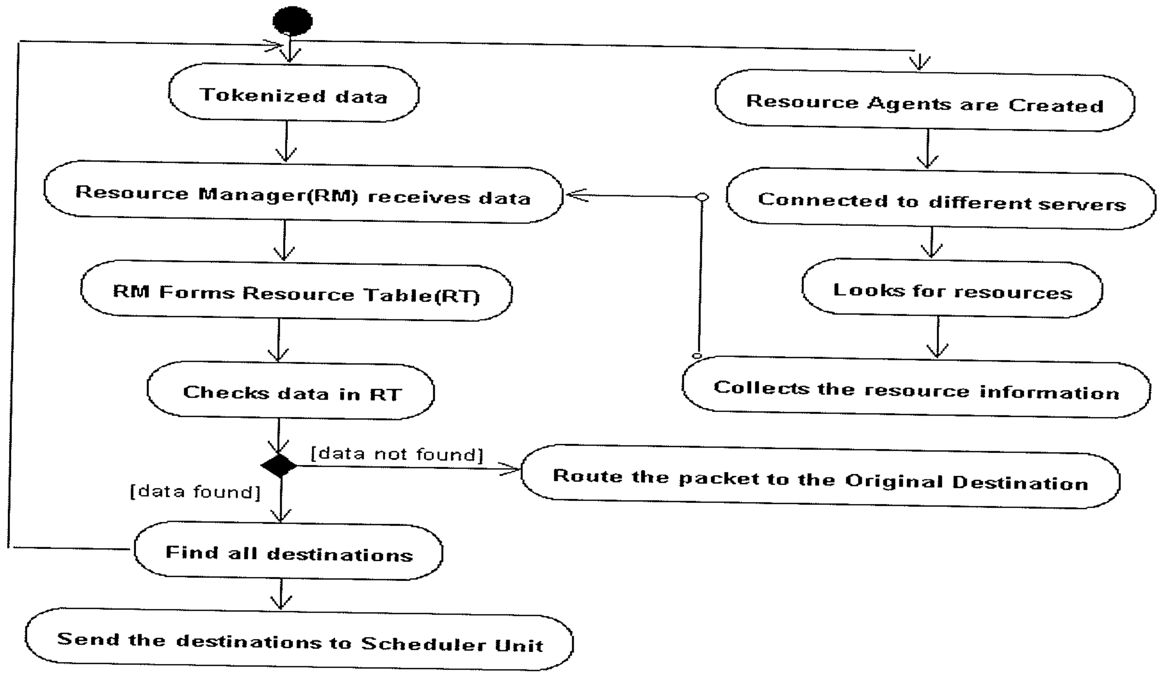
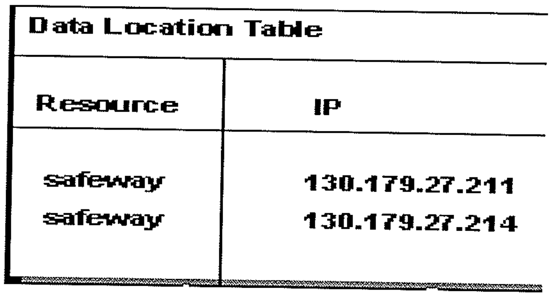

Tokenized data
Resource Manager(RM) receives data
RM Forms Resource Table(RT)
Checks data in RT
[data not found]
[data found]
Route the packet to the Original Destination
Find all destinations
Send the destinations to Scheduler Unit
Resource Agents are Created
Connected to different servers
Looks for resources
Collects the resource information

Figure 5.11 *Resource Inspector - Activity Diagram*

5.2.2 Assumptions

The implementation assumes that

- All Server Addresses are known.
- Permissions are granted on the servers.
- Data Source Names for all the databases are known.
- The databases are transparent to the system.


Data Location Table

Resource	IP
safeway	130.179.27.211
safeway	130.179.27.214

Figure 5.12 *Data Location Table*

5.2.3 Implementation Strategies

The Resource Inspector component is implemented using Java.

5.3 Scheduler Unit

The *Scheduler Unit* is a major part of the system, uses the information assembled by the Resource Manager to facilitate content-based routing. It is responsible for scheduling and allocating transactions to the various servers for execution based on the current processing / work load information of each server. This unit answers questions such as: how busy is each server and which server can process the request in the shortest time. Figure 5.13 gives the class diagram for scheduler.

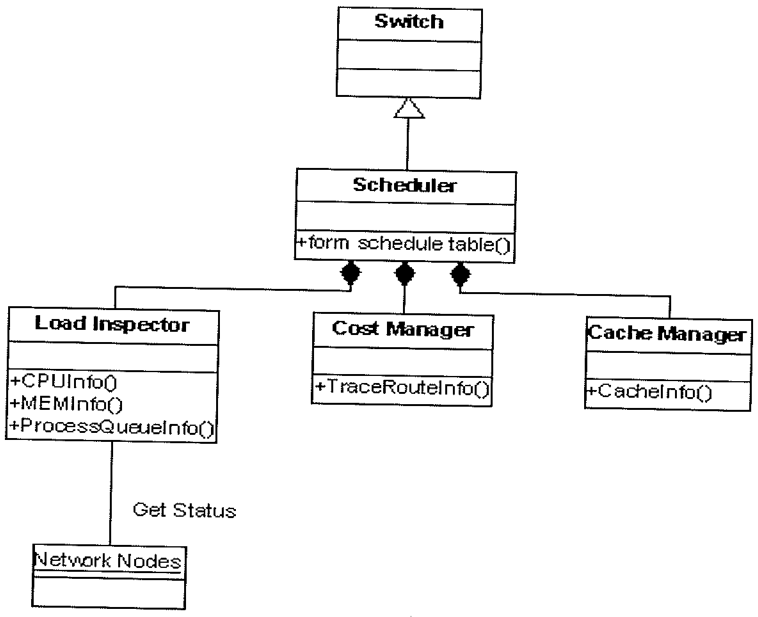


Figure 5.13 *Scheduler Unit - Class Diagram*

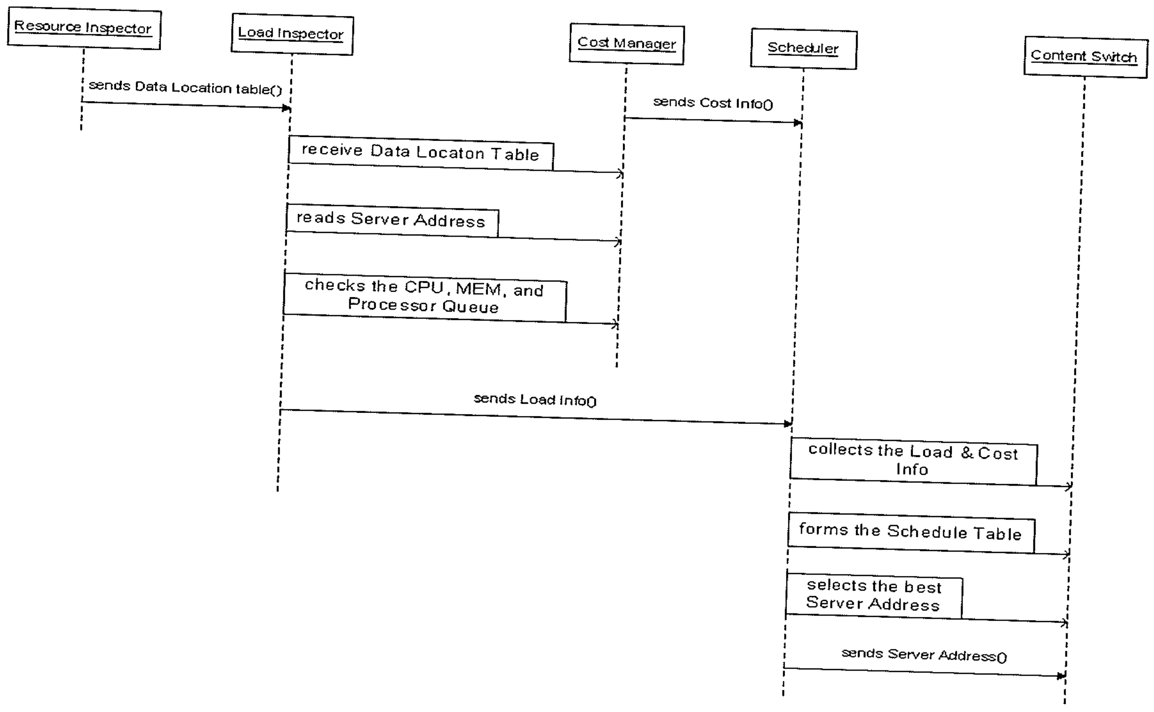


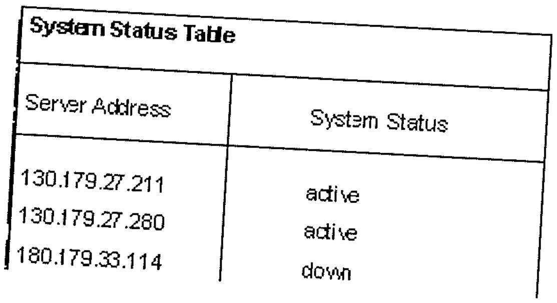
Figure 5.14 *Scheduler Unit - Sequence Diagram*

Functionality

The different components of the Scheduler Unit are the Load Inspector, Cost Manager, Cache Manager and the Scheduler. The Scheduler selects a best and efficient destination address based on a set of metrics. The sets of metrics are the load on the server and the distance between the client and the server. The following section discusses the functionality of each component elaborately. Figure 5.14 gives the sequence diagram of the Scheduler Unit.

5.3.1 Load Inspector

The Scheduler receives the Data Location Table from the Resource Inspector. For each entry in the table the Load Inspector creates Load Detector Agents. The agents are capable of moving from one location to another. Each entry in the Data Location Table has a server address. The Detector Agent reads the server address and enters the appropriate server to retrieve the Load information. Before entering the server the Detector Agent checks for the status of the server from the System Status Table (SST). The SST has the status information of all the participating servers. Figure 5.15 shows the SST.



System Status Table

Server Address	System Status
130.179.27.211	active
130.179.27.280	active
180.179.33.114	down

Figure 5.15 *System Status Table*

If the system is active the agent checks the percentage of CPU available for the next process, free Memory available and the length of the Processor Queue to find the total number of jobs waiting to get processed by the server. If the server is down or inactive the Detector Agent ignores the server and looks for the next Server Address in the Data Location table. The Detector Agent collects the load information and sends it to the Scheduler for further processing. Algorithm 5.4 gives the pseudo code and Figure 5.16 gives the activity diagram for the Load Inspector.

Algorithm 5.4 *Load Inspector*

INPUT: Data Location table from Resource Inspector;

OUTPUT: Load information of all Servers in Data Location table;

// **Abbreviations used and there corresponding meaning.**

```
    SA: Server address;
    DL: Data Location;
    MEM: Memory;
    PQ: Processor Queue;

FOREACH entry in DL table DO
        Read SA;
WHILE (system is active) DO
```

```
        IF (first row not empty) THEN
                Check the CPU status for the SA;
                Check the MEM status for the SA;
                Check the PQ length for the SA;
        END {IF};
        Collect all the above information;
        Send the load information to the Scheduler;
END {WHILE};
```

End of Algorithm;

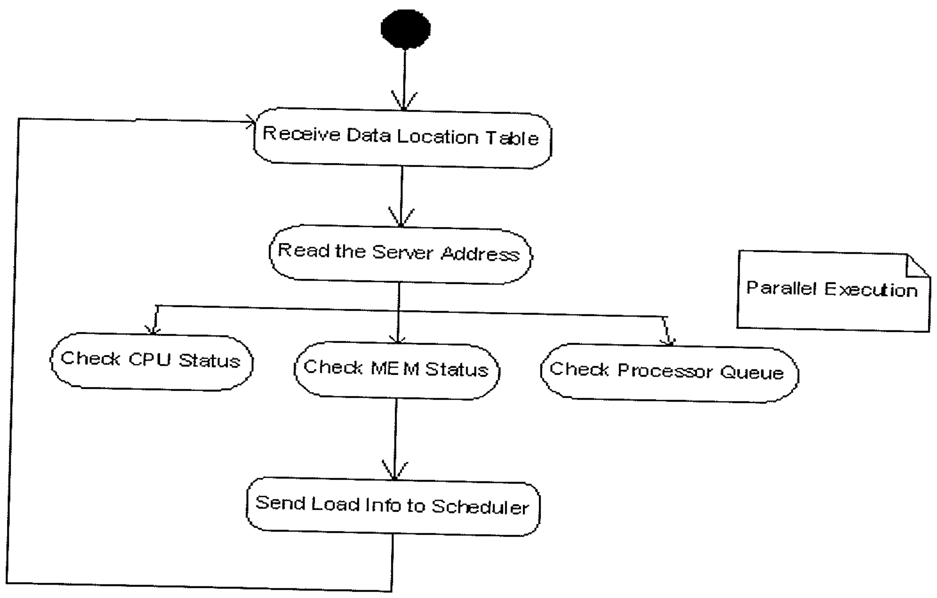


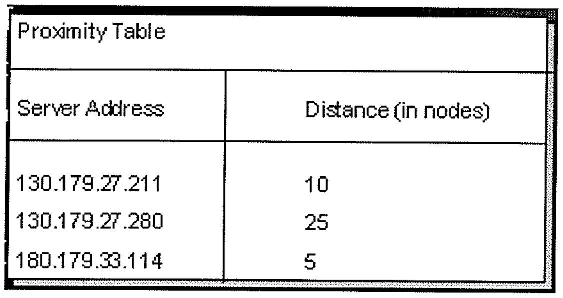
Figure 5.16 *Activity Diagram for Load Inspector*

Implementation Strategies

The Scheduler is implemented using Java. This component is implemented using Java Remote Method Invocation (RMI). The other approaches for implementing this module are Java Aglets and Simple Network Management Protocol (SNMP). In all the three approaches a Server should be running for the Resource Agents to collect the Resource information. The SNMP approach is very similar to the Remote Method Invocation. The SNMP server is same as the RMI Server. The SNMP is the standard protocol used for remote communication. The Java Aglets has its own Tahiti Server, which is built in with the Aglets Kit that has to be installed to use the Aglets. Aglets can create Mobile Agents that can roam from one machine to another. The advantage of using RMI is, we can have our own specification in creating the Server, which supports our application reducing the workload on the Server. In the case of Aglets and SNMP they have a built in Server, which is created to support all the applications. This increases the workload on the Server.

5.3.2 Cost Manager

The next component in the Scheduler unit is the Cost manager. Cost Manager finds the distance between the client and the server. The Cost Manager creates a simple traceroute procedure, which is used to find the total number of hops, or nodes in between the client and the given server address and form a Proximity Table. Figure 5.17 shows the Proximity Table. The Cost Manager reads the Data Location Table. Each row



Proximity Table

Server Address	Distance (in nodes)
130.179.27.211	10
130.179.27.280	25
180.179.33.114	5

Figure 5.17 Proximity *Table*

in the table is scanned for server address. For each scanned Server address, the distance information is obtained by looking into the Proximity Table. The distance information is sent to the Scheduler for further processing. Algorithm 5.5 gives the pseudo code and Figure 5.18 gives the activity diagram for Cost Manager.

Algorithm 5.5 *Cost Manager*

INPUT: Data Location table from Resource Inspector;

OUTPUT: Number of nodes in between the content switch and each Server in Data Location table;

// Abbreviations used and there corresponding meaning.

```
        SA: Server address;
        DL: Data Location;

**FOREACH** entry in DL table **DO**
        Read SA;
**WHILE** (system is active) **DO**
        **IF** (first row not empty) **THEN**
                Find the total number of nodes present in between the switch and the
                given server;
        **END {IF}**;
        Send the information to the Scheduler;

**END {WHILE}**;

**End of Algorithm;**
```

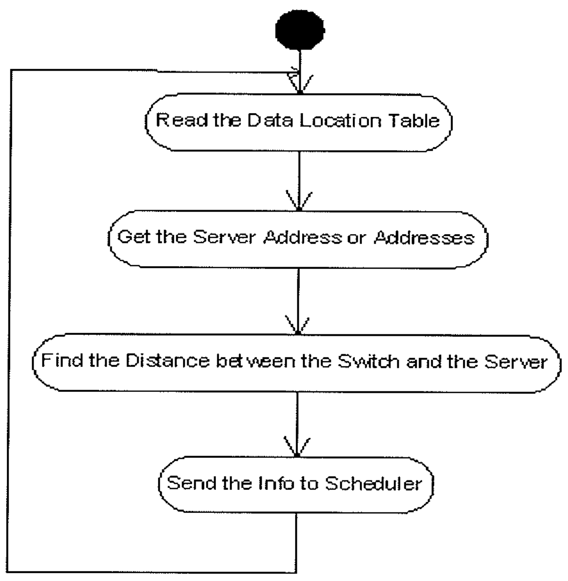
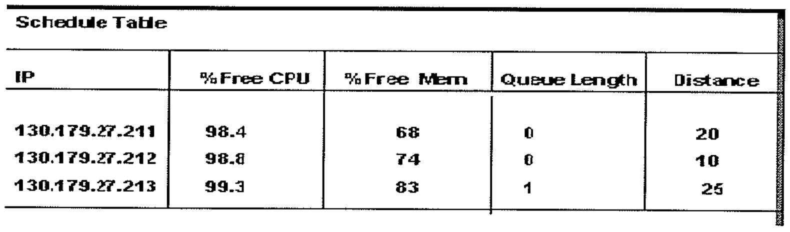

Read the Data Location Table
Get the Server Address or Addresses
Find the Distance between the Switch and the Server
Send the Info to Scheduler

Figure 5.18 *Activity Diagram for Cost Manager*

The Cost Manager is implemented using Java.

5.3.3 Scheduler

The next important component is the Scheduler. The Scheduler selects the best and efficient server address for routing the user request. The Scheduler collects the information from the Load Inspector and Cost Manager. Based on the collected information a Schedule table is formed. The Schedule table is shown in Figure 5.19.



Schedule Table

IP	%Free CPU	%Free Mem	Queue Length	Distance
130.179.27.211	98.4	68	0	20
130.179.27.212	98.8	74	0	10
130.179.27.213	99.3	83	1	25

Figure 5.19 *Schedule Table*

Algorithm 5.6 gives the pseudo code and Figure 5.20 gives the activity diagram for Scheduler.

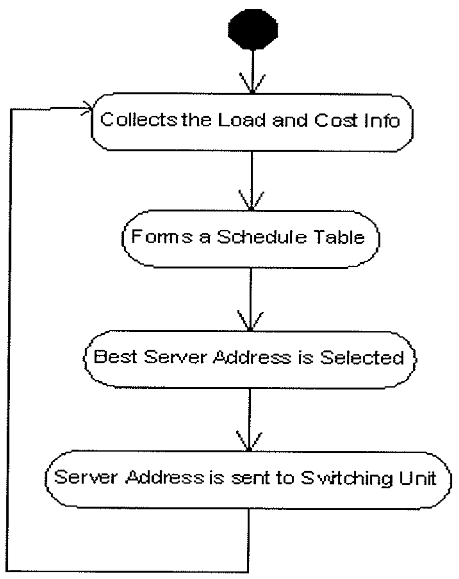

Collects the Load and Cost Info
Forms a Schedule Table
Best Server Address is Selected
Server Address is sent to Switching Unit

Figure 5.20 *Activity Diagram for Scheduler*

Algorithm 5.6 *Scheduler*

INPUT: Load Information from Load Inspector;
Cost Information from Cost Manager;

OUTPUT: Server Address for Routing user request;

// Abbreviations used and there corresponding meaning.

SA: Server address;
ST: Schedule Table;

WHILE (system is active) **DO**

```
        Collect all the information;
        Form a ST;

        Best and Efficient SA is selected based on set of metrics;
        Send the selected SA to Switching Unit;
**END {WHILE};**
```

End of Algorithm;

An efficient server address is selected from the Schedule table based on algorithm 5.7. The runtime for this algorithm is $O(n^2)$. The selected server address is sent to the switching unit for routing the user request.

Algorithm 5.7 *Selecting the Best Server Address*

INPUT: Schedule Table formed by Scheduler;

OUTPUT: Best and Efficient Server Address is selected;

```
SA: Server address;
ST: Schedule Table;
CPU: % CPU Available;
MEM: %Memory Available;
QL: Queue Length;
DIST: Distance between the switch and the server;

FOREACH columns in ST assign different arrays
DO
{
    Assign the 1st row element of each array to a temporary variable T;
    Compare the T row elements with the next row (N) in ST;
    IF ((|diff (T (CPU), N (CPU)|) > 0.5) THEN
    {
        IF (T (DIST) < N (DIST)) THEN
        {
            T'th row elements are selected and the SA is selected as best destination
            Address;
        }
        ELSE
        {
            Select the N row elements and assign SA as best destination
            Address;
        }
        END {IF};
        IF (T (DIST) = = N (DIST)) THEN
        {
            The server, which has more CPU available, is selected as best destination
            address;
        }
ELSE (ignore the CPU available and compare the DIST)
{
        IF (T (DIST) < N (DIST)) THEN
        {
                T'th row elements are selected and the SA is selected as best destination
                Address;
        }
        ELSEIF (T (DIST) > N (DIST)) THEN
        {
                Assign the temporary row to the next row elements and select
                the SA;
        }
        ELSE (ignore the DIST and compare the QL)
        END {IF};
        IF (T (QL) < N (QL)) THEN
        {
                T'th row elements are selected and the SA is selected as best destination
                Address;
        }
        ELSEIF (T (QL) > N (QL)) THEN
        {
                Assign the temporary row to the next row elements and select
                the SA;
        }
        ELSE (ignore the QL and compare the MEM)
        END {IF};
        IF (T (MEM) > N (MEM)) THEN
        {
                T'th row elements are selected and the SA is selected as best destination
                Address;
        }
        ELSEIF (T (MEM) < N (MEM)) THEN
        {
                Assign the temporary row to the next row elements and select
                the SA;
        }
        ELSE (ignore the MEM and find which server has more CPU available among
                the two rows);
        END {IF};
        IF (T (CPU) > N (CPU)) THEN
        {
                T'th row elements are selected and the SA is selected as best destination
                Address;
        }
        ELSEIF (T (CPU) < N (CPU)) THEN
        {
                Assign the temporary row to the next row elements and select
                the SA;
            }
            ELSE (select any row among the two compared rows and select the SA as the
                best destination address);
            END {IF};
        }
        END {IF};

        UNTIL all rows are compared;
    END {DO};
    End of Algorithm;
```

Java is used for implementing this component.

5.3.4 Cache Manager

The next component in the Scheduler Unit is Cache Manager. It is a separate component inside the Scheduler Unit. The main functionality of the Cache is to get the best and efficient destination address from the Scheduler and puts it into the cache with the corresponding data of interest for that server. When the request comes in from the client the router checks the cache for the requested data and its corresponding Server address. If the data is cached the router picks up the Server address and sends it to the Scheduler Unit for further processing. If the data is not available in the cache the router sends the tokenized data to the Resource Inspector to obtain an appropriate server address. This component is implemented in Java. The Cache is maintained in two different ways. The Surrogate Server or just a file can be maintained as a cache. Surrogate Server is similar to a cache where, the most frequently requested data is stored. The storage capacity in this server is very huge when compared to a file. In my thesis I am just using a file as my cache.

5.4 Summary

This chapter presented an object model, sequence diagrams and activity diagrams for the different components of the designed content router. Section 5.1 to 5.3 explains the functionality and presents the algorithm and implementation strategies for the different components of the content router. Chapter 6 presents some of the screenshots and explains them with a user scenario.

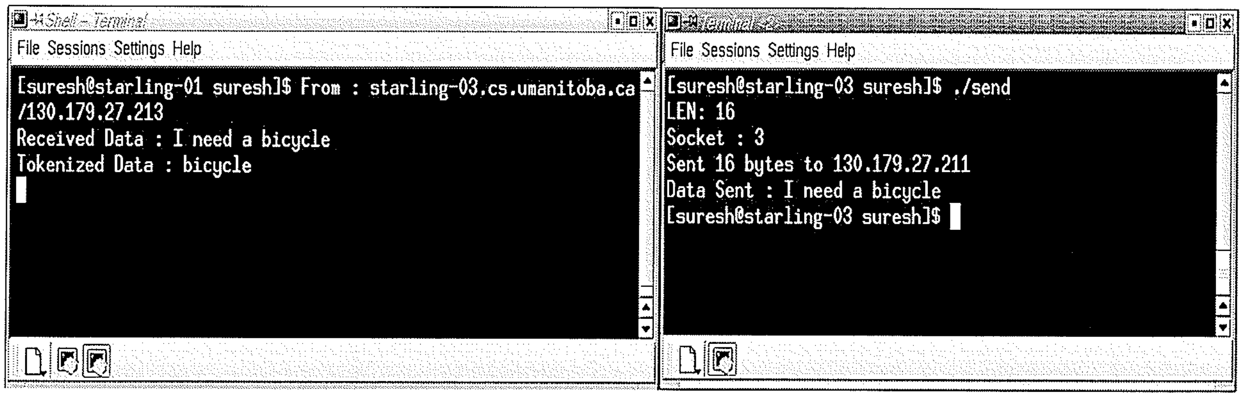
Chapter 6

Implementation

This chapter discusses different implementation strategies and explains different runtime screenshots of the implementation of the content-based router design. The different screenshots shown in this chapter are captured by running different modules of the implementation separately in order to show all the implemented components of the designed content router. Section 6.1 gives an overview of the overall system and Section 6.2 presents the user scenario of the full working model. Section 6.3 shows the implementation results for the designed content router and Section 6.4 presents a brief summary of the chapter.

6.1 System Overview

Figure 6.1 shows the screenshot of user request and how packet inspector captures the request.



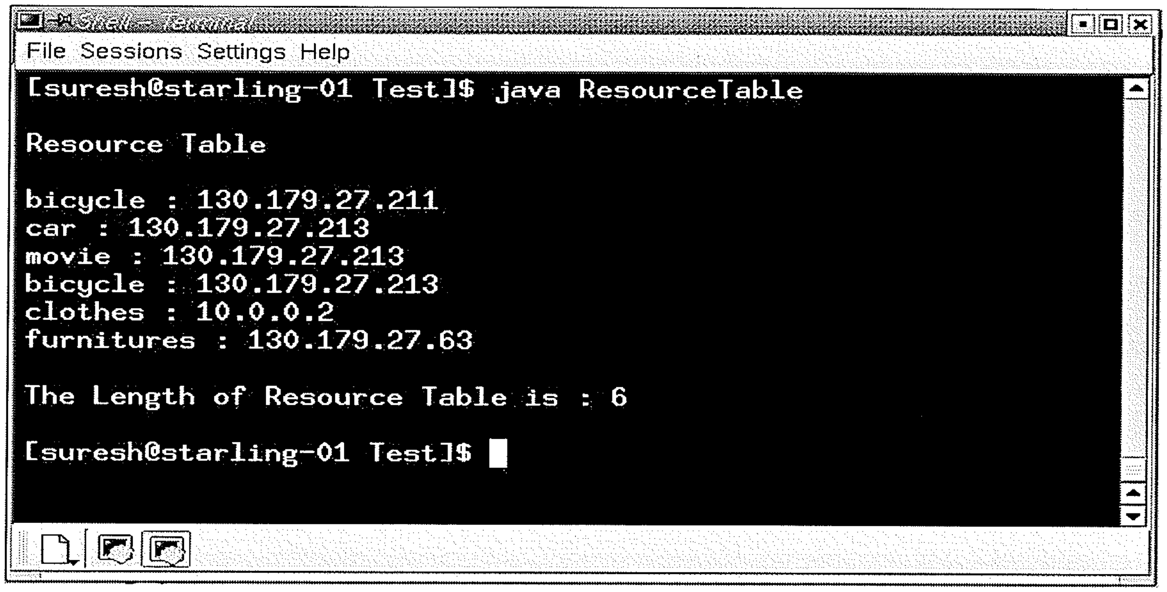
```
[suresh@starling-01 suresh]$ From : starling-03.cs.umanitoba.ca
/130.179.27.213
Received Data : I need a bicycle
Tokenized Data : bicycle
```

```
[suresh@starling-03 suresh]$ ./send
LEN: 16
Socket : 3
Sent 16 bytes to 130.179.27.211
Data Sent : I need a bicycle
[suresh@starling-03 suresh]$
```

Figure 6.1 *User Request and Tokenized Data*

Recall the Packet Inspector present in the content router has two components, packet capture and packet analyzer. The packet capture component captures the user request and extract the data sent by the user. From Figure 6.1, shows an example of how the user requests is sent and captured. The packet capture captures the packet and finds the address of the user and extracts the data sent by the user. The packet analyzer analyzes the data and tokenizes the data to select the keywords for finding a suitable server for processing the client's request. Figure 6.1 shows an example of how the tokenized data is extracted from the user request. The tokenized data is sent to the next module for further processing.

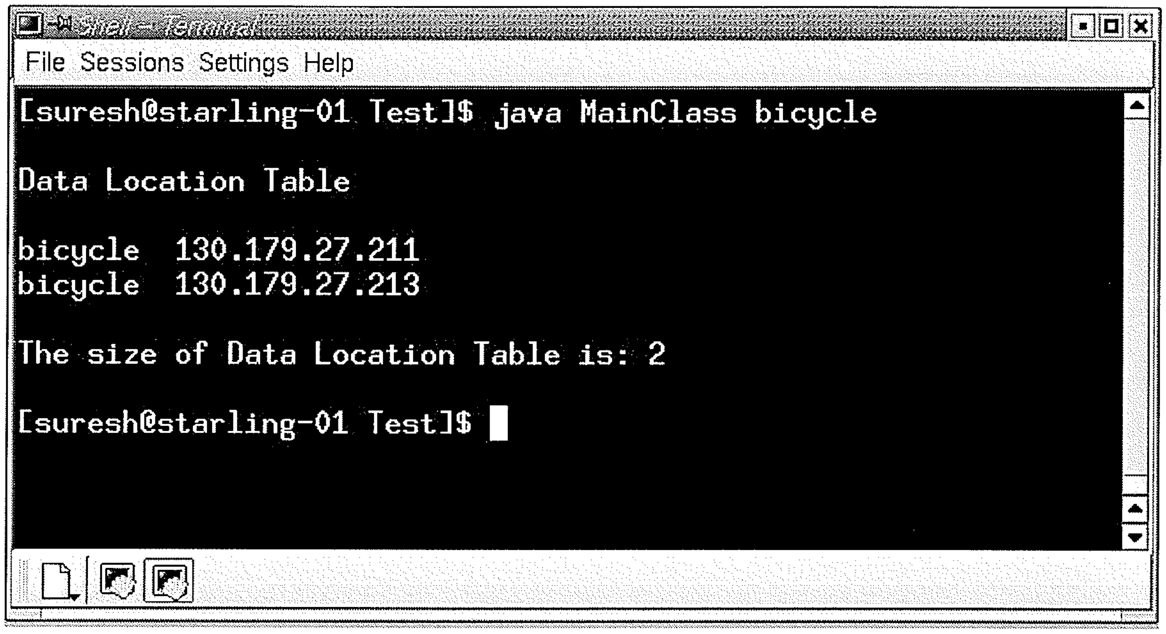
Figure 6.2 shows the structure of the resource table. The resource table is maintained by the resource inspector component of the content router. The resource table has two columns, resource name and server address. The resource locator, a module in the resource inspector, looks for different resources in the participating servers and collects the information. The collected information is given to the resource manager module present in resource inspector. The resource manager stores the collected data in a resource table maintained by it.



```
[suresh@starling-01 Test]$ java ResourceTable

Resource Table

bicycle : 130.179.27.211
car : 130.179.27.213
movie : 130.179.27.213
bicycle : 130.179.27.213
clothes : 10.0.0.2
furnitures : 130.179.27.63

The Length of Resource Table is : 6

[suresh@starling-01 Test]$
```

Figure 6.2 *Screenshot-Resource Table*

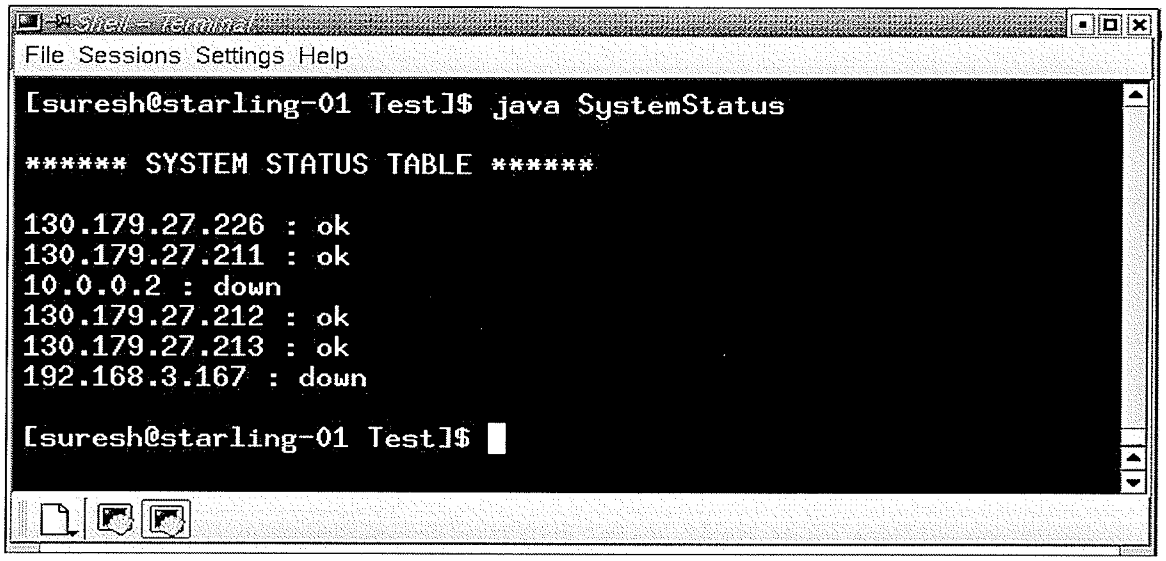
The resource table is scanned for the tokenized data obtained from the packet inspector to find different servers suitable for processing the user request. The scrutinized server address or addresses are stored in the data location table maintained by the scheduler component of the content router. Figure 6.3 shows an example of the data location table. The data location table has two columns tokenized data and server address. The content of the data location table changes dynamically each time based on the tokenized data forwarded by packet inspector. The main reason for having a data location table is to reduce the processing time of the content router for each user request, by avoiding the scanning of the resource table more than once for finding the load details. The data location table is sent to the scheduler unit for further processing.



```
[suresh@starling-01 Test]$ java MainClass bicycle

Data Location Table

bicycle  130.179.27.211
bicycle  130.179.27.213

The size of Data Location Table is: 2

[suresh@starling-01 Test]$
```

Figure 6.3 *Screenshot-Data Location Table*

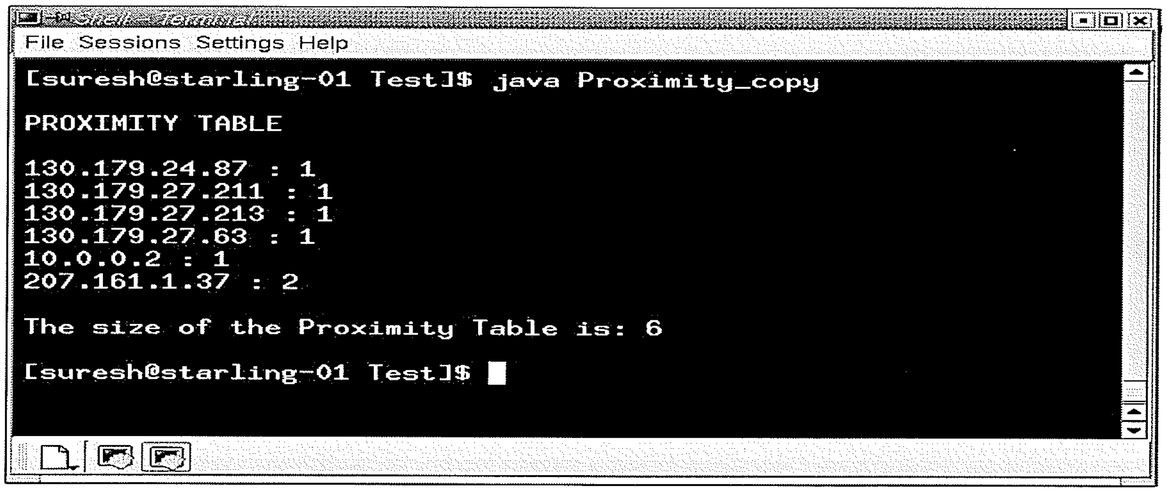
Figure 6.4 shows the system status table maintained by the scheduler unit. The scheduler is one of the core components of the routing system. It uses the information collected from the resource manager to facilitate content-based routing. The system status table contains the status of the participating servers, whether the participating servers are active or down. The scrutinized server address or addresses present in the data location table is checked with the system status table by the scheduler. If any of the server or servers is down they are eliminated from further processing. The remaining server or servers are processed further in order to process the user request. Figure 6.4 shows the status of different participating servers.



```
[suresh@starling-01 Test]$ java SystemStatus

****** SYSTEM STATUS TABLE ******

130.179.27.226 : ok
130.179.27.211 : ok
10.0.0.2 : down
130.179.27.212 : ok
130.179.27.213 : ok
192.168.3.167 : down

[suresh@starling-01 Test]$
```

Figure 6.4 *Screenshot-System Status Table*

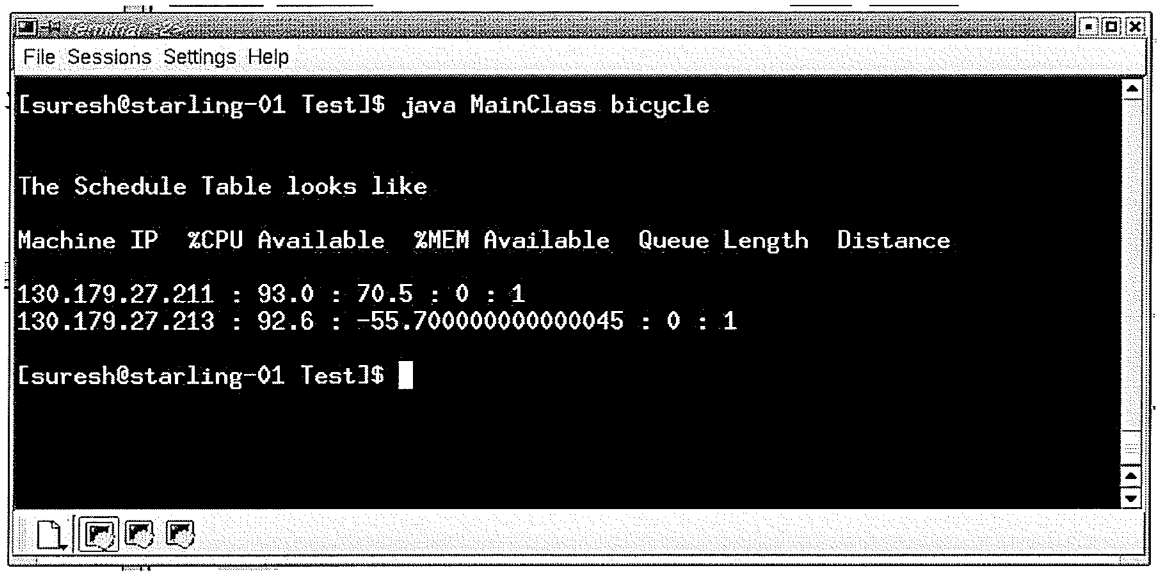
Figure 6.5 gives the proximity table maintained by the cost manager module in the scheduler component.



```
[suresh@starling-01 Test]$ java Proximity_copy

PROXIMITY TABLE

130.179.24.87 : 1
130.179.27.211 : 1
130.179.27.213 : 1
130.179.27.63 : 1
10.0.0.2 : 1
207.161.1.37 : 2

The size of the Proximity Table is: 6

[suresh@starling-01 Test]$
```

Figure 6.5 *Screenshot-Proximity Table*

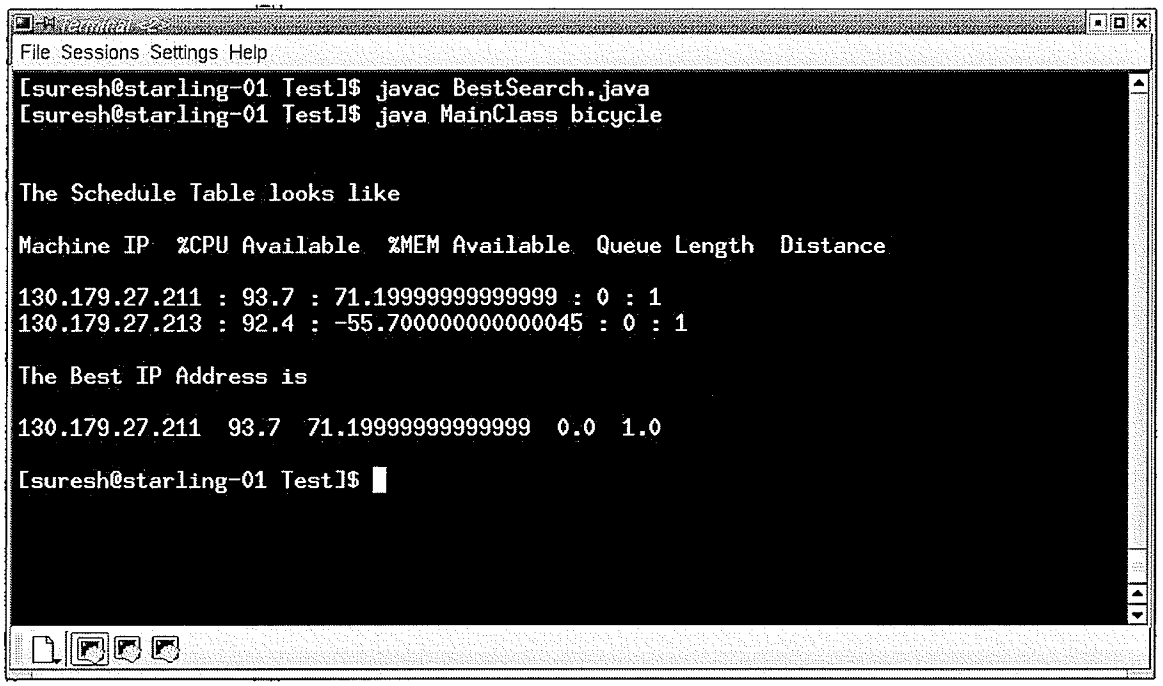
The proximity table maintains the distance information between the client and different participating servers. The distance that is calculated here is the number of nodes or hops in between the client and the participating server. It has two columns: server address and distance. The distance information is one of the load details. Cost manager maintains the proximity table. The cost manager gives the distance information for different servers present in the data location table. The distance information is sent to the scheduler along with other load details like percentage of CPU available, available memory and the length of the processing queue for further processing. The scheduler collects all the information and forms the schedule table shown in Figure 6.6.



```
[suresh@starling-01 Test]$ java MainClass bicycle


The Schedule Table looks like

Machine IP  %CPU Available  %MEM Available  Queue Length  Distance

130.179.27.211 : 93.0 : 70.5 : 0 : 1
130.179.27.213 : 92.6 : -55.700000000000045 : 0 : 1

[suresh@starling-01 Test]$
```

Figure 6.6 *Screenshot-Schedule Table*

Figure 6.6 shows the load details of each server present in the data location table. The different load details shown above are obtained from different components of the scheduler unit. The load inspector gives the percentage of free cpu, memory available, and length of the processor queue. The cost manager gives the distance information. Based on the results present in the schedule table the best server for processing is selected. The best server address is selected based on the set of metrics mentioned in previous Object model and Functionality chapter. Figure 6.7 shows the screenshot of the best-selected server.



```
[suresh@starling-01 Test]$ javac BestSearch.java
[suresh@starling-01 Test]$ java MainClass bicycle


The Schedule Table looks like

Machine IP  %CPU Available  %MEM Available  Queue Length  Distance

130.179.27.211 : 93.7 : 71.19999999999999 : 0 : 1
130.179.27.213 : 92.4 : -55.700000000000045 : 0 : 1

The Best IP Address is

130.179.27.211  93.7  71.19999999999999  0.0  1.0

[suresh@starling-01 Test]$ 
```

Figure 6.7 *Best Selected Server*

The above figure shows the screen shot of how the best efficient server is selected. Algorithm 4.7 in object model and functionality chapter explains how the best and efficient server address is selected from the schedule table.

6.2 User Scenario

This section discusses the user scenario of the designed intelligent system. Figure 6.8 shows an overview of the prototype of the intelligent content-based routing.

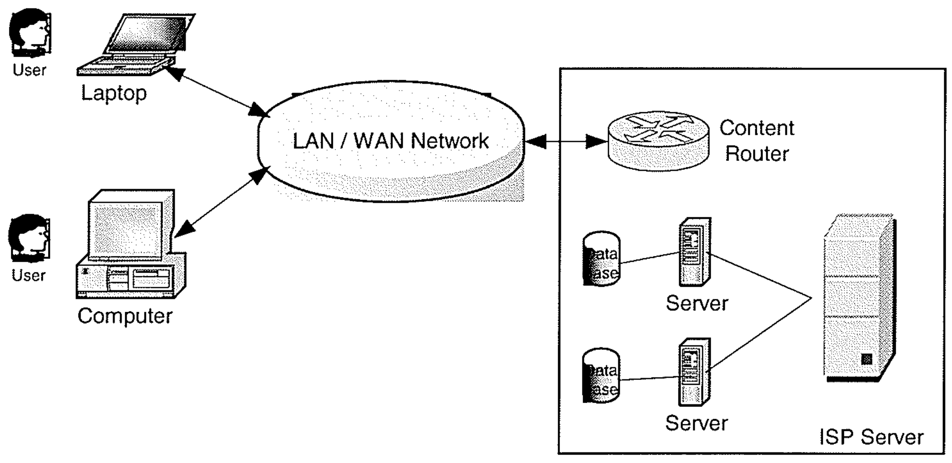


Figure 6.8 *Overview of the Intelligent Content Based Routing Architecture*

To test the prototype the necessary components are an intelligent content router, different participating servers and some clients connected to the network. When the system is started, the clients connected to the network may send their requests. The content router has different components embedded with it. The packet inspector component captures the request in the form of packets. It uses the divert sockets API to grab the packet from the network. The packet inspector extracts the data, tokenize it and forwards it to the resource inspector, which is the next component, embedded in the content router. When the system is switched on the resource inspector looks for the resources by sending queries to the participating servers. The different databases on the servers are accessed by using JDBC driver. Java has lots of JDBC API's for accessing the data present in different databases. Using JDBC we can access any type of data source. The resource locator in the resource inspector component collects all the resources. The user request is searched in the resource list maintained by the resource inspector to locate different servers. The user request is forwarded to the selected server and response is sent back to the client. The load information is obtained by using JAVA RMI. RMI supports different API's to send agents to different machines to collect load information. The above mentioned procedure is used to process the client's request quickly and efficiently in a short span of time. The full implementation of the content router is done in JAVA and C. We choose JAVA for implementation because of platform independence it enjoys. The packet capture is implemented in C.

6.3 Implementation Results

Many factors affect the performance of a router. Examples of such factors are network traffic, load on network nodes, and system configuration. A small change in any of these factors can drastically affect the performance of a router. To measure the performance of the content-based router designed in this thesis, test were performed on various algorithms and time taken to complete their execution is calculated in milliseconds. For each tests we experimented 20 different test cases and we took the average of the 20 cases to find the time taken. The network used for the test has 12 different cluster nodes. We converted the cluster nodes as 12 different servers and we measured the time taken by each algorithm to complete its functionality. The algorithms were implemented in Java. The first test measured the Load Inspector algorithm. We start by finding the time taken to find load information for one node and we increase the number of server nodes incrementally, until all the 12 nodes are included. Finally we tested the algorithm for 12 server nodes. Figure 6.9 shows the graph for time taken to find the load on nodes versus number of different nodes.

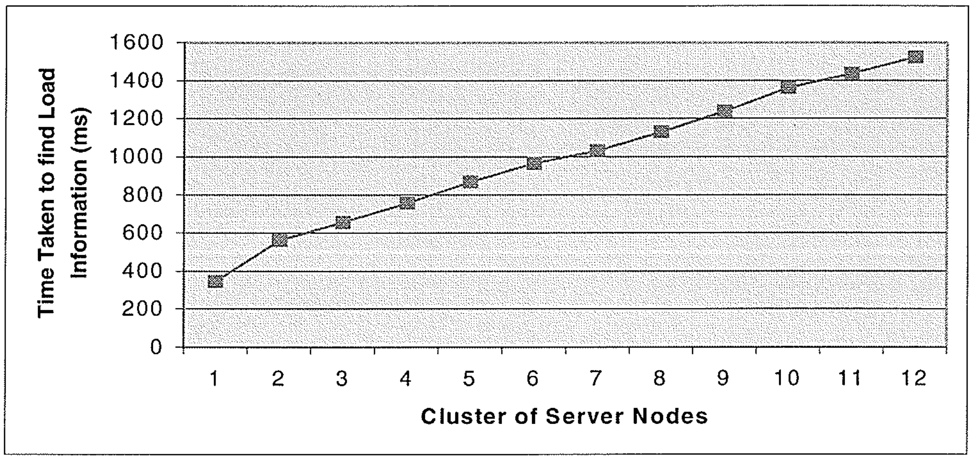


Figure 6.9 *Performance test results for Load Inspector Algorithm*

The load information that the algorithm looks for are free percentage of cpu and memory available and length of processor queue. In the graph, most of the processing time is spent on parsing the result of load details.

Next we measured the performance of the Proximity algorithm. This test was performed for the traceroute implementation used in our algorithm. Similar to the previous test we started the test by finding the distance between the client and server for two nodes and we extend the test to 12 nodes with an interval of two. Figure 6.10 shows the performance for the proximity algorithm.

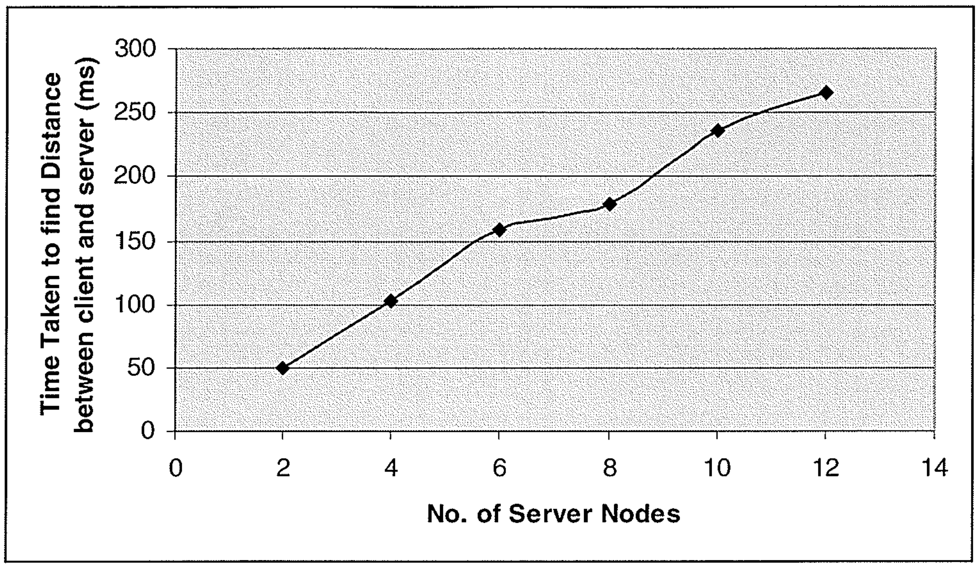


Figure 6.10 *Performance test results for Proximity Algorithm*

The execution of the algorithm takes place in parallel, which reduces the execution time. The elapse time is spent on finding the size of the result vector that stores the distance between a client and server.

The performance of the router can be improved by implementing the different algorithms using C instead of Java. The initial overhead that is seen in the above graphs is might be, due to the time taken by Java to load the Java Virtual Machine, time it takes to load different libraries included in the program as well as the time taken to handshake with other network nodes.

6.4 Summary

Section 6.1 presented some of the screenshots and explained how the actual system will work in real time. Section 6.2 presents a user scenario and explains the interaction between the users and Section 6.3 gives the performance result for the designed content router. Chapter 7 presents conclusion and summary of various contributions presented in this dissertation and gives some future research directions.

Chapter 7

Conclusion

In this thesis, we present the design and implementation of an Intelligent Content-based router that finds a suitable server for processing a client's request quickly and efficiently. The main reason for developing a new intelligent content-based router is that the current routers fail to deliver information to users in right time due to the large increase in Internet users. This led to the increase in network traffic and load on different servers. The newly developed content router could potentially reduce network traffic and optimizes routing cost. By reducing the network traffic and optimizing the routing cost the performance of the router might be increased. The different components present in my content router are Packet Inspector, Resource Inspector and Scheduler. The key features of the newly designed content router are, existing content routers handle the data present in the servers in a monotonous way. The newly developed content router handles different data based on their classification obtained from the Resource Locator. The other key feature of the architecture is that the execution of each of the components is done in parallel. The functionality of these components is explained in detail in Chapter 5. Based on the information obtained from these components a user's requests are forwarded to the appropriate server.

7.1 Comparison between Existing Content Routers and newly Designed Content Router

The main difference between the newly designed content router and the existing routers are:

Existing Content Routers	Newly Designed Content Router
1. Static Routing	1. Dynamic Routing.
2. Consider only one or two components for routing.	2. Consider all the specified components for routing.
3. Examines only the HTTP based request	3. Examines all types of TCP - based requests.
4. Replicates data.	4. No replication of data.
5. Maintains a single large database for storing the resource information, which in turn increases the access time for accessing the resources.	5. Have heterogeneous databases for storing the resources. Uses cache as one of the components for locating server addresses which decreases access time.

This design is also mathematically proved to be robust and fail-safe. The correctness of the design is done using a formal specification language (Z), and the specification is verified using Z-Eves tool.

7.2 Summary of Contributions

This thesis addresses different problems like network traffic, load on different servers, replication of data, status of the servers, and performance of the router. The list of contributions that were made for this dissertation is listed below.

- Provided a new architecture for Intelligent Content-Based Router.

- Provided different network designs to utilize the services of the newly designed Intelligent Content Router efficiently.
- Provided an Object Model for the developed content router.
- Provided a Formal Specification for the newly developed Content-Based Router.
- Provided a prototype implementation of the newly proposed architecture.

7.3 Future Directions

This thesis provides a verified, content-based routing technology that can be used to build application-specific intelligent software routing environments. Such environments can be exploited to create more efficient geographically distributed databases and other similar applications. The areas of applications are vast, ranging from e-commerce to intelligent network switches and call-center processing.

Intelligent content-based routing provides the following key services: (i) content-based routing, (ii) traffic optimization, (iii) economically scalable services that provide appropriate response to varying processing loads, and (iv) the ability to track content requests and respond with appropriate content.

We conclude this thesis by identifying some of the important issues to be addressed as an extension of this research. The future work suggested here is based on this work combined with directions to address the general problem of content-based routing with respect to Internet applications like E-Commerce.

- The different network designs that were proposed in this thesis can be simulated to study the performance and obtain best design for different applications.
- This thesis has provided some of the background research for implementing the different network designs using MPLS. In this thesis we have not implemented the different network designs.

- Using some standard optimization techniques the different algorithms such as load inspector algorithm, proximity algorithm and system status algorithm that were implemented might potentially be optimized to obtain a better performance.
- Another interesting direction for future work would be in the area of wireless access of resource data from remote locations. We need to assess the performance of the content-router in a wireless environment.
- In this thesis, I did not consider security problems. If this factor is considered, the encryption and decryption of data that is passed in the network has to be done. This is potentially one of the future works that is suggested because all the transactions done in an E-Commerce application should be secured.

The content router design proposed in this thesis lays a solid foundation for future work where different E-Commerce applications can be built over it. As a final comment, it is expected that the report presented in this thesis will also lead to the development of a hardware intelligent content - based router.

Bibliography

[KLS98] V. P. Kumar, T. V. Lakshman, and D. Stiliadis, "Beyond Best Effort: Router Architecture for the Differentiated Services of Tomorrow's Internet", *IEEE Communications Magazine*, 36(5):152-164, May 1998.

[GLH95] D. Ghosal, T. V. Lakshman, and Y. Huang, "Parallel Architectures for Processing High Speed Network Signaling Protocols", *IEEE / ACM Transactions on Networking*, pages 716 – 728, December 1995.

[GLM98] Pankaj Gupta, Steven Lin, and Nick McKeown, "Routing Lookups in Hardware at Memory Access Speeds", *IEEE INFOCOM*, April 1998.

[SV97] V. Srinivasan and G. Varghese, "Efficient Best Matching Prefix Using Tries", Pre– Publication Manuscript, January 1997.

[KS98] S. Keshav and R. Sharma, "Issues and trends in Router Design", *IEEE COMMUNICATONS Magazine, 35(6) :* 144-151, May 1998.

[DKS89] A. Demers, S. Keshav, and S. Shenker, "Design and Analysis of a Fair Queuing Algorithm", Proceedings of ACM SIGCOMM '89, Austin, September 1989.

[P+98] Craig Partridge et al, "A 50-Gb/s IP Router", *IEEE / ACM Transactions on Networking*, Vol. 6 No. 3, June 1998.

[ADJ+92] A. Asthana, C. Delph, H. V. Jagadish, and P. Krzyzanowski, "Toward a Gigabit IP Router", Journal of High Speed Networks, Vol. 1, No. 4, pp. 281 – 288, 1992.

[K94] S. Konstantindou, "Segment Router – A Novel Router Design for Parallel Computers", IBM T. J. Watson Research Center, Yorktown Heights, NY 10598. (Also published in the Proceedings of ACM SPAA-94, Cape May, N.J., USA, 1994).

[WVT+97] Marcel Waldvogel, George Varghese, Jon Turner, Bernhard Plattner, "Scalable High Speed IP Routing Lookups", *In Proceedings of SIGCOMM' 97,* September 1997.

[APP+99] G. Apostolopoulos, V. Peris, P. Pradhan, and D. Saha, "L5: A Self-Learning Layer-5 Switch", IBM Research Report RC21461, T.J. Watson Research Center, 1999.

[S88] J. M. Spivey, *Introducing Z: A Specification Language and its Semantics.* Cambridge University Press, 1988.

[SM99] *Z/EVES Version 2.0*, ORA Canada, Ottawa, Ontario, K1Z 6X3, CANADA (available at http://www.ora.on.ca/z-eves/welcome.html). (Also associated with this is *The Z/EVES Reference Manual* by Mark Saaltink and Irwin Meisels, ORA Canada, December 1995; revised September 1997 and October 1999).

[UML99] *Unified Modeling Language Specification* (draft), Version 1.3 alpha R5, Object Management Group, Inc., March 1999.

[E00] S. A. Ehikioya, "Formal Specification of Intelligent Routing Infrastructure for Electronic Commerce Systems", Technical Report # TR-CS-22-2000, Dept of Computer Science, University of Manitoba, Winnipeg, Canada, June 2000.

[HGK+98] G. Hunt, G. Goldszmidt, R. King, and R. Mukherjee, "Network Dispatcher: A Connection Router for Scalable Internet Services", Proceedings of the 7th International World Wide Web Conference, Brisbane, Australia, April 1998.

[AM98] D. Andresen and T. McCune, "Towards a Hierarchical System for Distributed WWW Server Clusters", Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC7), Chicago, IL, July 1998, pp. 301-309.

[PAB+98] V. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum, "Locality-Aware Request Distribution in Cluster-based Network Servers", Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII), San Jose, California, October 1998.

[SLI+00] J. Song, E. Levy-Abegnoli, A. Iyengar, and D. Dias, "Design Alternatives for Scalable Web Server Accelerators", Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software, Austin, TX, April 2000.

[SLI+99] J. Song, E. Levy-Abegnoli, A. Iyengar, and D. Dias, "A Scalable and Highly Available Web Server Accelerator", IBM Research Report RC 21377, Shorter version appeared in Poster Proceedings of the 8th International World Wide Web Conference (WWW8), Toronto, Canada, May 1999.

[GC00] Z. Genova and K. Christensen, "Challenges in URL Switching for Implementing Globally Distributed Web Sites". Proceedings of the Workshop on Scalable Web Services, August 2000, pp. 89 - 94.

[CFH99] M. Crovella, R. Frangioso, and M. Harchol-Balte. "Connection Scheduling in Web Servers". In Proceedings of the 1999 USENIX Symposium on Internet Technologies and Systems (USITS '99), October 1999.

[C00] Cisco Systems Inc,. "Content Routing Protocols", White Paper, Cisco Systems Inc, October 31, 2000.

http://www.cisco.com/warp/public/cc/pd/cxsr/cxrt/tech/ccrp_wp.htm.

[CCY00] V. Cardellini, M. Colajanni, and P. S. Yu. "Geographic Load Balancing for Scalable Distributed Web Systems". Proc. IEEE Mascots 2000, San Francisco, CA, Aug./Sept. 2000.

[CID+00] J. Challenger, A. Iyengar, P. Dantzig, D. Dias, and N. Mills. "Engineering Highly Accessed Web Sites for Performance". Web Engineering, Y. Deshpande and S. Murugesan editors, Springer-Verlag, 2000.

[B95] T. Brisco. "DNS Support for Load Balancing". *Technical Report RFC 1974*, Rutgers University, April 1995.

[M87] P. Mockapetris. "Domain Names - Implementation and Specification".*Technical Report RFC 1035*, USC Information Sciences Institute, November 1987.

[DS94] Andrzej Duda and Mark A. Sheldon, "Content Routing in a Network of WAIS Servers", *14th International Conference on Distributed Systems*, Poznan, Poland, June 1994.

[SDW+94] Mark. A. Sheldon, Andrzej Duda, Ron Weiss, James W. O'Toole, Jr., and David K. Gifford, "A Content Routing System for Distributed Information Servers", *Proceedings Fourth International Conference on Extending Database Technology*, March 1994.

[ID] http://www.unitechnetworks.com/IntelliDNS/Understanding/

[AP] http://www.knowware.co.uk/ArrowPoint/solutions/whitepapers/WebNS.html