DYNAMIC NETWORK TRAFFIC MANAGEMENT

- Approaching with Intelligent Switching -

A Thesis

Presented to the Faculty of Graduate Studies

of the University of Manitoba

in partial fulfilment of the requirements

for the degree of Master of Science

in the Department of Electrical and Computer Engineering

University of Manitoba

Winnipeg, Manitoba, CANADA

By

Wei Gu

Thesis advisor: Ken Ferens, Ph.D., P.Eng.

0-612-62744-6

DYNAMIC NETWORK TRAFFIC MANAGEMENT

- Approaching with Intelligent Switching -

BY

WEI GU

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

ABSTRACT

This thesis presents the research and development of an intelligent gateway (switching) capable of dynamically distributing Internet-related tasks among a number of servers based on applications, traffic load and system logs, oriented to improving network quality of services (QoS).

To design an intelligent switching, advanced methodologies are required beyond conventional layer 2/3 switches. Tag switching based on IEFT multiprotocol label switching (MPLS) mainly targets QoS implementation over IP networks. Layer 4/7 (session/content) switching has emerged to manage, route and load balance traffic in order to increase performance, security, availability and scalability at the transport and application levels using server clustering (e.g., Web server farms), server load balancing algorithms, network address translation, URL-aware switching, and transparent proxy cache switching of HTTP traffic. This thesis introduces and discusses the related background, technologies and methodologies in detail.

This thesis provides a system architecture of intelligent switching design. It consists of five main modules: Network Management based on SNMP model and sysLogs monitoring, Data Analysis focusing on network performance (mainly availability and response time mainly), Switching Decision using server load balancing algorithms (SLB), Feedback Information Control approaching with connection quality (QoS) and Cisco Dynamic Feedback Protocol (DFP), and Server Farms that provide available network resources. As an application example of system design, the prototype of SmartSwitch Router, a kind of TCP traffic intelligent redirector, is provided regarding the issue of implementation. The software design of TCP Redirector is implemented in the Java programming language based on objected-oriented design (OOD) and objected-oriented programming (OOP). In addition, a three-tier architecture of Linux virtual load server clusters (VLSC) is presented in this thesis, with three kinds of Linux VLSC strategies over IP Networks, i.e., VLSC/NAT (via NAT), VLSC/TUN (via IP tunnel), and VLSC/DR (via direct routing).

This thesis provides the system experiments and analysis results. The TCP traffic redirector is tested on Unix Workstations (SunOS 5.7), an assortment of distributed environments, including

client machines, switching machines, and real server (RServer) machines. The comparison results of the SLB algorithms' performance (Fig. 6.2.) is given based on the test results using different SLB approaches. The experimental results analysis and theory prediction show some conclusions of SLB algorithms: Weighted Round-Robin approaches (WRR-3/4) improve the SLB performance as compared to random Round-Robin approaches (RR), improving around 40% in the case of the extreme server load balance (SLB points: eb*). Least-Connection (LC) scheduling needs to count the number of active connections, and Weighted Least-Connection (WLC) scheduling requires both the count of the number of active connections and the assign of performance weight to each real server. Both LC and WLC are dynamic load balancing algorithms, but they require more resources in practice. In the future, nonlinear and/or distributed SLB algorithms are expected to emerge in the next generation of intelligent switching products.

ACKNOWLEDGEMENTS

This thesis was financially supported through a research scholarship from the Telecommunication Research Laboratory (TR*Labs*), Winnipeg, Canada.

I would like to acknowledge my graduate study advisor Dr. Ken Ferens for introducing me to the subject of computer engineering, and particularly to microprocessing systems, Internet Protocols, and Internetworking. Also, I would like to acknowledge his understanding, encouragement, and enormous help to me during my course work, thesis work, and TA positions. Furthermore, I would like to acknowledge him for spending much time in discussing and reviewing my thesis versions.

I would like to acknowledge my thesis project supervisor Dr. R.D. McLeod for offering me an opportunity to involve myself in the subject of telecommunication networks, and particularly network management, TCP/IP networks, and wireless networks. Also, I would like to acknowledge his financial support of my first research project, and his valuable recommendation of me to join the TR*Labs*. In addition, I would like to acknowledge his encouragement, suggestion, and help to me during my course of study, research project proposals, and thesis work.

I would like to acknowledge TR*Labs* giving me this research opportunity. I would like to acknowledge Dr. Jose A. Rueda for offering me an interesting and challenging research topic on data networks. Also, I would like to acknowledge his suggestions, support, and valuable recommendations to me during my research. Furthermore, I would like to acknowledge the help and friendship I received from TR*Labs*, Paul Card, Xiaoming Bao, Jeff Diamond, and other nice people.

I would like to acknowledge Dr. David Blight for his discussion, valuable suggestion and encouragement of my research progress in the areas of telecommunication networks.

I would like to acknowledge the other help and friendship I received during my attendance at the University of Manitoba over the past three years, from Liting Han, Wan Lin, Jizong Li, Guy Jonatschick, Mount-first Ng, Steve Kretschmann, Hart Poskar, Richard Danserea, Fred Corbett, Blair Yoshida, and other engineering and science majors.

This thesis is dedicated to my dearest Mom and Dad for their enormous encouragement, support and love.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

ABBREVIATION	PAGE
Abstract Data Types **(ADTs)**	84
Address Resolution Protocol **(ARP)**	80
Application Layer **(Layer 7)**	5
Application Programming Interfaces **(API)**	51
Abstract Syntax Notation 1 **(ASN.1)**	61
Application-Specific Integrated Circuits **(ASICs)**	120
Asynchronous Transmission Model **(ATM)**	16
Border Gateway Protocol **(BGP)**	21
Broadband ISDN **(B-ISDN)**	30
Committed Access Rate **(CAR)**	18
Cisco Dynamic Feedback Protocol **(CDFP)**	59
Cisco Express Forwarding **(CEF)**	37
Common Object Request Broker Architecture **(CORBA)**	3
Custom Queuing **(CQ)**	19
Differentiated Service Model **(DiffServ)**	18
Domain Name Service **(DNS)**	45
Data Link Layer **(Layer 2)**	7
Institution of Electrical and Electronic Engineers **(IEEE)**	16
External Gateway protocol **(EGP)**	62
File Transmission Protocol **(FTP)**	4
First In First Out **(FIFO)**	19
Forwarding Information Base **(FIB)**	38
Frequency Division Multiplexing **(FDM)**	30
Hyper Text Transfer Protocol **(HTTP)**	5
Integrated Services Model **(IntServ)**	18
Integrated Services Digital Networks **(ISDN)**	29

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

1.1. Introduction

The development of telecommunication and computer networks today makes the Internet an integrate part of our lives; however, there are still many performance complaints such as "Why does this Web site respond so slowly?".

In response to these complaints, network engineers and researchers are faced with solving highly-technical problems to provide better solutions. For instance, for problems of insufficient bandwidth, network bottlenecks and traffic congestion, we need appropriate approaches such as intelligent bandwidth management.

With the rapid growth of the Internet and Intranet deployment and usage, the higher performance and availability of networks and increased quality of services are required. The advanced services with quality of service (QoS) extensions will also enable new Internet business models.

1.2. Today's Challenge of Critical Networks

In order for Internet Service Providers (ISPs) to offer widespread value-added services (or QoS services) which meet customers' demanding application needs, the network infrastructure must handle a key set of technological and business requirements.

- Service Scalability: An increasing number of network capabilities and services will be activated to meet customer needs and to implement service provider network resource allocation policies. The network must maintain high packet throughput in this services-rich environment.
- Intelligent Congestion Control: The network must actively avoid congestion conditions that can lead to throughput degradation, recover gracefully from congestion situations and distinguish

- Intelligent Congestion Control: The network must actively avoid congestion conditions that can lead to throughput degradation, recover gracefully from congestion situations and distinguish between temporary traffic bursts and long term traffic overload conditions, and also be able to maintain preferential treatment of higher priority traffic without getting worsening congestion under overload conditions.
- Traffic Classification and Prioritization: The network must be able to efficiently sort and classify packets into traffic classes or service levels for appropriate network handling and meet customer application requirements and willingness to pay for services. Both service providers as well as customer applications must have the capability to classify traffic, but the service provider must be able to override customer classifications under appropriate conditions.
- Bandwidth Allocation: Service providers must be able to provide and enforce bandwidth commitments to traffic sources and to flexibly determine packet handling policy when bandwidth allocation is exceeded.
- Policy and Service Flexibility: Service providers require the flexibility to define and offer highly differentiated services to target customers with minimal vendor interactions, and must also have the flexibility to specify resource allocation policies at different levels including by physical port, by address and by application.
- Investment Protection: Service providers must be able to gain the benefits of new services and capabilities via meeting the installed base of network elements and software without requiring widespread, fundamental changes to the underlying network hardware and protocols.
- Adaptive Application Support: Customers must be able to request network resources (e.g., bandwidth allocation) which meet application requirements without modifying the installed applications, and applications must also be able to choose to actively leverage new network capabilities to reach the maximum base of users under a variety of network conditions.
- Network Element Cooperation: Network elements at the edge and backbone of the network must actively cooperate to maximize throughput and performance, minimize the impact of congestion and provide end-to-end service levels and QoS policies to meet customer requirements. Service providers also require the means to provide end-to-end service quality from networks built with multiple technologies (e.g. router, Frame Relay, tag switching) to internal works with external networks while meeting QoS goals.

There are many researchers and engineers in the area of telecommunications and computer networks who are working hard on these challenges. My research project focuses on dynamic network traffic management using intelligent switching techniques to improve the network quality of services.

1.3. Motivation of Project

1.3.1. Project Origin

The project entitled "Design of a Dynamic Network Traffic Allocation Gateway" is supported by TR*Labs* (Telecommunication Research Labs, Canada). This project is proposed based on an extension of our previous work on the Web-based network management [GuMF20] [LiJi99], at the Department of Electrical and Computer Engineering, University of Manitoba, Canada.

The project Web-based Network Management System [LiJi 99] uses a three-tier client/server architecture, approaching with SNMP/CORBA/Database. Based on SNMP (Simple Network Management Protocol) management model, CORBA (Common Object Request Broker Architecture) [ViBr99] is used as a middleware. This distributed system includes several machines, at least one SNMP server manager, several SNMP agents, one mSQL (Mini Structured Query Language) Database server [HuTc99], one CORBA gatekeeper on Web server machine, and Web client machines. The whole system with multiple servers is implemented on Unix Workstations (SunOS). The advantage of using CORBA as a transparent gateway is that a Web client, who has authentication to access the gateway of the network management system, might be able to get the SNMP information of devices on LANs (Local Area Networks) through this client/server management system, thus Web clients (e.g., a network administrator) are not restricted to work only in local offices, that is, they can access a local network and manage the information on this LAN at home and/or remote locations.

Based on these research results, TR*Labs* proposed to telecommunications industry-leading sponsors a new project originated as the design of a gateway to manage web-based applications. This sort of gateway is supposed to be able to distribute requests dynamically based on measurements. The basic idea is to automate the process of reconfiguring which tasks are handled by which servers, for example, the system will be able to reallocate a specific machine as a web server on demand. Thus, the purpose of this project is to develop an intelligent gateway capable of dynamically distributing Internet-related tasks among a number of servers based on application, traffic load and system logs [GuRM20].

1.3.2. Technology and Methodology Required

There are several techniques that are being developed for application-layer switching, such as techniques that perform the tasks based on port number or URLs (Uniform Resource Locators). In this project, we consider utilizing as much information as possible about the application and the status of the system. The designed system will not only operate based on redundancy, but it will be capable of dynamically distributing load and tasks among several servers. The priority will be given to certain servers that will be configured to handle specific tasks (such as Web server or FTP server), and they will also be able to assist with other tasks during heavy load conditions.

Initially this project focused on understanding the available techniques for layer-4/7 switching. Methodologies and techniques were developed, including load measurements and utilization of system logs in the decision making process. A prototype was built and its performance evaluated.

1.3.3. Benefits for Sponsor and Academic Contribution

One project intent was provision of a technical report to TR*Labs* sponsors who are interested in application-layer switching. The research report includes the different techniques available and a prototype of a system that utilizes several sources of information about the system and application. We expect that this research result will be helpful as there is a general trend to build more intelligence in routers and switches so that they are more aware of the applications. This in turn will help in reaching network QoS goals.

As a Master degree thesis in Computer Engineering, this thesis also provides certain academic research results in the area of telecommunication networks, such as network measurement based on applications, traffic load and system logs; network traffic management by dynamically distributing system resources; and the design, implementation and analysis of TCP (Transmission Control Protocol) traffic allocation using intelligent switching algorithms.

1.4. Contents of the Thesis

This thesis primarily consists of seven chapters.

Chapter 1 (*Introduction*) briefly introduces the background and motivation on this research project.

Chapter 2 (*Background & Technology*) presents some basic background and advanced technologies, including the concepts of intelligent gateway, network management based on SNMP mode, network Quality of Service (QoS), and network switching.

Chapter 3 (*Methodology of Intelligent Switching*) systematically discusses the different methodologies of intelligent switching design based on TCP/IP (Transmission Control Protocol / Internet Protocol) reference model, i.e., the technical methods of switching at Layer-3 (Network Layer), Layer-4 (Transportation Layer), and Layer-7 (Application Layer). It primarily includes QoS smart router (Netflow, switching, Cisco Express Forwarding, and tag switching), server farms (clusters), server load balancing algorithms, network address translation, URL-aware switching, transparent proxy cache switching of HTTP (Hyper Text Transfer Protocol) traffic.

Chapter 4 (*System Design & Architecture*) presents the approaches of system design, and provides the system architecture consisting of five major modules: Network monitoring, Data analysis, Switching decision, Feedback information control, and Server farms. The system functionality is discussed with some practical examples, including SNMP monitoring, system log monitoring, network performance analysis, and feedback information control approaches.

Chapter 5 (*System Implementation*) firstly presents some implementation issues of Linux Virtual Server Clusters (LVSC) architecture and three strategies over IP networks. Next, an application example of system design, called SmartSwitch Router, is provided, which includes a prototype of TCP intelligent traffic redirector and its programming implementation using server load balancing algorithms.

Chapter 6 (*System Experiments and Analysis*) provides the system test environments, various experiments and results, and results analysis of server load balancing (SLB) algorithm performance.

Chapter 7 (*Conclusions & Recommendations*) gives the summary of this thesis and some recommendations for future works.

Finally, this thesis includes **Appendix A.-E.**, providing some relevant background, technologies and further information.

Chapter 2

Background & Technology

This chapter discusses some background and technologies related to network traffic management and system design. They are intelligent gateway technology, network management technology, network QoS, and network core switching.

2.1. Concepts of Intelligent Gateway

Generally speaking, a *gateway* is a device that can handle and control incoming and outgoing traffic. In the context of telecommunications, a network gateway is a device that controls inbound and outbound network traffic. GATEWAYS, defined by Andrew S. Tanenbaum [Tane96], are machines used to make the connection among different networks which are frequently incompatible, and to provide the necessary translation, both in terms of hardware and software. Here a collection of interconnected networks is called an internetwork or just internet.

Many different functionalities can be included in gateway products, such as security gateways (or called firewalls), traffic management gateways, and connection link gateways (such as X.25 gateway link, ATM switch), and so on. For example, the *Lucent VPN Firewall* family [Luce20] layer-2 (Data Link layer) devices are completely invisible in the network and can address the needs of customers with growing networks or significant remote access VPN (Virtual Private Network) requirements. Another good example of a traffic management gateway is the *Cisco DPA 7630 Voice Mail Gateway* [Ciso20], which is a Voice over IP (VoIP) gateway that enables legacy voice mail equipment to connect to a Cisco IP telephony solution network.

TRAFFIC, according to the Merriam Webster Dictionary, is "the movement (as of vehicles) along a route: also: the vehicles, people, ships, or planes moving along a route" or "the passengers and cargo carried by a transportation system." Similarly, **NETWORK TRAFFIC** means various data moving across networks, such as common e-mail traffic, WWW traffic and multimedia traffic including pictures and voice data, and so on.

Naturally, to manage traffic, there are some public regulations and policies to be followed, such as traffic lights, stop signs, and yield rules, and sometimes police cars with radar are needed to monitor the speedy cars. **TRAFFIC MANAGEMENT** is a way to monitor and control traffic so that network quality can be improved. In the past, network management usually focused on LAN management, mostly using SNMP protocol, a Simple Network Management Protocol. The SNMP is a good way to monitor LAN devices [Stal93]. With the rapid growth of enterprise networks and the Internet, WAN (Wide Area Network) management is becoming more necessary and important. Consequently, advanced network management is targeting network quality of services and network design optimization.

The **INTELLIGENT GATEWAY**, as proposed in this thesis, provides a new architecture of network traffic management with some intelligent functional components. It can handle a variety of inbound and outbound traffic between WAN and LAN [GuMR20].

Figure 2.1 shows the components of an intelligent gateway. Packet Coloring, Security Service, QoS Service Agent, Monitoring Agent, Policy Manager, Feedback Agent and Decision & Switch are responsible for dealing with inbound traffic. The Policy Agent and Distributor are used for handling outbound traffic. The basic functionalities of these components are discussed next.

The **PACKET COLORING** primarily deals with intelligent packet classification, which is designed for precise control of traffic (by identifying specific traffic types) and bandwidth management (by guaranteeing bandwidth to applications). Discovering and classifying traffic is based on information layer-2 through layer-7, by ports, by protocol or service, and by specific applications and URLs.

The **SECURITY SERVICE** acts as an access admission agent with firewall, policy and database. The technology behind this includes authentication, public-private keys, RSA algorithm, digital signature, IPSec protocol, and so on. More information can be found in [Tene96].

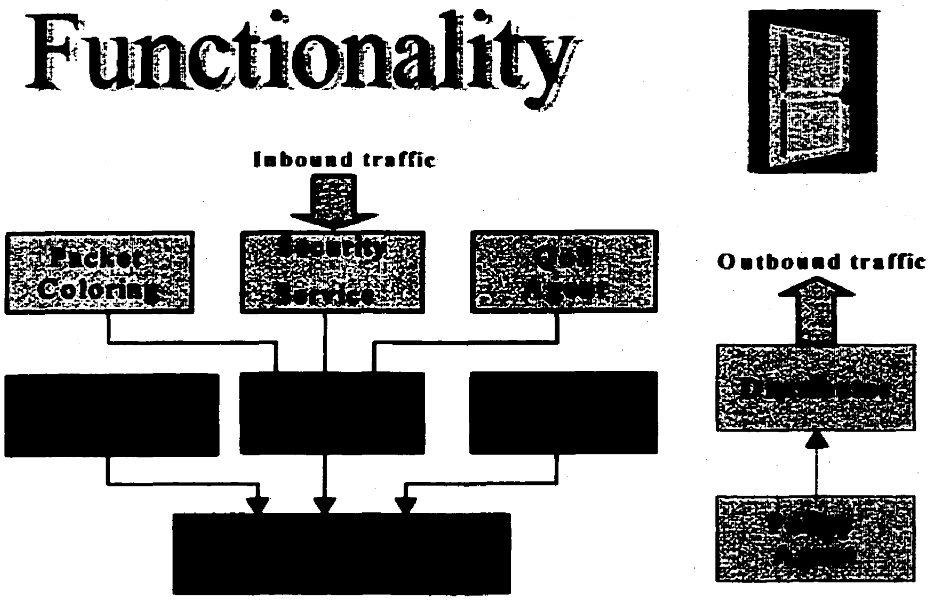

Fig. 2.1. Functionality of intelligent gateway.

The **QoS Agent** handles network quality of services functions, such as traffic policing, shaping and queuing, bandwidth allocation using WFQ (Weighted Fair Queuing), congestion avoidance using WRED (Weighted Random Early Detection), and so on.

The **Monitoring Agent** mainly copes with LAN network traffic management by basically using SNMP protocol and system logs to monitor LAN devices and incoming/outgoing traffic.

The **Feedback Agent** monitors and records traffic load and connection quality, and collects the feedback information from Internet clients.

The **Policy Manager** creates and regulates QoS policies and implements these rules.

The **Policy Agent** provides the QoS policy to the Distributor.

The **Distributor** implements the policies and distributes priority traffic to the outbound routers (or channels).

The **DECISION & SWITCH** acts as a decision making process, which is based on the information from the monitoring agent, policy manager and feedback agent, and uses some intelligent switching algorithms to select the best servers from server clusters and reallocate the traffic.

2.2. Network Management

2.2.1. Network Management System

The Network Management System (NMS) [Stal93] is a collection of tools for network monitoring and control that is integrated in the following two senses: (a) a single operation interface with a powerful but user-friendly set of commands for performing most or all network-management tasks; (b) a minimal amount of separate equipment, that is, most of the hardware and software required for network management is incorporated into the existing user equipment.

A network-management system consists of incremental hardware and software additions implemented among existing network components. The software used in accomplishing the network-management tasks resides in the host computers and communications processors (e.g., front-end processors, terminal cluster controllers, bridges, and routers). A network-management system is designed to view the entire network as a unified architecture, with addresses and tables assigned to each point and the specific attributes of each element and link known to the system. The active elements of the network provide regular feedback of status information to the network-control center.

In OSI (Open Systems Interconnection) system management, the basic network management functional features include: fault management, accounting management, configuration and name management, performance management and security management. a) *Fault management:* the facilities that enable the detection, isolation and correction of abnormal operation of the OSI environment. b) *Accounting management:* the facilities that enable charges to be established for the use of managed objects and costs to be identified for the use of those managed objects. c) *Configuration and name management:* the facilities that exercise control over, identify, collect data from, and provide data to managed objects for the purpose of assisting in providing for continuous operation of interconnection services. d) *Performance management:* the facilities needed to evaluate the behavior of managed objects and the effectiveness of communication activities. e) *Security management:* the facilities that address those aspects of OSI security essential to operate OSI network management correctly and to protect managed objects.

The key functional areas of network management listed above are defined by the International Organization for Standardization (ISO). Although this functional classification was developed the OSI environment, it has gained broad acceptance by vendors of both standardized and proprietary network-management systems.

2.2.2. Network Monitoring and Control

The fundamental capability of network management is to gather information about the status and behavior of the networking configuration, which is the function of network monitoring. Indeed, many network-management systems only provide a network-monitoring capability. A complete network-management system should include the capability of controlling a configuration as well as monitoring it.

The network-monitoring portion of network management is concerned with observing and analyzing the status and behavior of the end systems, intermediate systems, and subnetworks that make up the configuration to be managed. It consists of three major design areas: a) Access to monitored information which is concerned with how to define monitoring information and how to get that information from a resource to a manager. b) Design of monitored mechanisms that are concerned with how best to obtain information from resources. c) Application of monitored information that is concerned with how the monitored information is used by various management functional areas. Within the network-monitoring architecture, the information that should be available for network monitoring can be classified as follows: a) Static: information that characterizes the current configuration and elements in the current configuration, such as the number and identification of ports on a router. This information will change only infrequently. It is stored into a statistical database. b) Dynamic: information related to events in the network, such as a change of state of a protocol machine or the transmission of a packet on a network. It is stored into a dynamic database. c) Statistical: information that may be derived from dynamic information, such as the average number of packets transmitted per unit of time by an end system. It is stored into a statistical database.

There are three functional areas that are generally most important for network monitoring: performance monitoring, fault monitoring, and accounting monitoring. In the area of performance management, the most important categories of management information are: availability, response time, accuracy, throughput, and utilization. In the area of fault monitoring, the objective is to identify faults as quickly as possible after they happen and to determine their cause so that remedial action may be taken. The fault monitoring function is complicated because it is needed at a time when some portion of the network is not functioning properly, and it may be difficult to learn of and identify a certain fault. In the area of accounting management, network monitoring is concerned with gathering usage information to a level of detail required for proper accounting.

Our research work and implementation primarily focuses on performance management, that is, network monitoring is used to observe and track the events and activities on the network. Performance measurement is used to gather statistics about network traffic and timing, while performance analysis is used to analysis and present data for making the synthetic traffic generation and network control is used to make adjustments to improve network performance.

2.2.3. SNMP Network Management Model

This section includes some issues pertaining to the SNMP model concept, MIB (Management Information Base), and SNMP protocol.

The network-management framework for TCP/IP-based Internet has been developed and standardized for use in conjunction with the TCP/IP protocol suite. This framework includes: a simple network-management protocol (SNMP), a structure of management information (SMI), and a management information base (MIB).

The foundation of a TCP/IP-based network-management system is a database containing information about the elements to be managed. In both the TCP/IP and OSI environments, the database is referred to as a *management information base (MIB)*. Each resource to be managed is represented by an object. The MIB is a structured collection of such objects. Each node in the system will maintain a MIB that reflects the status of the managed resources at that node. A network-management entity can monitor the sources at that node by reading the values of objects in the MIB and may control the resources at that node by modifying those values.

The second point addresses a ***structure of management information (SMI)***, which is specified in RFC 1155 [RFCs] (Request For Comments), defining the general framework within which a MIB can be defined and constructed. The SMI identifies the data types that can be used in the MIB, and how resources within the MIB are represented and named. The philosophy behind SMI is the encouragement of simplicity and extensibility within the MIB. Thus, the MIB can store only simple data types: scalars and two-dimensional arrays of scalars. SMI does not support the creation or retrieval of complex data structures. SMI avoids complex data types to simplify the task of implementation and to enhance interoperability.

Simple Network-Management Protocol (SNMP), as defined in RFC 1157 [RFCs], is the heart of the SNMP management approach. This protocol is used to manage the objects contained in the MIB. There are three kinds of operations supported by SNMP: *GET* enables the management station to retrieve the value of objects at the agent, *SET* enables the management station to set the value of objects at the agent, and *TRAP* enables an agent to notify the management station of exceptional events.

The second characteristic of SNMP network management is its COMMUNITY, which is a relationship between an SNMP agent and a set of SNMP managers that define authentication, access-control, and proxy characteristics. The community concept is a local one, defined at the agent. The agent establishes one community for each desired combination of authentication, access-control, and proxy characteristics: a) Authentication service: the agent may wish to limit access to the MIB to authorized management stations. b) Access policy: the agent may wish to give different access privileges to different management stations. c) Proxy service: the agent may act as a proxy to other managed stations, and this may involve implementing the authentication service and/or access policy for other managed systems on the proxy system.

Thirdly, SNMP FORMATS [Stal93] of protocol specification are described as the following five types, shown in Figure 2.2.

version	community	SNMP PDU

(a) SNMP message

PDU type	request-id	0	0	variable-bindings

(b) GetRequest PDU, GetNextRequest PDU, and SetRequest PDU

PDU type	request-id	error-status	error-index	variable-bindings

(c) GetResponse PDU

PDU type	enterprise	agent-addr	generic-trip	specific-trap	time-stamp	variable-binding

(d) Trap PDU

name1	value1	name2	value2	• • •	name *n*	value *n*

(e) variable-bindings

Fig. 2.2. SNMP formats.

SNMP-BASED MANAGEMENT MODEL includes the following key elements: management station, management agent, management information base (MIB), and network management protocol. The architecture is shown in Fig. 2.3. In this kind of SNMP model, there are two parts: MANAGEMENT STATION and MANAGED AGENTS, communicating with each other by SNMP protocol. The management station can be any machine in a LAN, having a manager with its own MIB, and the management applications which can be developed by the software. The managed agent can be any device needed to monitor and control, such as a host, router, or bridge. A MIB is installed on any agent being managed.

An application layer protocol SNMP is used to communicate between the management station and agents. Actually, a manager sends requests to an agent on port 161 for getting device information; the agent receives messages and accesses the data from its own MIB, then sends datagrams back to the manager, which listens on port 162. This sort of communication and message exchange is based on User Datagram Protocol (UDP). Here, only one manager is needed, but there can be any number of agents on this LAN.

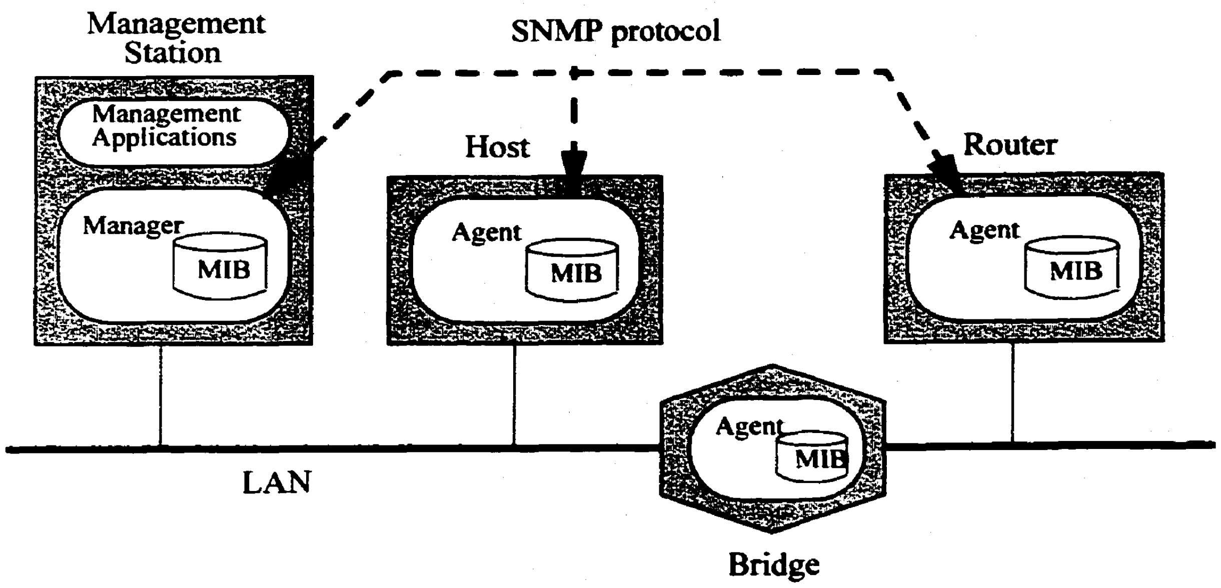

Fig. 2.3. Components of the SNMP management model [Stal93].

The management applications can be developed in different ways, such as a software only running only on a local LAN machine, or a Web-based monitoring server system by which a system administrator (or an authentication user) can access a SNMP manager only from an appropriate Web site. SNMP is designed to operate over the connectionless user datagram protocol; however, SNMP can be implemented to operate over a variety of transport-level protocols.

2.2.4. SNMP Family

As the name suggests, Simple Network Management Protocol (SNMP) provides a simple facility for network management. It is easy to implement and should consume minimal processing resources. Many users of SNMP have felt the need for a more powerful and comprehensive facility. In response to this need, fortunately, there are various versions of SNMP protocol in the SNMP family, currently including SNMP, RMON, Secure SNMP, SNMPv2, Secure SNMPv2, SNMPv3 for different purposes and requirements of network-management systems.

Remote-monitoring (RMON) refers to the monitoring of a network by one of the nodes on the networks for the purposes of network management. A network-management system is concerned not only with the status and behavior of individual nodes on a network but with the traffic on the network itself. A RMON MIB has been defined in RFC 1271 [RFCs], specifying the information that is to be collected and stored by a remote monitor. The specification also defines the functionality of both the monitor and the manager-monitor interaction.

A deficiency of the original SNMP set of specifications is that it contains only a very weak mechanism for enforcing security. To remedy this deficiency, a security enhancement to SNMP has been specified to the security SNMP facility, defined in RFC 1351, RFC 1352, RFC 1353, and RFC 1321 [RFCs].

Version 2 of SNMP (SNMPv2) was developed for providing the security features of Security SNMP plus functional enhancements to SNMP. SNMPv2 was actually developed by two separate working groups, one concentrating on management information and protocol issues, and the other on security. SNMPv2 is specified in RFC 1825-1829 [RFCs].

In the context of SNMP, the Management Information Base (MIB) needs to be adapted to the diversity of different versions of SNMP protocols. MIB-II, defined by RFC1213 [RFCs], is usually installed on most Sun Microsystems Unix Workstations.

2.3. Network Quality of Services

2.3.1. What is Quality of Service?

QUALITY OF SERVICE, or QoS, refers to the ability of a network to provide better service through the selected network traffic over various underlying technologies including: Frame Relay, Asynchronous Transfer Model (ATM), Ethernet and IEEE 802.1 networks, Synchronous Optical Network (SONET), and IP-routed networks. Specifically, QoS features provide better and more predictable network service by: 1) supporting dedicated bandwidth; 2) improving loss characteristics; 3) avoiding and managing network congestion; 4) shaping network traffic; 5) setting traffic priorities across the network [Tane96].

QOS ARCHITECTURE: QoS features can be configured throughout a network to provide for end-to-end QoS delivery. The following three components are necessary to deliver QoS across a heterogeneous network: 1) QoS within a single network element, which includes queuing, scheduling, and traffic shaping features. 2) QoS signalling techniques for coordinating QoS from end-to-end between network elements. 3) QoS policing and management functions to control and administer end-to-end traffic across a network.

Not all QoS techniques are appropriate for all network routers. Because edge routers and backbone routers in a network do not necessarily perform the same operations, the QoS tasks they perform might differ as well. To configure an IP network for real-time voice traffic, for example, the functions of both edge and backbone routers in the network are considered, then appropriate QoS features are selected.

In general, edge routers perform QoS functions including: packet classification, admission control, and configuration management. Backbone routers perform QoS functions including: congestion management, and congestion avoidance.

2.3.2. End-to-End Quality of Service Models

A service model, also called a level of service, describes a set of end-to-end QoS capabilities. End-to-end QoS is the ability of the network to deliver service required by specific network traffic from one end of the network to another. Basically there are three types of services models: best effort, integrated, and differentiated services.

- BEST-EFFORT MODEL

Best effort is a single service model in which an application sends data whenever it must, in any quantity, and without requesting permission or first informing the network. For best-effort service, the network delivers data if it can, without any assurance of reliability, delay bounds, or throughput. Typically, the QoS feature that implements best-effort service is first-in, first-out (FIFO) queuing. Best-effort service is suitable for a wide range of network applications such as general file transfers or e-mail.

- INTEGRATED SERVICE (INTSERV)

IntServ is a multiple service model that can accommodate multiple QoS requirements. In this model the application requests a specific kind of service from the network before sending data. The request is made by explicit signalling; the application informs the network of its traffic profile and requests a particular kind of service that can encompass its bandwidth and delay requirements. The application is expected to send data only after it gets a confirmation from the network. It is also expected to send data that is within its described traffic profile.

Resource Reservation Protocol (RSVP) can be used by applications to signal their QoS requirements to the router, and intelligent queuing mechanisms can be used with RSVP to provide the different kinds of services including: Guaranteed Rate Service and Controlled Load Service. The Guaranteed Rate Service allows applications to reserve bandwidth to meet their requirements, and the Controlled Load Service allows applications to have low delay and high throughput even during times of congestion.

- DIFFERENTIATED SERVICE (DIFFSERV)

DiffServ is a multiple service model that can satisfy differing QoS requirements. Unlike the IntServ model, an application using differentiated service does not explicitly signal the router before sending data. The network tries to deliver a particular kind of service based on the QoS specified by each packet. This specification can occur in different ways; for example, it can use the IP Precedence bit settings in IP packets or source and destination addresses. The network uses the QoS specification to classify, shape, and policy traffic, and to perform intelligent queuing.

The DiffServ model is used for several mission-critical applications and for providing end-to-end QoS. This service model is appropriate for aggregate flows because it performs a relatively coarse level of traffic classification. QoS features supported by the DiffServ model include: Committed Access Rate (CAR) and Intelligent Queuing Mechanisms. CAR performs packet classification through IP Precedence and QoS group settings, and it also performs metering and policing of traffic, providing bandwidth management. An intelligent queuing scheme, such as WRED, WFQ and their equivalent features on the versatile interface processor (vip), can be used with CAR to deliver differentiated services.

2.3.3. QoS Techniques & Services

Table 2.1 lists current QoS techniques and features [Cisc20]. Some specific details will be discussed in this section.

Table 2.1. QoS technique features.

QUEUING TECHNIQUES	FOR CONGESTION MANAGEMENT ON OUTBOUND TRAFFIC
First In, First Out (FIFO) Queuing	Basic Store and Forward
Priority Queuing (PQ)	Basic Traffic Prioritization
Custom Queuing (CQ)	Advanced Traffic Prioritization
Weighted Fair Queuing (WFQ)	Intelligent Traffic Prioritization
Weighted Round Robin (WRR)	Traffic Taking Turns
VIP-Distributed WRED/WFQ.	Virtual Interface Processor Distributed WRED/WFQ
RANDOM EARLY DETECTION	**FOR CONGESTION AVOIDANCE ON OUTBOUND TRAFFIC**
Weight Random Early Detection (WRED)	Avoiding Congestion
WRED Cooperation	with QoS Signaling Technologies
Distributed-WRED	Delivering High-Speed Differentiated Traffic
TRAFFIC SHAPING OR TRAFFIC LIMITING TECHNIQUES	**FOR CONTROLLING BANDWIDTH**
Generic Traffic Shaping (GTS)	Controlling Traffic on Non-Frame Relay Interfaces
Committed Access Rate (CAR)	Controlling Traffic at a Committed Rate
Frame-Relay Traffic Shaping (FRTS)	Controlling Traffic on Frame Relay Interface and Subinterfaces
Limiting Bandwidth	Limiting Bandwidth and Optionally Coloring Traffic
TRAFFIC COLORING	**FOR INTELLIGENT CLASSIFICATION**
IP Precedence	IP ToS Singling Classification
Committed Access Rate (CAR)	Packet Classification

2.3.3.1 IP Precedence and Type of Service

Figure 2.4 shows the Internet Protocol version 4 (IPv4) header format, with an emphasis on the type-of-service byte near the beginning of the header.

The IP type-of-service byte specifies both precedence and type of service. *Precedence* helps a router determine which packet to send when several packets are queued for transmission to the same output interface. Type of Service helps a router select a routing path when multiple paths are available. The type-of-service byte is divided into two fields (that are followed by a bit that is always set to zero): a) the 3-bit precedence field supports eight levels of priority; b) the 4-bit type-of-service field supports four types of service.

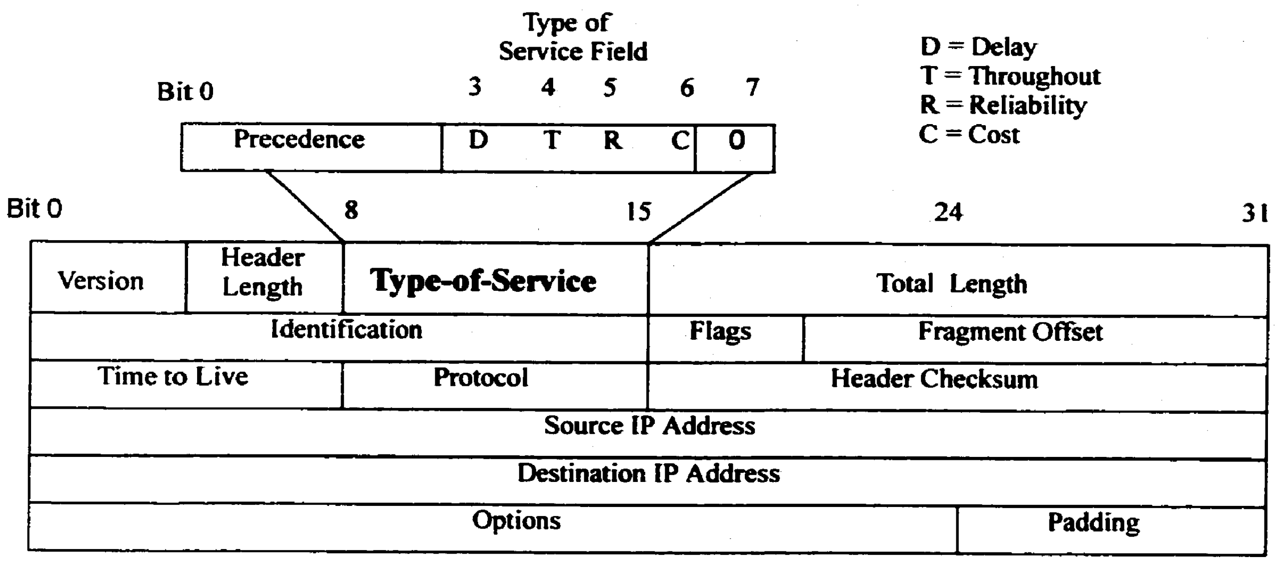

Fig. 2.4. The Internet Protocol Version4 (IPv4) header format.

- IP PRECEDENCE FIELD

An application can use the precedence field to specify the importance of a packet. The importance can range from routine priority (the bits are set to 000) to high priority (the bits are set to 111). One of the original uses envisioned for the precedence field was congestion control. By giving congestion-control packets a precedence of 7 (111 in binary), an application can specify that the congestion-control packets are more important than data packets, thus facilitating the effectiveness of the congestion-control mechanism. By selecting a precedence of 7, the application hopefully is ensuring that the congestion-control packets are not affected by the congestion that they are trying to control.

Precedence values 6 and 7 are reserved for network and internetwork control packets. Values 0-5 can be used for applications and user data. Some service providers offer a premium service that uses the precedence field to mark a customer's traffic as high-priority data. Many routers, and some hosts, support configuring the precedence value for applications. The precedence value is typically set to 5 for Voice over IP (VoIP) and other real-time applications.

- IP TYPE-OF-SERVICE FIELD

The type-of-service field helps a router select a route from a set of routes with different characteristics. A telephone circuit, for example, has low delay but limited throughput. A satellite link has high throughput; it also has high delay because of the distance to the satellite. A radio channel might be inexpensive, but error prone. Routing protocols attempt to determine the "best" route to a destination, but there are several definitions of "best": cheapest, fastest, most reliable, and so on. Some routing protocols, notably OSPF (Open Shortest Path First) and BGP (Border Gateway Protocol), support selecting a route based on the type of service that an application specifies.

The type-of-service field within the type-of-service byte in an IP header has four bits, referred to Fig. 2.4. The delay bit (D) tells routers to minimize delay, the throughput bit (T) tells routers to maximize throughput, the reliability bit (R) tells routers to maximize reliability, and the cost bit (C) tells routers to minimize monetary cost.

According to RFC 1349 [RFCs], an application can set one (and only one) of the four type-of-service bits to tell routers in the path to the destination how to handle the packet. A host can also use the type-of-service bits to decide which initial path to a local router to select. If all four bits are zero, then routine service is implied.

A router uses the type-of-service field to select a path that has the best chance of meeting an application's service requirements. The router is not required to offer any guarantees about service.

1) Interactive applications, such as Telnet and Rlogin, set the D bit. Voice and video applications can also set the D bit. When the D bit is set, a router should select a path that minimizes delay, for example, a dedicated high-speed leased line instead of a shared Frame Relay link.

2) File transfer applications, or any applications that send bulk data, set the T bit. Setting the T bit tells routers to select high-capacity links.

3) Routing protocols and network-management applications set the R bit. Setting the R bit tells routers to select fault-tolerant paths.

4) Applications for which neither delay nor throughput are critical, but a low monetary cost is important, setting the C bit. The Network News Transfer Protocol (NNTP), which reads Usenet news, sets the C bit, presumably because reading news is not a critical activity and should not use a lot of monetary resources. The C bit was more recently defined than the other bits and is not widely used yet.

2.3.3.2 Resource Reservation Protocol

The Resource Reservation Protocol (RSVP) is an alternative to the IP type-of-service and precedence capabilities inherent in the IP header. It supports more sophisticated mechanisms for hosts to specify QoS requirements for individual traffic flows. RSVP can be deployed on LANs and Internets to support multimedia applications or other types of applications with strict QoS requirements.

Unlike the type-of-service byte in the IP header that offers in-band signaling (i.e., bits within the frame header signal to routers how the frame should be handles), RSVP serves as an out-of-band signaling, that is, hosts send additional frames, beyond data frames, to indicate that a certain QoS service is desired for a particular traffic flow.

RSVP is a setup protocol used by a host to request specific qualities of service from the network for particular application data streams or flows. It is also used by routers to deliver QoS requests to other routers along the path of a flow. RSVP operates on top of IP version 4 or 6, occupying the place of a transport protocol in the protocol stack (although it does not transport application data).

RSVP provides a general facility for creating and maintaining information on resource reservations across a mesh of unicast or multicast delivery paths. RSVP transfers QoS parameters, but does not define the parameters or the different types of services that an application can request. RSVP simply passes parameters to the appropriate traffic-control and policy-control modules for interpretation.

RSVP is a relatively new protocol, having been proposed by the IETF (Internet Engineering Task Force) ISWG (Internet System Work Group) in RFC 2205 [RFCs] in September 1997. Nonetheless, it is supported by most router vendors and many multimedia application developers, including Microsoft, Intel, and Sun.

2.3.3.3 Queuing Mechanisms & Services

To optimize a network that is experiencing congestion, intelligent and fast queuing methods are also necessary. Queuing allows a network device to handle an overflow of traffic. Queuing methods here include: First-in First-out (FIFO) queuing, Priority queuing (PQ), Custom queuing (CQ), and Weighted fair queuing (WFQ).

- FIRST-IN FIRST-OUT QUEUING

FIFO queuing provides basic store-and-forward functionality. It involves storing packets when the network is congested and forwarding them in the order they arrived when the network is no longer congested. FIFO has the advantage that it is the default queuing algorithm in some instances, so it requires no configuration. FIFO has the disadvantage that it makes no decision about packet priority. The order of arrival determines the order a packet is processed and outputed.

FIFO provides no QoS functionality and no protection against an application using network resources in a way that negatively affects the performance of other applications. Bursty sources can cause high delays in delivering time-sensitive application traffic, and potentially defer network control and signaling messages. Long packets can cause unfair delays for applications that use short packets. For these reasons, advanced queuing algorithms were developed, which provide more features than basic FIFO queuing, including priority queuing, custom queuing, and weighted-fair queuing.

- PRIORITY QUEUING

Priority queuing ensures that important traffic is processed first. It was designed to give strict priority to a critical application, and is particularly useful for time-sensitive protocols such as SNA (System Network Architecture). Packets can be prioritized based on many factors, including protocol, incoming interface, packet size, and source or destination address.

Priority queuing is especially appropriate in cases where WAN links are congested from time to time. If the WAN links are constantly congested, the customer should investigate protocol and application inefficiencies, consider using compression, or possibly upgrade to more bandwidth. If the WAN links are never congested, priority queuing is unnecessary. Because priority queuing requires extra processing and can cause performance problems for low-priority traffic, it should not be recommended unless necessary.

Priority queuing has four queues: high, medium, normal, and low. The high-priority queue is always emptied before the lower-priority queues are serviced, as shown in Fig. 2.5.

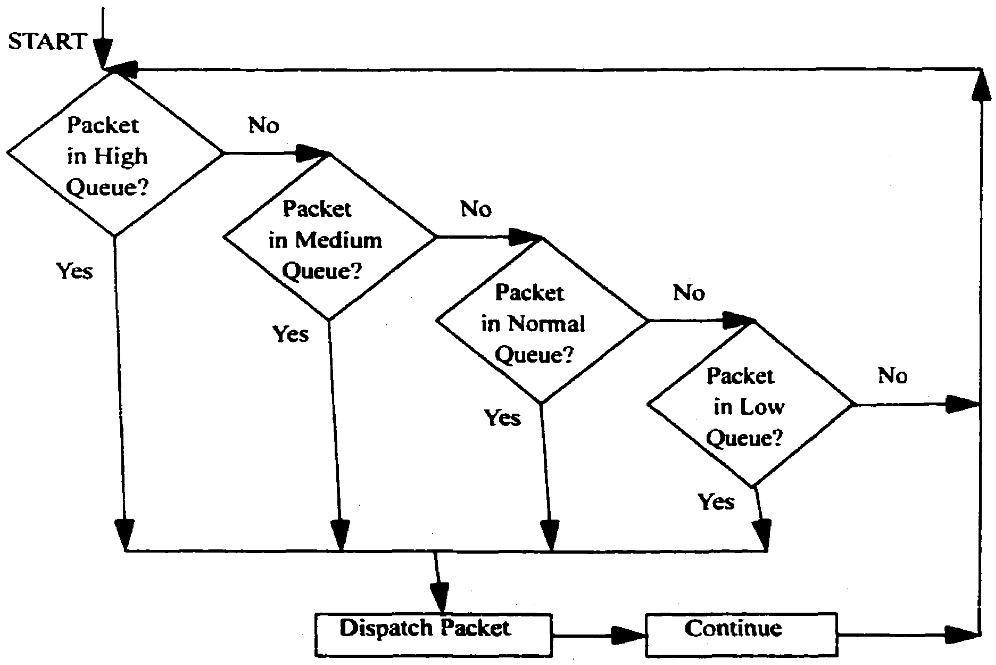

Fig. 2.5. The behavior of Priority Queuing.

- Custom Queuing

Custom queuing was designed to allow a network to be shared among applications with different minimum bandwidth or latency requirements. Custom queuing assigns different amounts of queue space to different protocols and handles the queues in round-robin fashion. A particular protocol can be prioritized by assigning it more queue space. Custom queuing is more "fair" than priority queuing, although priority queuing is more powerful for prioritizing a single critical application.

Custom queuing is used to provide guaranteed bandwidth at a potential congestion point. It assures each specified traffic type a fixed portion of available bandwidth, while at the same time avoids any application achieving more than a predetermined proportion of capacity when the link is under stress.

Custom queuing places a message in one of 17 queues. The router services queues 1 through 16 in round-robin order. Queue 0 holds system messages, such as keep alive and signaling messages, and is emptied first. A network administrator configures the transmission window size of each queue in bytes. Once the appropriate number of packets are transmitted from a queue such that the transmission window size has been reached, the next queue is checked, as shown in Fig. 2.6.

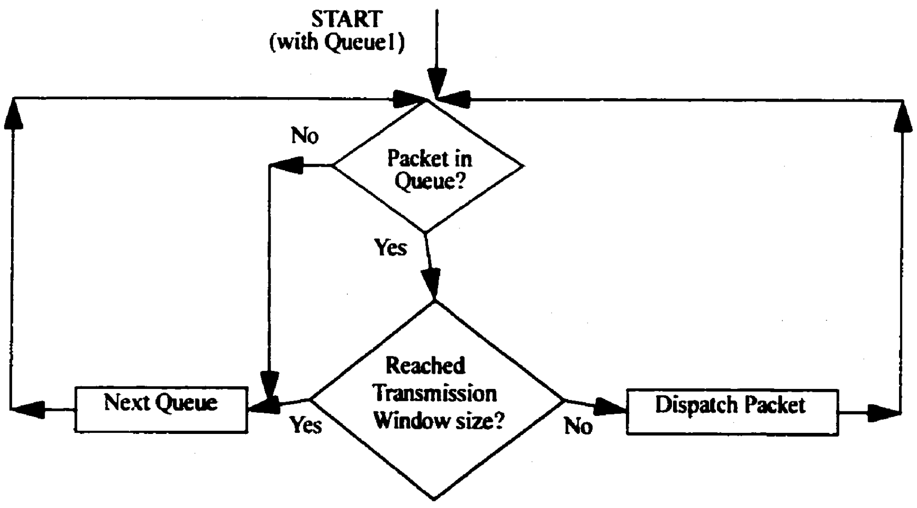

Fig. 2.6. The behavior of Custom Queuing.

Many businesses use custom queuing for SNA traffic. Because SNA is delay sensitive, one can set aside some portion of bandwidth, often 40 to 50 percent, for the SNA traffic to use during times of congestion.

- WEIGHTED FAIR QUEUING

WFQ is a sophisticated set of algorithms designed to reduce delay variability and provide predictable throughput and response time for traffic flows. A goal of WFQ is to offer uniform service to light and heavy network users alike. WFQ ensures that the response time for low-volume applications is consistent with the response time for high-volume applications. Applications that send small packets are not unfairly starved of bandwidth by applications that send large packets.

WFQ is a flow-based queuing algorithm that recognizes a flow for an interactive application and schedules that application's traffic to the front of the queue to reduce response time. Low-volume traffic streams for interactive applications, which include a large percentage of applications on most networks, are allowed to transmit their entire offered loads in a timely fashion. High-volume traffic streams share the remaining capacity.

Unlike custom and priority queuing, WFQ adapts automatically to changing network traffic conditions and requires little to no configuration. It is the default queuing mode on most serial interfaces configured to run at or below E1 speeds (2.048 Mbps).

For applications that use the IP precedence field in the IP header, WFQ can allot bandwidth based on precedence. The algorithm allocates more bandwidth to conversations with higher precedence, and makes sure those conversations get served more quickly when congestion occurs. WFQ assigns a weight to each flow, which determines the transmit order for queued packets. IP precedence helps determine the weighting factor.

WFQ also works with RSVP. RSVP uses WFQ to allocate buffer space, schedule packets, and guarantee bandwidth based on resource reservations. Additionally, WFQ understands the discard eligibility bit (DE), and the forward explicit congestion notification (FECN) and backward explicit congestion notification (BECN) mechanisms in a Frame Relay network. Once congestion is identified, the weights used by WFQ are altered so that a conversation encountering congestion transmits less frequently.

2.4. Network Switching

2.4.1. General Concepts of Network Switching

NETWORK SWITCHING technology is about 100 years old. Today, switching uses two technologies: digital circuit switching and digital packet switching.

2.4.1.1 Why Do We Need Switching?

Hundreds of million people are connected together every day by thousands of worldwide telecommunications networks. At the heart of this communications effort is the telecommunications switch, some of which are designed to transmit primarily voice and low-throughput data signals, while others are specialized switches that connect high volumes of voice, data, and video signals.

Telecommunications switches can be found everywhere. For example, if we here in Canada make a call to our friends in Australia, how many switches does it take to make these connections? For this international call, we would probably use the following switches: at least one local exchange switch, a toll gateway to our long distance carrier, perhaps one or more long distance toll switches, an international gateway switch in Canada, an international gateway switch in Australia, then perhaps one or two long distance and local switches in Australia as well. A minimum of five to seven switches are needed to complete this call. Therefore, switching is very important, and is actually is very complicated as well.

2.4.1.2 Origin and Types of Switching

First came the telephone, then the telephone operator, then the telephone switch. The first telephones in commercial use were directly connected to each other by a hotline (or private line). As the number of customers grew, the concept of an operator-controlled switch board was developed. The first widely used automatic switching system was patented in 1891 by a Kansas City undertaker, A. B. Strowger. Since then a number of different switches have been used in the world's telecommunications networks, such as the Lucent Technologies 5ESS® switch which has evolved since its introduction in 1985 by Bell Labs to become one of the most flexible and efficient switches today.

The first switches used in telecommunications networks were called *Analog Circuit Switches*, including the electro-mechanical step-by-step switches of the 1930s and 1940s, and the original electronic switches of the 1960s, such as the 1ESS®. There are very few mechanical or electro-mechanical switches in use today.

Digital Circuit Switching was introduced in the early 1970s. The 4ESS® toll switch was developed by Bell Labs in 1974. Instead of using the continuous waveform of analog, the digital circuit switch encodes voice signals in the form of binary 1s and 0s, and routes them through the assigned circuit path. That is, each digitally encoded call occupies the path through the switch called the switching matrix, at different fractions of time called time slots. Since the time slot is assigned to the call for its duration, it can not be used by any other call, so the call is said to have its own separate path through the switch. Almost all the switches in a modern telecommunications network today are *digital switches*.

With the increasing amount of data traffic being transmitted on the world's networks, the holding time is becoming a problem with the use of circuit switching to route this traffic. The longer holding time means the resources of the switch are not made available to other callers. In response to this problem, *Packet Switches* have been introduced to handle data calls. These switches use the concept of data packets, which involves taking a user's data stream and breaking it down into small chunks, or packets, and routing the packets through the network. In a packet switch, incoming traffic is passed through the switch on a first-come, first-served basis, and packet traffic is routed according to the address in the packet header. All the packets travel through the switch on a multiplexed bus, similar to a circuit switch, but there is no predetermined time slot when the packet can be sent. If the user isn't sending a message, the switch can transmit the message packets of other users. The holding times of data users no longer affect the efficiency of the switch.

2.4.1.3 The Evolution of Packet Switching

In the past few years, packet switching networks and services have evolved substantially, including the following stages:

- X.25 NETWORKS

X.25 protocol was used in the early packet-switching networks. Data was "piggy-backed" on existing circuit-switched networks, which limited throughput to multiplexed DS0 rates of 64Kb/s. The extensive error checking was performed at each switching node, which ensured very low error rates, but also precluded high transmission rates.

- FRAME RELAY NETWORKS

As error correction techniques improved, faster data networks become possible. Frame relay sends data in variable-sized packets called frames, which reduces the amount of error checking and permits higher throughputs. However, this type of frame sometimes causes larger frames that are transmitted first to arrive at their destination after subsequently transmitted smaller packets. This is acceptable for data frames that can be reordered at their destination, but it rules out packet switching for real-time voice traffic.

- CELL RELAY NETWORKS

The next packet service to be developed was cell relay, in which data are placed in fixed-size cells. This one-size-fits-all technology eliminates the timing problems of frame relay, but it presents problems of its own. Since small-sized cells are efficient of switching voice traffic, and large-sized cells are more efficient for data traffic, both voice and data can be transmitted, but not efficiently, on the same cell relay network.

- ISDN NETWORKS

Integrated Services Digital Network (ISDN) evolved in the 1980s as a new network transport service, providing customers with a single access channel to a network for voice, data and video. However, ISDN still requires either circuit or packet switches and their separate underlying networks to transmit the data through a network. The ISDN limits bit rates to multiples of 64Kb/s or 1.544Mb/s (DS1) in the digital hierarchy; the Broadband ISDN (B-ISDN) then was developed for higher rate transmission, supporting data rates of up to 45Mb/s (DS3).

- ATM NETWORKS

In the early 1990s, the asynchronous transfer mode (ATM) with a fixed cell size of 53 octets, was introduced for the perfect all-purpose network. ATM provides different classes of service for voice, data and video, all on a single network. ATM begins to combine for the first time the functionality of both transmission and switching into one network element for use with synchronous optical network (SONET) and synchronous digital hierarchy (SDH) optical transmission systems.

2.4.2. Network Core Switching

2.4.2.1 Network Switching Taxonomy

There are two fundamental approaches in building a network core: *Circuit Switching* and *Packet Switching*. In circuit-switched networks, the resources that need to pass along a path (buffers, link bandwidth) to provide communication between the end systems are reserved for the duration of the session. In packet-switched networks, these resources are not reserved and session messages use the resource on demand, so the packet may have to wait (i.e., queue) for access to a communication link.

A link in a circuit-switched network can employ either FDM (Frequency Division Multiplexing) or TDM (Time Division Multiplexing). With FDM, the frequency spectrum of a link is shared among the connections established across the link. With a TDM link, time is divided into frames of fixed duration and each frame is divided into a fixed number of time slots. The ubiquitous telephone networks are examples of circuit-switched networks.

Packet-switched networks include *Datagram networks* and *Virtual-circuit networks*. Any network that routes packets according to destination addresses is called a datagram network, such as the IP protocol of the Internet, and any other network that routes packets according to virtual-circuit numbers is called a virtual-circuit network, such as, X.25, frame relay, and ATM.

In a virtual-circuit network, a virtual circuit (VC) consists of a path (i.e., a series of links and packet switches) between the source and destination hosts, virtual circuit numbers (one number for each link along the path), and entries in VC-number translation tables in each packet switch along the path.

In a datagram network, each packet that traverses the network contains its header with the destination address. When a packet arrives at a packet switch in the network, the packet switch examines a portion of the packet's destination address and forwards the packet to an adjacent switch. A routing table in a packet switch is used to map the packet's destination address for sending packets to an outbound link.

Not all telecommunication networks can be neatly classified as pure circuit-switched networks or pure packet-switched networks. For example, with networks based on the ATM technology, a connection can make a reservation and yet its messages may still wait for congested resources.

2.4.2.2 Network Layer Switches

The network layer is the basis of the transportation layer that provides communication service between two processes running on two different hosts. The role of the network layer is thus to transport packets from a sending host to a receiving host. To do so, three important network layer functions can be identified:

Path determination: The network layer must determine the route or path taken by packets as they flow from a sender to a receiver. The algorithms that calculate these paths are referred as routing algorithms. The two most prevalent classes of routing algorithms are link state routing and distance vector routing.

Switching: When a packet arrives at the input to a router, the router must move it to the appropriate output link.

Call Setup: Similarly to the three-way handshaker of a TCP connection, which allows the sender and receiver to setup the needed state information (e.g., sequence number and initial flow control window size), some network layer architectures (e.g., ATM) require that the routers along the chosen path from source to destination handshake with each other in order to setup state before data actually begins to flow. The network layer of the Internet architecture does not perform any such call setup.

2.4.2.3 Network Service Models

Network service model is the service model provided by the network layer; it defines the characteristics of end-to-end transportation of data between sending and receiving end systems. There are two such kinds of models: *VCs-based service* and *Datagram-based service.*

ATM architecture which uses virtual circuits (called virtual channels in the ATM jargon) provides for multiple service models. ATM networks can guarantee bandwidth, guarantee no packet loss, transmission ordering, maintained timing and congestion indication; that is, ATM can keep a specific degree of Quality of Service.

The Internet uses a datagram network layer, also called IP networks. The current Internet architecture provides only one service model, the datagram service, which is also known as "best effort service". With this sort of "no service at all", timing between packets is not guaranteed to be preserved, packets are not guaranteed to be received in the order in which they were sent, and the eventual delivery of transmitted packets is not guaranteed. Thus, a network which delivered *no* packets to the destination would satisfy the definition of best effort delivery service. Indeed, today's congested public Internet might sometimes appear to be an example of such a network. Internet's best-effort only service model is being extended to include so-called "Integrated Services" (IntServ) and "Differentiated Service" (DiffServ), two kinds of Quality of Service (QoS) models. QoS implementation over Internet Protocol (IP) is still a hot topic and a challenging issue in the area of telecommunication networks.

In the next chapter, we will discuss some advanced methodologies of intelligent switching, which are progressive approaches to QoS over IP networks.

Chapter 3

Methodology of Intelligent Switching

In this Chapter, we discuss some new and advanced methodologies of intelligent switching design, based on the TCP/IP (or OSI) reference model [Stal97] [Tane96].

3.1. Intelligent Network Switching

Network switching actually originated from an electrical device, and real network switching usually means Data Link (layer-2) switching. At the same time, we also refer to network routers as Network Layer (layer-3) switching, or simply switching. The router usually receives data (or packets) from the input port, through switching fabrics and routing processor based on routing table and queuing scheduling, and then scheduled packets on the output port are sent out to next routers or local networks. Using this sort of simple routing approach, because the router's buffer space is limited, serious packet loss and traffic congestion will occur, especially when providing UDP service that is unreliable connection. To meet the requirements of today's critical networks, good quality of services is absolutely necessary and is becoming an emerging feature. Therefore, the first point about intelligent network switching to ne addressed here is Network Layer Switching with QoS feature.

Figure 3.1 shows intelligent network switching based on the TCP/IP (or OSI) reference model. In the TCP/IP reference model, the Internet stack consists of five layers: the physical, data link, network, transport and application layers. The PDUs (Protocol Data Units) in the five layers are: 1-PDU (Bits), frame, datagram (called segmented by some reference), packets (called datagram by some reference), and message.

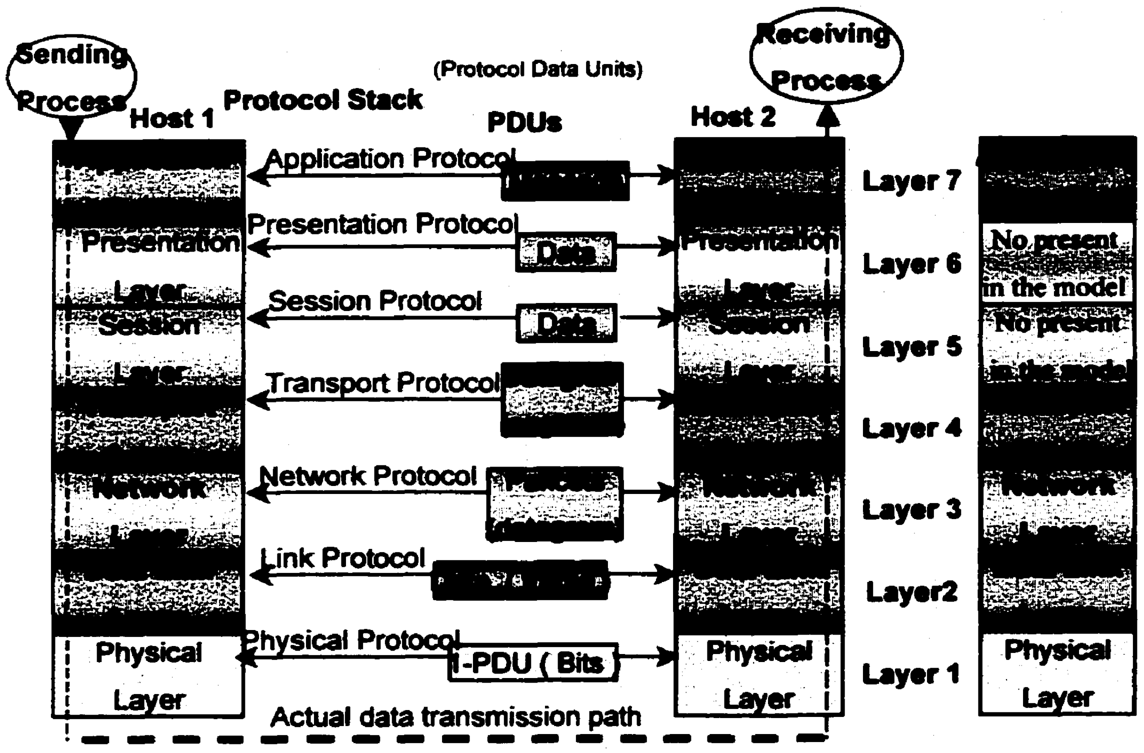

Fig. 3.1. Data transmission in the OSI and TCP/IP reference model.

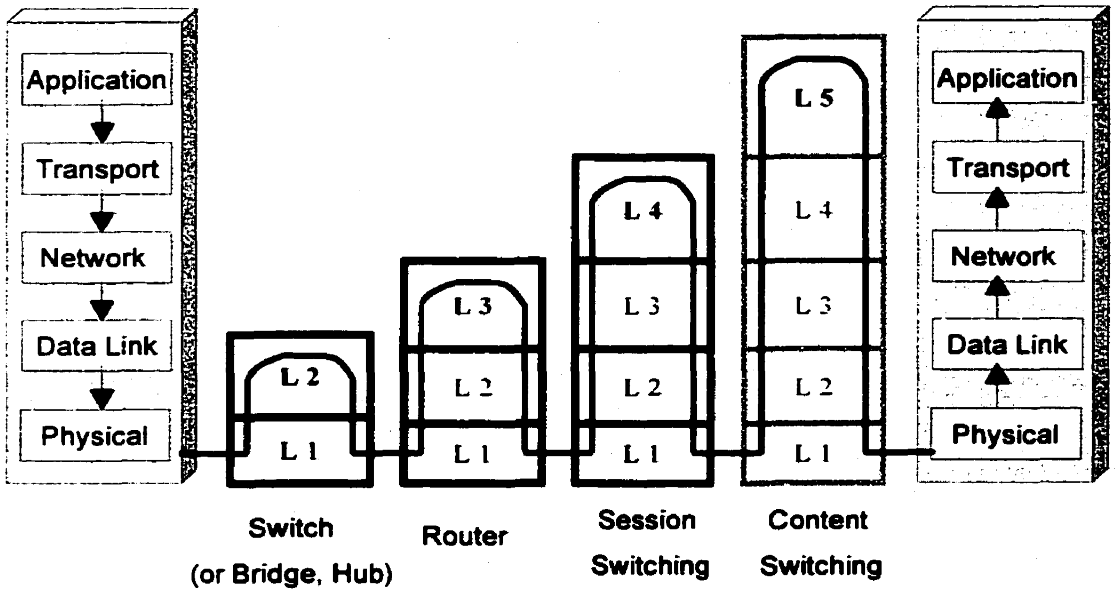

Fig. 3.2. Intelligent network switching in the TCP/IP reference model.

A protocol layer can be implemented in software, in hardware, or using a combination of the two. Application-layer protocols, such as HTTP and SMTP (Simple Mail Transfer Protocol) are almost always implemented in software in the end system, as are transport layer protocols. Because the physical and data link layers are responsible for handling communication over a specific link, they are typically implemented in a network interface card (e.g., Ethernet or ATM interface cards) associated with a given link. The network layer is often a mixed implementation of hardware and software. A brief introduction of five layers and their services is described in the following:

- APPLICATION LAYER: is responsible for supporting network applications. The application layer includes many protocols, including HTTP to support the Web, SMTP to support electronic mail, and FTP to support file transfer. We also can create our own new application-layer protocols.
- TRANSPORT LAYER: is responsible for transporting application-layer messages between the client and server sides of an application. In the Internet there are two transport protocols, TCP and UDP, either of which can transport application-layer messages. TCP provides a connection-oriented service to its applications. This service includes guaranteed delivery of application-layer messages to the destination and flow control. TCP also segments long messages into shorter segments and provides a congestion control mechanism. The UDP protocol provides its applications with a connectionless service.
- NETWORK LAYER: is responsible for routing datagrams from one host to another. The Internet's network layer has two principal components. First it, has a protocol that defines the field in the IP datagram as well as how the end systems and routers act on this field. This is the celebrated IP protocol. The Internet's network layer also contains routing protocols that determine the routes that datagrams take between source and destination. The Internet has many routing protocols, such as RIP (Routing Information Protocol), OSPF (Open Shortest Path First), and BGP (Border Gateway Protocol).
- DATA LINK LAYER: the network layer must rely on the services of the data link layer. In particular, at each node IP passes the datagram to the link layer, which delivers the datagram to the next node along the route. At this next node, the link layer passes the IP datagram to the network layer. The services provided at the data link layer depend on the specific link-layer protocol that is employed over the link. Examples of data link layer include Ethernet and PPP

(Point-to-Point Protocol); in some contexts, ATM and frame relay can be considered as data link layers.

- PHYSICAL LAYER: the job of the physical layer is to move the *individual bits* within the network frame from one node to the next. The protocols in this layer are again link dependent, and further depend on the actual transmission medium of the link. For example, Ethernet has many physical layer protocols: one for twisted-pair copper wire, another for coaxial cable, another for fiber, etc. In each case, a bit is moved across the link in a different way.

In Fig. 3.1 and Fig. 3.2, the communication from one host (source) to another host (destination), data transmission paths from application layer, transport layer, network layer, data link layer, and physical layer, then goes up to another host's physical layer, data link layer, network layer, transport layer, and application layer. With this kind of communication between host-to-host (or end-to-end), switches are needed to route data across different networks, or the Internet. Based on the layer structure in the TCP/IP reference model, a variety of switching can be defined: a) if the switching action is based only on the physical and data link layers, the general switching is used; it is called LAYER-2 SWITCHING (or SWITCHING) (for example, Bridge and Hub). b) once the network layer is involved, such as with routing protocols and algorithms, and network topology, a router switching is needed, it is called as LAYER-3 SWITCHING (or ROUTER). c) if the included transport layer information (i.e. TCP/UDP header) is used for switching, it is referred to LAYER-4 SWITCHING (or SESSION SWITCHING). d) finally, once the ability to use information contained in the payload of packets of application layer is added, with which more sophisticated capabilities could be provided for today's mission critical networks, we call it LAYER-7 SWITCHING (or CONTENT SWITCHING). In the following section, more details of advanced technical methods for layer-3/4/7 intelligent switching are given.

3.2. Network Layer Switching Techniques

The key issue of the network layer is router function, and the major job of a router is to switch packets from incoming interfaces to outgoing interfaces. Switching involves receiving a packet, determining how to forward the packet based on the routing topology and QoS and policy requirements, and switching the packet to the right outgoing interface or interfaces. The speed at which a router can perform this task is a major factor in determining network performance in a routed network. Currently there many switching methods, with varying speeds and behaviors, supported by industry-leading technologies. This section describes one basic (old but still being used) and three newer switching technologies: classic packet switching methods, NetFlow switching, Cisco Express Forwarding (CEF), and Tag switching.

3.2.1 Classic Methods for Layer-3 Packet Switching

Process switching is the slowest of the switching methods. With process switching, when a packet arrives at an interface, the system processor is interrupted for the time it takes to copy the packet from the interface buffer to system memory. The processor looks up the layer-3 destination address for the packet in the routing table to determine the exit interface. The packet is rewritten with the correct header for that interface and copied to the interface. At this time, an entry is also placed in the fast-switching cache so that subsequent packets for the destination address can use the same header. The first packet to a destination is always process switched.

Fast switching allows higher throughput by switching a packet using an entry in the fast-switching cache that was created when a previous packet to the same destination was processed. With fast switching, a packet is handled immediately without scheduling an interrupt of the system processor. Fast switching is enabled by default for most protocols, so there are some different products available. An example is the Cisco 7000-series router.

3.2.2 NetFlow Switching

NetFlow switching is a relatively new switching mode that is optimized for environments where services must be applied to packets to implement security, QoS features, and traffic accounting. An example of such an environment is the boundary between an enterprise network and the Internet.

NetFlow switching identifies traffic flows between hosts, and then quickly switches packets in these flows at the same time that it applies services. NetFlow switching also lets a network manager collect data on network usage to enable capacity planning and bill users based on network and application resource utilization. The data can be collected without slowing down the switching process.

To maximize network scalability, a good design practice is to use NetFlow switching on the periphery of a network to enable features such as traffic accounting, QoS functionality, and security, and to use an even faster switching mode in the core of the network. At the core of the network, the switching mode should forward packets based on easily-accessible information in the packet, and generally should not spend time applying services.

3.2.3 Cisco Express Forwarding

Cisco Express Forward (CEF) [CiEF20], is a Cisco-patented technique for switching packets very quickly across large backbone networks and the Internet. Rather than relying on the caching techniques used by classic switching methods, CEF depends on a forwarding information base (FIB). The FIB allows CEF to be much less processor-intensive than other layer-3 switching methods, because the FIB tables contain forwarding information for all routes in the routing tables (whereas a cache contains only a subset of the routing information).

CEF evolved to accommodate Web-based applications and other interactive applications that are characterized by sessions of short duration to multiple destination addresses. CEF became necessary when it became clear that a cache-based system is not optimized for these types of applications. Consider a Web-surfing application, for example. When a user jumps to a new Web site, TCP opens a session with a new destination address. It is unlikely that the new destination is in the router's cache, unless it happens to be a destination the user, or other users, visited recently. This means that the packet is process switched, which is slow.

CEF improves switching speed, and avoids the overhead associated with a cache that continually changes, through the use of the FIB, which mirrors the entire contents of the IP routing table. An example is the Cisco 7500 router.

3.2.4 Tag Switching

Tag switching has a slightly different goal than the other switching methods covered. Whereas the other methods optimize the switching of a packet through a single router, tag switching optimizes packet-switching through a network of tag switches.

Tag switching is an implementation of the IETF standard for multiprotocol label switching (MPLS). The idea behind these algorithms is that by "labeling" or "tagging" the first packet in a flow of data, subsequent packets can be expedited to the final destination. Tagging minimizes the processing required of a router, and thus significantly reduces delay on packet and cell-based networks that include tag switches. In addition, packets can be tagged to travel along specified routes to implement load balancing, QoS, and other optimization features.

Tag information can be carried in a packet as a small header inserted between the layer-2 and layer-3 headers, or as part of the layer-3 header if the layer-3 protocol supports it. In IPv6 (Internet Protocol version 6), tag information can be included in the flow-label field. Some layer-2 implementations, such as ATM, support carrying the tag directly in the layer-2 header.

An internetwork that uses tag switching has three major components, as shown in Fig.3.3: Tag edge routers, Tag switches, and Tag Distribution Protocol.

- *Tag edge routers*, located at the boundaries of a network, perform network layer services and apply tags to packets.
- *Tag switches*, including routers and ATM or other layer-2 switches, switch tagged packets based on the tags. Tag switches can also support traditional layer-3 routing and layer-2 switching.
- *Tag Distribution Protocol* (TDP) distributes tag information between devices in a tag-switched network. Tag edge routers and switches use the tables generated by standard routing protocols to assign and distribute tag information via TDP.

When an ingress tag edge router receives a packet for forwarding across the tag network, it performs the following tasks: a) analyzes the network-layer header; b) performs applicable network-layer services such as security, NetFlow accounting, QoS classification, and bandwidth management; c) selects a route for the packet from the routing tables; d) applies a tag and forwards the packet to the next-hop tag switch.

When a tag switch receives a tagged packet, it switches the packet based solely on the tag without re-analyzing the network-layer header. The switch uses the tag as an index for its Tag Information Base (TIB). A TIB is a database of records that include the following elements: incoming tag, outgoing tag, outgoing interface, and outgoing link-level information.

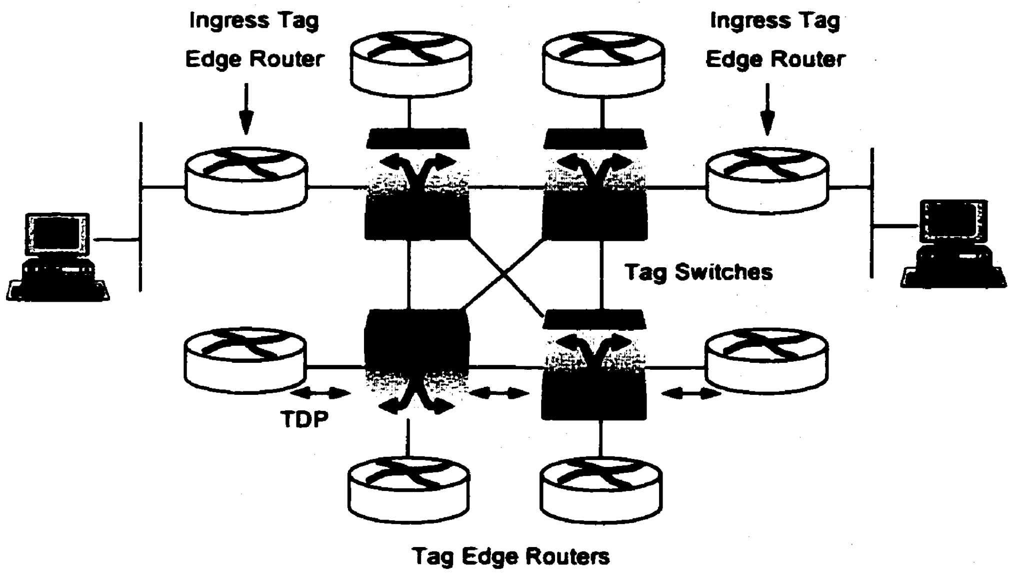

Fig. 3.3. The architecture of a network that uses tag switching.

When a switch finds an entry in the TIB with the incoming tag equal to the tag carried in the packet, the switch replaces the tag in the packet with the outgoing tag, replaces the link-level information (such as the MAC (Medium Access Control) address) in the packet with the outgoing link-level information, and forwards the packet over the outgoing interface.

When a packet reaches a tag edge router at the egress point of a network, the tag is stripped off and the packet delivered. As can be seen, tag switching fuses the intelligence of routing with the performance of switching. This approach allows networks to scale to a large size and handle more traffic, users, media-rich data, and bandwidth-intensive applications.

A typical application for tag-switching is a large IP network built from a core of ATM switches surrounded by edge routers.

3.3. Layer-4 Session Switching

3.3.1 Why and What Session Switching Is

- WHY IS SESSION SWITCHING?

Since the client/server computing revolution began, network managers and system administrators have been chasing bottlenecks. The layer-2 and layer-3 switching solutions at Fast Ethernet and Gigabit Ethernet speeds have been aimed at easing network bottlenecks and improving network application performance. These technologies have been so successful that the bottlenecks in many client/server computing systems have moved to the network/server interface and servers themselves. That is, the network is now faster than the computers attached to it. So, despite all the new investment in high-speed networking gear, users continue to perceive response times as unacceptably long because of the capacity mismatch between the network and servers.

Meanwhile, widespread deployment of Internet technology in Intranets, as well as the growth of the Internet itself, means that the potential number of clients that must be supported by a server has dramatically increased. Intranet servers must support many hundreds or even thousands of clients, while Internet servers must support literally millions. This growth in aggregate client demand has created an imbalance between client and server capacity. New intelligent switching technology can help here by scaling server capacity to match aggregate client demand and available network bandwidth.

- WHAT IS THE REALLY SESSION SWITCHING?

In the simplest terms, a session switch enables application-session switching, i.e., the ability to control and direct entire TCP/IP sessions. This requires keeping track of the initiation and completion of TCP connections. Session switches read transport layer, or layer-4, information and make switching decisions based on this information while maintaining the state of each established session.

With session switching, users gain the ability to transparently scale server capacity, shrink user response times, and increase application availability without creating any multivendor interoperability issues. So, new session switching technology is completely transparent to the clients, servers and other devices in the network in both configuration and operation. From the server side, server CPUs and network resources can be optimally utilized so that enough bandwidth is provided for users and congestion control can be reduced.

Some advanced technologies and methodologies are required for layer-4 switching: server clusters, server load balancing algorithms, and network address translation.

3.3.2 Server Farms (Server Clusters)

A Server Farm, or server cluster, is defined as a pool of servers tied together to act as a single unit, or *server clustering*, providing incremental scalability, as shown in Fig. 3.4.

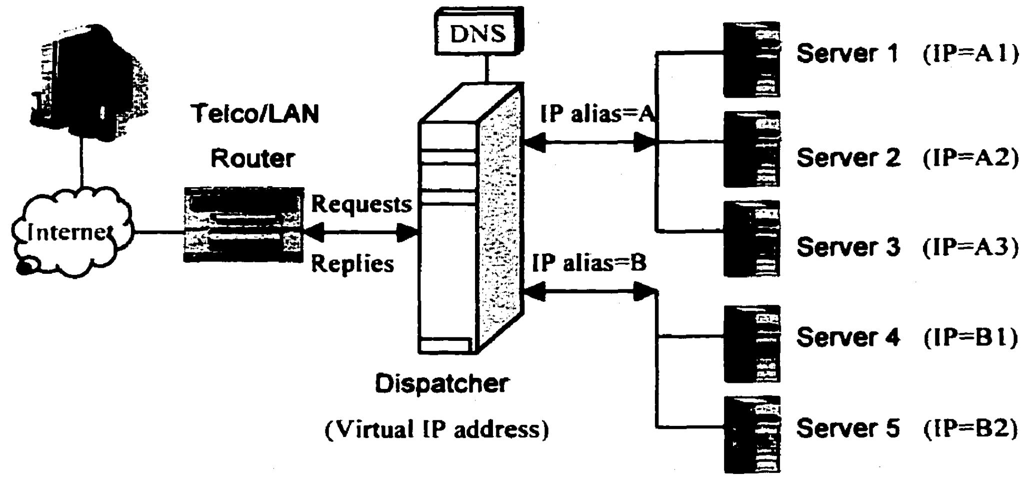

Fig. 3.4. The diagram of scalable server clusters.

In Fig. 3.4, a dispatcher connects a Telco/LAN router and server clusters. Only this dispatcher registers with DNS so that its IP address is published to clients; however, this IP address is actually is a Virtual IP address (VIP) because the dispatcher does not provide any real service to clients' request. Thus, this dispatcher acts as a front-end processor, redirecting traffic to the real servers of server clusters. Here, there are two group clusters: group A and group B. For example, if the VIP alias with group A, the traffic will relocate to server 1 (IP=A1) and/or server 2 (IP=A2) and/or server 3 (IP=A3); alternatively, the traffic could go to group B, connecting to server 4 (IP=B1) and/or server 2 (IP=B2); also the traffic connection could be partially group A and group B (for example, distributed to server 1, server 3, and server 5). The policy of selecting real servers to connect to depends on the intelligent switching scheduling, switching implementation, and application requirements.

The appearance of server clustering technology is mainly in response to the need for more and faster Web servers capable of serving over 100 million Internet users. The only solution for scaling server capacity in the past has been to complete replace the old server with a new one. Organizations must discard their investment in the old server and purchase a new one, so it is an expensive, short-term solution. A long-term solution requires incremental scalability, which provides the ability to grow gradually with demand. With scalable Web server clustering technologies, services providers may gradually add additional low-cost computers to augment the performance of existing servers. As Internet usage has grown, so has investigation into Web server clustering. The past four years have seen the emergence of several promising experimental server clustering approaches as well as a number of commercial solutions.

To reach full transparency to both Web clients and server clusters, the design and implementation of dispatcher could be different based on layer switching, that is, layer-4/2, layer-4/3, or layer-7. In addition, other techniques could be introduced into a pure dispatcher, such as LARD dispatcher (Locality Aware Request Distribution), and Cache plus dispatcher. Table 2.1 shows the feature comparisons of transparent clustering techniques, according to [ScGR20].

Table 3.1. A summary of transparent clustering techniques.

	Layer 4/2	Layer 4/3	Layer 7
Mechanism	Link-layer address translation.	Network address translation.	Content-based routing.
Flow	Incoming only.	Incoming/outgoing.	Varies.
Fault tolerance	Yes	Yes.	Yes
Restrictions	Physical interface on every network with server; incoming traffic passes through dispatcher.	Dispatcher lies between client and server.	All of incoming traffic passes through dispatcher; all outgoing as well, if caching.
Performance	Thousands of chips/s; hundreds of Mb/s.	Hundreds of chips/s; tens of Mb/s	Tens of thousands of chips/s; Gb/s
Bottleneck	Connection dispatching.	Integrity code calculations.	Connections dispatching complexity.
Advantage	Simple.	Flexible.	Server specialization.

3.3.3 Server Load Balancing Algorithms

Server load balancing (SLB) is the process of distributing service requests across a group of servers, or a server farm. Server load balancing addresses several requirements that are becoming increasing important in networks: the increased scalability, high performance, high availability and disaster recovery. We discuss the server farm connection scheduling algorithms in this section.

Basically, we have implemented four scheduling algorithms for selecting servers from the cluster for new connections: Round-Robin (RR), Weighted Round-Robin (WRR), Least-Connection (LC) and Weighted Least-Connection (WLC). The first two algorithms are self-explanatory, because they don't have any load information about the servers. The last two algorithms count the number of active connections for each server and estimate their load based on those connection numbers.

- ROUND-ROBIN SCHEDULING

The round-robin scheduling algorithm directs network connections to another different server in a round-robin manner. It treats all real servers as equal regardless of the number of connections or response time. Although the round-robin DNS (Domain Name Service) works in this way, it is quite different. The round-robin DNS resolves a single domain to different IP addresses, the scheduling method is host-based, and the caching of DNS make the algorithm ineffective, this leads to significant dynamic load imbalance among real servers. However, the scheduling method of the virtual server is network connection-based, and it is much superior to the round-robin DNS.

- WEIGHTED ROUND-ROBIN SCHEDULING

The weighted round-robin scheduling can treat real servers with different processing capacities. Each server can be assigned a weight, an integer value that indicates the processing capacity. The default weight is 1. For example, real servers, named A, B and C, have the weights 4, 3, and 2 respectively; a good scheduling sequence will be ABCABCABA in a scheduling period (mod sum(Wi)). In the implementation of the weighted round-robin scheduling, a scheduling sequence will be generated according to server weights after the rules of the virtual server are modified. Network connections are directed to different real servers based on the scheduling sequence in a round-robin manner.

The weighted round-robin scheduling doesn't need to count network connections for each real server, and the overhead of scheduling is smaller than that of dynamic scheduling algorithms, so it can have more real servers. However, it may lead to dynamic load imbalance among real servers if the load of requests vary greatly. In shortly, it is possible that most long requests may be directed to a real server.

The round-robin scheduling is a special instance of the weighted round-robin scheduling, in which all of the weights are equal. The overhead of generating the scheduling sequence after modifying the virtual server rules is trivial, and it doesn't add any overhead in real scheduling, so it is unnecessary to implement the round-robin scheduling alone.

- LEAST-CONNECTION SCHEDULING

The least-connection scheduling algorithm directs network connections to the server with the least number of established connections. This is one of the dynamic scheduling algorithms; it needs to count live connections for each server dynamically. At a virtual server where there is a collection of servers with similar performance, the least-connection scheduling is good for smoothing distribution when the load of requests vary a lot, because not all long requests will have chance to be directed to a server.

At first glance, least-connection scheduling can also perform well even there are servers of varying processing capacities, because the faster servers get more network connections. In fact, it cannot perform very well because of the TCP's TIME_WAIT state. The TCP's TIME_WAIT is usually 2 minutes; a busy web site often gets thousands of connections. For example, if the server A is twice as powerful as server B, server A has processed thousands of requests and kept them in the TCP's TIME_WAIT state, while server B is slowly getting its thousands of connections finished. Thus, the least-connection scheduling cannot get-load balanced among servers with various processing capacities.

- WEIGHTED LEAST-CONNECTION SCHEDULING

The weighted least-connection scheduling is a superset of the least-connection scheduling, a performance weight is assigned to each real server. A server with a higher weight value will receive a large percentage of live connections at any one time. The virtual server administrator can assign a weight to each real server, and network connections are scheduled to each server in which the percentage of current number of live connections for each server is a ratio to its weight. The default weight is one.

The weighted least-connections scheduling works as follows: supposing there are *n* real servers, each server *i* has weight Wi(i=1,...,n) and alive connections Ci (i=1,...n), ALL_CONNECTIONS is the sum of Ci(i=1,...,n), the next network connection will be directed to the server *j*, in which

$$(Cj/ALL\_CONNECTIONS)/Wj = \min \{(Ci/ALL\_CONNECTIONS)/Wi\} \qquad (i=1,...,n)$$

Since the ALL_CONNECTIONS is a constant in this lookup, there is no need to divide Ci by ALL_CONNECTIONS, it can be optimized as:

$$Cj/Wj = \min \{Ci/Wi\} \qquad (i=1,...,n)$$

The weighted least-connection scheduling algorithm requires more division than the least-connection. In the hope of minimizing the overhead of scheduling when servers have the same processing capacity, both the least-connection scheduling and weighted least-connection scheduling algorithms are implemented.

3.3.4 Load Sharing Network Address Translation

Network Address Translators (NATs) translate IP addresses in a datagram, transparent to end nodes, while routing the datagram. NATs have traditionally been used to allow private network domains to connect to global networks using as few as one globally unique IP address [EgFr94]. Extending the use of NATs is to offer load-sharing feature, where session load can be distributed across a pool of servers instead of a single server.

LOAD SHARE is defined as the spread of session load among a cluster of servers which are functionally similar or the same [SrGa98]. In other words, each of the nodes in a cluster can support a client session equally well with no discernible difference in functionality. Once a node is assigned to service a session, that session is bound to that node until termination. Sessions are not allowed to swap between nodes in the midst of a session. Load sharing may be applicable for all services if all hosts in the server cluster have the capability to carry out all services. Alternately, load sharing may be limited to one or more specific services alone and not to others.

LOAD SHARING NETWORK ADDRESS TRANSLATION (LSNAT) operates as the following: A client attempts to access a server by using the server virtual address. The LSNAT router transparently redirects the request to one of the hosts in the server pool, selected using a real-time load sharing algorithm. Multiple sessions may be initiated from the same client, and each session could be directed to a different host based on load balance across server pool hosts at the time. If load share is desired for just a few specific services, the configuration of LSNAT could be defined to restrict load share for just the services desired. In the case where the virtual server address is same as the interface address of a LSNAT router, server applications (such as telnet) on a LSNAT router must be disabled for external access on that address. This is the limitation of using an address owned by a LSNAT router as the virtual server address.

There are three phrases that LSNATs must monitor in relation to address translation: session binding, address lookup and translation, and session unbinding. a) *Session binding:* An incoming session should be associated with the address of a host in the server pool. This essentially sets the translation parameters for all subsequent datagrams pertaining to the session. For addresses that have static mapping, the binding happens at startup time. Otherwise, each incoming session is dynamically bound to a different host based on a load sharing algorithm. b) *Address lookup and translation:* Once session binding is established for a connection setup, all subsequent packets belonging to the same connection will be subject to session lookup for translation purposes. The source (destination) IP address, source (destination) TCP/UDP port, and IP/TCP/UDP/ICMP (Internet Control Message Protocol) header checksum will undergo translation for outbound (inbound) packets of a session. c) *Session unbinding:* When the end of session is detect, session unbinding happens.

There are two kinds of load-sharing algorithms: 1) local load share algorithms including: round-robin algorithm, least load first algorithm, least traffic first algorithm, least weighted load first algorithm, and ping to find the most responsive host. 2) distributed load share algorithms including: weighted least load first algorithm, and weighted least traffic first algorithm.

There are some limitations to use LSNATs. It is mandatory that all requests and responses pertaining to a session between a client and server be routed via the same LSNAT router. Another limitation of LSNATs is the inability to switch loads between hosts in the midst of sessions. The major advantage, however, is that it can be installed without changes to clients or servers.

3.3.5 How Layer-4 Switching Works

A layer-4 switch makes the packet forward decision using the information of TCP/UDP transport layer, i.e., based on TCP/UDP header information. Transmission Control Protocol (TCP) segment header format [Post81] and User Datagram Protocol (UDP) segment header format [PosU80] can be found in Appendix A in this thesis.

A layer-4 switch, or session switch, acts as a virtual front-end processor to a cluster of connected servers. A virtual IP (VIP) address is configured in the session switch for each application hunt group (i.e., each group of servers that supports a common application or set of applications). This VIP address is registered with and advertised by Domain Name Services (DNS). For example, if *www.win.trlabs.ca* were being run on three servers and the client load was balanced among them by a switch with a VIP address of VIP_1, DNS would advertise VIP_1 as the address of *www.win.trlabs.ca* and not the individual IP addresses of the physical servers.

When a service request is made, the session switch recognizes the start of session by identified the TCP SYN packet. It then uses one of several intelligent algorithms (i.e., server load balancing algorithms) to determine the best server to handle that request. Once the determination is made, the switch binds the session to the IP address of one of the physical servers in the application hunt group. The session switch maintains a binding table that associates each active session with the physical server to which it is assigned. Next, the session switch performs address substitution so that the server will receive packets for that session. The switch inserts that server's real IP address into the network layer destination address field in place of the VIP address and inserts that server's media access control (MAC) address into the data link layer destination address field in place of the switch's MAC address.

The session switch then forwards the connection request to the chosen server. All subsequent packets belonging to that session undergo the same address substitution process and are forwarded to the same physical server until the switch sees a session termination packet (i.e., a TCP FIN packet). Likewise, the session switch intercepts packets traveling from the physical server to the client, inserts the VIP address in the network layer address field in place of the physical server's real IP address, inserts its own MAC address into the data link layer destination address field in place of the server's MAC address, and forwards the modified frame to the client.

Upon receipt of a TCP FIN packet, the session switch performs the necessary address substitution, forwards the packet to the appropriate physical server (causing the server to tear down the connection), and removes the session-to-server binding from its binding table.

By looking at session layer information and making switching decisions based on this information, connections can be bound to physical servers to meet user-specified criteria, such as having an equal number of connections to each server or weighting traffic by the capacities of different servers. This requires that the chosen load balancing algorithm be executed whenever a new connection is opened, and that the information required to execute the load balancing algorithm be available and up to date.

3.4. Application Layer Content Switching

The layer-4 switching uses the transport layer (i.e., TCP/UDP) information for making packet-forwarding decisions. Layer-7 switching is defined as the devices that switch traffic based on URL and other application-layer level information found in the payload of packets, so it add the ability to provide more sophisticated capabilities. There are two advanced methodologies that currently used for intelligent switching products providing layer-4/7 services [GuRW 20], that is, URL-aware switching and transparent switching of HTTP traffic to proxy cache.

3.4.1 URL-aware Switching

URL-aware switching (also known as "content-smart switching") [Arro20] refers to the capacity of a switch located on the path between clients and servers to redirect HTTP requests to servers based on the URL specified by the client in its GET request. When a user enters a URL into a browser, the browser constructs an HTTP GET request which contains the URL and other HTTP client header information. A switch with URL-aware redirection will intercept the request to make a decision about the server to which the request should be directed. All this happens transparently to the client.

Redirecting HTTP traffic based on application-level information is not as simple as layer-4 switching. For any HTTP transaction, application-level information is not available until the TCP connection establishment phase has been completed. This means that connections cannot be redirected at a switch by simple peering into a SYN packet as is possible with basic layer-4 switching. The TCP connection request from the client needs to be accepted at the switch by an application-layer proxy, and the connection must be established between the client and the switch before any application-level information is received. Once the application-level information is received, this information is parsed to determine which back-end server should receive this request and the request is redirected.

The TCP splicing mechanism has been proposed [MaBh98] in response to this challenge. When the client sends a request to a switch or gateway located between the client and servers, one TCP connection is opened. Next, another TCP connection is established between the switch and the back-end server; the client request is passed to the server through that connection, the response is received at the switch from the server on the connection and transferred through the other connection to the client. Once the two TCP connections are established, they are "spliced" or "patched" together so that IP packets are forwarded from one TCP connection to the other at the network layer without having to traverse the TCP layer to the application level. This requires that appropriate address translations and sequence number modifications be performed on the packets.

A URL-aware switch can be implemented that incorporates TCP splicing using *TCP gateway programs* contained in the NEPPI (Network Element for Programmable Packet Injection) API (Application Programming Interfaces) [Bell20]. The performance of TCP splicing for URL-aware redirection can be found in [CoRH99.1].

3.4.2 Transparent Switching of HTTP Traffic to Proxy Cache

Proxy caching is used to decrease both the latency of object retrieval and traffic on the Internet backbone. Web browsers generally have to be configured such that requests are channeled through a proxy.

Cache server products supporting transparent caching is actually called network cache. The cache is used for dealing with object requests in a way that is transparent to the Web browser, i.e., no proxy setting, but this needs to be located on the router that a client request takes when going from the browser to the origin server. The layer-4 switches with network cache on the market [Foun20], [Danz98], [CaFI20], [Inkt20], [CiLD20] are designed to redirect transparently such client requests to the caches. Indeed, network caches act in specific TCP mode where the packets can carry the origin server's IP address as the destination. Hence, the network cache will have the origin server's IP address and it can make a connection to the server to retrieve an object if it is not available locally. A *complete URL*, which does not have the origin server name but only the path relative to the document, is sufficient in this case.

Proxy servers (or proxy caches) handle requests differently than network caches. When the browser is configured to use a proxy cache, the browser first makes a TCP connection to the proxy cache and then sends an *absolute URL* of the requested object. The absolute URL (which includes the origin server name and the path to the required object from the document root at the server) is required because the packets that go from the browser to the proxy contain the proxy's IP address as the destination address. If only a complete URL is provided, the proxy will not know the origin server name or IP address.

When a browser is not configured to use a proxy, it retrieves objects as follows: the browser first makes a TCP connection to the origin server and then sends only the *complete* URL. A complete URL is sufficient as the connection is already established with the origin server and it is understood that the origin server serves documents by parsing URLs relative to its document root.

However, a standard or pure application-level proxy cannot expect this feature to be provided on the system that it is run on. To make standard proxies support transparent caching, some new approaches are needed to extend the complete URL to an absolute URL by processing the packet at the IP level. A good example of transparent switching HTTP to proxy cache is TCP gateway programs implemented by NEPPI API.

Using TCP gateway program functions, when the packet containing the HTTP GET request is received from the client (this is the first payload packet from the client), the gateway program processes this packet and transforms the complete URL to an absolute URL by pre-fixing the origin server IP address to the complete URL. The origin server IP address is available as the destination address on the IP packets. If the payload has been modified, the TCP checksum has to be recalculated [CoRS99.2].

To make this URL transformation transparent to the endpoints, IP and TCP header changes are required. The total length field on the IP header is an offset (*lengthof* (absolute URL) - *lengthof* (complete URL)) because the length of the IP packet that carried the GET request is now increased. In addition, the TCP header contains the sequence number (*seq*) and acknowledgement sequence numbers (*ack_seq*) that need translation. The *seq* on the TCP header indicates the byte number of the first byte in this packet going from the sender to the receiver during the TCP session, while the *ack_seq* indicates the byte number of the next byte that the sender expects to receive from the receiver. For all packets after the GET packet(s) that go from the client to the server, the *seq* needs to be increased by an offset so that the *seq* matched the byte number that the server expects to receive from the client. Similarly, on all packets starting with the acknowledgement to the GET packet that go from server to client, the *ack_seq* matches the byte that the server should expect the client to send in the next packet following the GET packet. Performing the above changes to the header makes the TCP endpoint unaware of the changes to the GET packet.

3.5. Intelligent Features of Layer 4/7 Switching

This chapter presents different intelligent switching methodologies. The layer-3 switching consists of NetFlow, CEF switching, and Tag switching. The layer-4/7 switching advanced techniques includes: transparent server clustering, server load balancing algorithms, load-sharing network address translation, URL-aware switching, and transparent switching of HTTP traffic to proxy cache.

Today's switching is becoming more and more intelligent. The layer-3 switching is mainly targeting QoS implementation over IP network, such as Tag switching based on MPLS. The layer 4/7 switching has been developed to meet the demands of the rapid growth of Internet and Intranet and usage. The Web switch, as a new breed of product, uniquely designed to meet the needs of Web data center infrastructure that is a new architecture scaling e-business growth. Using layer-4/7 switching techniques, Web switching is supposed to have the combination of server, application, content and network intelligence that is not found in the conventional layer-2/3 switches and routers. According to [Alet20], a variety of intelligent features is summarized by server-awareness, application-awareness, content-awareness, and network-awareness, as shown Table 3.2.

Table 3.2. The intelligent features in the Web switching.

Intelligence	Characteristics
Server-awareness	• Optimum load distribution across server farms • Web farm available management • Web farm performance management
Application-awareness	• Application-intelligent health checking • Web session and application state management
Content-awareness	• Supporting applications with dynamic ports • Web traffic control based on URL • Cookie cutting
Network-awareness	• Full wire speed layer-2 and layer-3 switching on every port • Standard-based layer-2 functions • Standard-based layer-3 functions • Support for network-oriented QoS functions • Support for mesh and redundant network configurations
Advanced-Web switch functions	• Multi-site Web farm coordination • Transparent traffic redirection • Non-server load balancing • Web farm bandwidth management • Security measures

Chapter 4

System Design & Architecture

Based on the technology and methodology discussed in Chapter 2 and Chapter 3, this chapter discusses some issues related to system design and architecture. Section 4.1 gives two approaches to system design, using server clustering. Section 4.2 provides our system architecture consisting of five main parts: Network monitoring, Data analysis, Switching decision, Feedback information control, and Server farms. Section 4.3 illustrates some of the system functionalities available within this designed system architecture, primarily including: SNMP monitoring, system logs monitoring, network data analysis, and feedback information control.

4.1. Approaches to System Design

The basic idea of this sort of system design is based on server farms (or server clusters) to achieve server load balancing that can actually further improve network performance. Server farms, consisting of a number of real servers, run a variety of server applications so that different Internet server services could be available on these servers (real server machines). In the following subsection, we will discuss two approaches to reaching this goal.

4.1.1 Two Design Approaches

The first design approach is the deployment of various application-layer services on different servers, effectively increasing total server capacity. For example, a Web server could be deployed on one server, a News server on another, and a File-transfer server on a third. However, this approach does nothing to increase application availability. If the news server is down, no one has access to newsgroups. Also, the approach doesn't use server capacity efficiently. The web server could be highly congested, giving users slow responses, while the file transfer server is lightly loaded, with CPU cycles to spare.

An expectedly better solution is the one that could increase the processing power of applications, provide high application availability, and use server CPU cycles more efficiently. Basically this approach basically is the running of each supported service on several servers and balance the client load across the these servers using session switches. For example, the same three servers are each able to run all the applications (e.g., News, Web, File, or others), and application sessions could be distributed automatically (i.e., load balanced) among the three servers using new session switching technology. Fig. 4.1 shows these two design approaches [GuMR20].

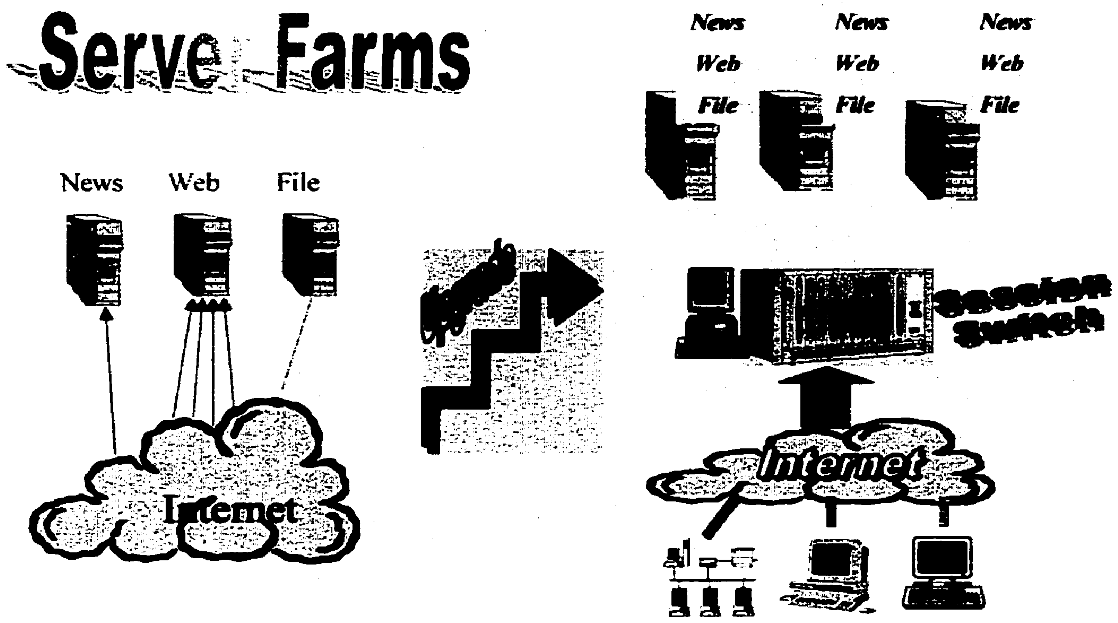

Fig. 4.1. Intelligent switching using server farms.

4.1.2 Advantages of Upgrade Version

The upgrade version of system infrastructure is designed to obtain the server load balancing within server farms. Server load balancing, the process of distributing service requests across a group of servers (or server clusters), could address several requirements that are becoming increasingly important in networks, i.e., scalability, high performance, high availability, and disaster recovery.

Firstly, server load balancing could INCREASE THE SCALABILITY OF NETWORKS. Indeed, server load balancing makes multiple servers appear as a single server, namely a single virtual service, by transparently distributing user requests among these real servers. The end-user requests are sent to a load-balancing device (also called session switch) that determines which server is most capable of processing the request to that server. Server load balancing can also distribute workloads to firewalls and redirect requests to proxy servers and caching servers.

Secondly, HIGHER PERFORMANCE IS ACHIEVED when the processing power of servers is used intelligently. Advanced server load-balancing products can direct end-user service requests to the servers that are least busy and therefore capable of providing the fastest response time. Necessarily, the load-balancing device should be able to handle the aggregate traffic of multiple servers. If a server load-balancing device becomes a bottleneck, it is no longer a solution; it is just an additional problem. In this case, some other implementation issues should be considered, such as adding a backup load-balancing device.

The third benefit of server load balancing is its ability to IMPROVE APPLICATION AVAILABILITY. If an application or a server fails, load balancing can automatically redistribute end-user service requests to other servers within a server farm or to servers in another location. Server load balancing also prevents planned outages for software or hardware maintenance from disrupting service to end-users.

Lastly, distributed server load-balancing products can also provide DISASTER RECOVERY SERVICES by redirecting service requests to a backup location when a catastrophic failure disables the primary site.

4.2. System Architecture

4.2.1 Diagram of System Architecture

Figure 4.2 [GuRM20], [GuMR20], [GuRW99] illustrates a system architecture of intelligent switching design. There are five main modules: Networking monitoring, Data analysis, Switching decision, Feedback information control, and Server farms.

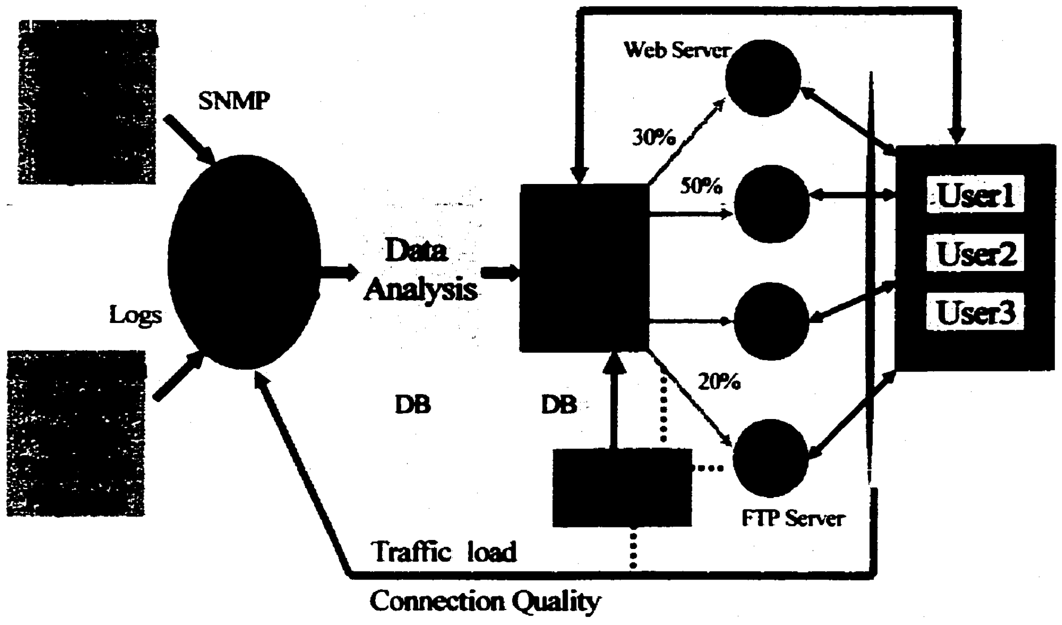

Fig. 4.2. System architecture of intelligent switching design.

In Figure 4.2, Module Network Monitoring is based on the models of both SNMP and system logs. It is also related to the traffic load and connection quality between Server Farms (S1~S4) and Users. Module Data Analysis, with a database, receives the traffic data from Module Network Management and sends its results to Module Switch Decision. Module Switch Decision, with a database, receives inputs from Module Data Analysis, Module Feedback Information Control, and Users' connections or requests, and its output goes to Module Server Farms (S1~S4). Module Feedback Information Control works with some link-elements such as the link of connection quality and traffic load, the link between Switch Decision and Server Farms, and the servers on Server Farms, then it outputs the information to Module Switch Decision.

4.2.2 System Description

NETWORK MONITORING: The primary task of this part is to collect data and provide some useful information for data analysis and switching decision. It includes SNMP monitoring, system logs monitoring, and the monitoring of connection quality and traffic load. SNMP model is used to monitor LAN local devices, collecting SNMP MIB data. The purpose of system logs monitoring is network administration, system security and resource utilization. Moreover, the connection quality and traffic load are being monitored for feedback information, and this process could be done through counting connections, recording numbers, calculating the packet payload of clients requests, and analyzing the distributed bandwidth of server, etc. The combination of these network monitoring will be beneficial for dynamic traffic management.

DATA ANALYSIS: This takes advantage of the network monitoring data collection that is stored in a database. Data analysis here primarily focuses on network performance analysis, i.e., availability, response time, accuracy, throughput, and utilization. Data analysis can be processed by using some existing measurement tools [CAID99]. For example, *tcptrace* (Unix, free) is a tool for *tcp* session analysis, and it is a free Unix-based resource; similarly, *Keynote* (Java, $$) a tool for Web service monitoring (server response time), and *IPTraf* (Java, Linux 2.0 and higher, free) is a tool for IP LAN collection monitoring, and so on. The R & D (research and development) work in this area has already been done more, as compared with the area of intelligent switching technology. We use the part of data analysis here for providing sufficient information for Switching Decision to make intelligent switching decisions of traffic management.

SWITCHING DECISION: This is a core part of this system architecture. Different switching techniques and approaches of the layer-4/7 switching that are discussed in Chapter 3 can be used here, depending on network system requirements. The intelligent switching decision to select the best server for the next connection could basically be made by using server load balancing algorithms. For dynamic traffic management, it is necessary to utilize network monitoring, data analysis, and feedback information. We will make this point more clearly later in both this chapter and other chapters.

FEEDBACK INFORMATION CONTROL: This is a distinguishing feature of this system architecture. The literature review tells us that so far, most of layer-2~7 switching products have not implemented this kind of mechanism on feedback information control directly. Therefore, it is a new direction and it is also a challenge for networking management. There are two methods that originate in this paper to obtain the feedback information: it could be done by using QoS approaches, or by using Cisco Dynamic Feedback Protocol (CDFP). More details will be presented in section

4.3.4. Furthermore we are also expecting that other new kinds of active feedback protocols could show up to contribute to this new area.

SERVER FARMS: This part includes several servers that actually provide a variety of services in each. For instance, a Web server is installed for dealing with *http* traffic, a FTP server for *ftp* file-transfer requests, a Mail server for e-mails service, a News server for newsgroup application, a NFS (Network File System) server for network file system service, and so on. Each installed server machine is capable of providing all of these services. Users' service requests sending directly to Switching Decision could be transparently redirected to different servers, thus the entire service bandwidth available to users is actually provided by several real servers on Server Farms with different percent of total bandwidth amount.

POLICY-BASED MANAGEMENT could be used here for this sort of traffic redirection. That is, Switch Decision is operated by a Policy Manager that produces and implements different policies, such as QoS policy, Security policy, VPN policy, etc. Now let's take a look at a typical example.

Assuming Switch Decision receives a client request for getting a large file, e.g., 200Mb in size (this is maybe a *http* or *ftp* by URL, or a telnet session). One TCP session opens, and we next need to manage some TCP traffic streams. We now setup and assign some policies for obtaining network security and server load balancing:

Policy-1: Security policy is used on server-3 (S3 in Fig. 4.2) because S3 is used mainly for network administration and/or providing sufficient resources for internal research projects; this means we don't want distribute any external traffic to this server.

Policy-2: QoS policy is adopted here for server load balancing. Server-1 (S1) is supposed to primarily serve as a Web server and handle most of the *http* traffic; unfortunately, at this time the network monitoring information shows that S1 is already carrying 80% of its capacity (or full load). Thus, Switching Decision gives S1 only 30% TCP traffic payload. On the other hand, the network monitoring shows server-2 (S2) is fairly free at this moment, only having 10% of its capacity. In this case, 50% of the TCP traffic payload is allocated to S2 by Switch Decision. Finally, the remaining 20% of the TCP traffic payload is redirected to server-4 (S4) because this server is supposed to deal with most of the *ftp* traffic, but currently it has an average traffic load (neither higher load nor lower load). Therefore, one TCP session of the request traffic is appropriately distributed to the different servers, and reasonable server load balancing is obtained. Furthermore, it is also obvious that TCP traffic aggregation could achieve better server load balancing through using this sort of dynamic policy-based management approach.

4.3. System Functionality

This section discusses some of the available functionalities for the designed system architecture, including SNMP monitoring, sysLogs monitoring, data analysis, and feedback information control.

4.3.1 SNMP Monitoring

In SNMP network management, each source to be managed is represented by an object and the Management Information Base (MIB) that is a structured collection of such objects. The set of defined objects has a tree structure with the root of the tree being the object referring to the ASN.1 (Abstract Syntax Notation) standard. The MIB structure [Appendix B] describes the definition and organization of this sort of management information base, by which we can access and collect monitoring data.

We use the basic SNMP model here to monitor network devices and collect traffic data based on MIB-II (defined in RFC 1213) [RFCs]. Because MIB is a large data Base of management information, only certain Object Groups of interest related to this system architecture are selected for discussion.

Referring to the subtree of mib-2(1) shown in Fig. B.1 [Appendix B], the object mib-2 with Object Identifier (OID: 1.3.6.1.2.1) is subdivided into the following groups: *system, interfaces, at, ip, icmp, tcp, udp, egp, transmission,* and *snmp*. The basic functions on these groups are presented below.

- SYSTEM GROUP *(system 1.3.6.1.2.1.1)*: contains general information about the managed system.

- INTERFACES GROUP *(interface 1.3.6.1.2.1.2)*: contains generic information about the physical interfaces of the entity, including configuration information and statistics on the events occurring at each interface that is treated as being attached to a subnetwork. The implementation of this group is mandatory for all systems.
- ADDRESS-TRANSLATION GROUP *(at 1.3.6.1.2.1.3)*: contains a single address-translation table for internet-to-subnet address mapping. Each row in this table provides a mapping from a network address (that is typically the IP address for this system at this interface) to a physical address (that depends on the nature of the subnetwork). For example, if the interface is to a local area network, then the physical address is the MAC (Medium Access Control) address for that interface; if the subnetwork is an X.25 packet-switching network, then the physical address may be an X.121 address.
- IP GROUP *(ip 1.3.6.1.2.1.4)*: contains the information relevant to the implementation and operation of IP (Internet Protocol) at a node (or device, system). There are three tables included: *ipAddrTable*, *ipRouteTable*, and *ipNetToMediaTable*, providing IP packets and routing information on the network layer. Since IP is implemented in both end systems (hosts) and intermediate systems (routers), not all of the objects in this group are relevant for any given system. Objects that are not relevant have null values.
- ICMP GROUP *(icmp 1.3.6.1.2.1.5)*: contains the information relevant to the implementation and operation of ICMP at a node. This group provides the various types of ICMP messages sent and received.
- TCP GROUP *(tcp 1.3.6.1.2.1.6)*: contains information relevant to the implementation and operation of TCP at a node. There are some objects and only one table (*tcpConnTable)* in this group, providing TCP session information on the transportation layer.
- UDP GROUP (*udp 1.3.6.1.2.1.7)*: contains the information relevant to the implementation and operation of UDP at a node. This group provides the information about datagrams sent and received (such as *udpInDatagrams*, *udpNoPorts*, *udpInErrors*, and *udpOutDatagrams*), and one table (*udpTable)* that includes *udpEntry* with groups *udpLocalAddress* and *udpLocalPort*.
- EGP GROUP (*egp 1.3.6.1.2.1.8)*: It contains the information relevant to the implementation and operation of EGP at a node. This group provides the information about EGP (External Gateway Protocol) messages sent and received, (such as *egpInMsgs*, *egpInErrors*, *egpOutMsgs*, and *egpOutErrors*) and one table (*egpNeighTable)* containing the information about each of

the neighbor gateways known to this entity (i.e., *egpNeighEntry*). The table is indexed by *egpNeighAddr*, which is the IP address of a neighbor gateway.

- TRANSMISSION GROUP (*transmission 1.3.6.1.2.1.10)*: contains the information about transmission schemes and access protocols at each system interface. Currently, no such objects are defined for MIB-II.
- SNMP GROUP (*snmp 1.3.6.1.2.1.11)*: contains the information relevant to the implementation and execution experience of Snmp (simple network management protocol) on this system.

Using the information from these Object groups, we can obtain some monitoring messages. For example, if the monitoring data shows *sysServices (1.3.6.1.2.1.1)* in SYSTEM group is equal to 7 (its syntax is INTEGER), that means the service provided by this node (or device, or system) primarily is Applications (layer-7 services). The reason is that the value of *sysServices* indicates the set of services that this entity mainly offers, i.e., Layer-1 (value=1) stands for Physical (e.g., repeaters), Layer-2 (value=2) for Data-link/subnetwork (e.g., bridges), Layer-3 (value=3) for Internet (e.g., IP routers), Layer-4 (value=4) for End-to-end (e.g., IP hosts), and Layer-7 (value=7) for Applications (e.g., mail relays).

Similarly, we can take full advantages of other data collection monitored from SNMP MIB-II. For instance, within the INTERFACE group, *ifspeed (1.3.6.1.2.1.2.5)* shows an estimate value of the interface's current data-rate capacity, *ifOutOctets (1.3.6.1.2.1.2.16)* indicates the total number of octets transmitted on the interface (outbound traffic), *ifInDiscards (1.3.6.1.2.1.2.13)* indicates the number of inbound packets discarded (e.g., buffer overflow); within the AT group, *ifOutQlen* indicates the length of the output packet queue; within the group TCP, *tcpMaxConn (1.3.6.1.2.1.6.4)* indicates the limit on the number of local TCP connections the entity can support, *tcpActiveOpens (1.3.6.1.2.1.6.5)* is the number of active opens that have been supported by this entity, *tcpAttemptFails (1.3.6.1.2.1.6.7)* is the number of failed connection attempts that have occurred at this entity, *tcpRtoMin (1.3.6.1.2.1.6.2)* is the minimum value for the retransmission, *tcpCurrEstab (1.3.6.1.2.1.6.9)* is the number of TCP connections for the current state is either Established or Closed-wait, *tcpInSegs (1.3.6.1.2.1.6.10)* is the total number of segments received including those containing only retransmitted octets, *tcpRetransSeg (1.3.6.1.2.1.6.12)* is the total number of retransmitted segments, and *tcpConnTable (1.3.6.1.2.1.6.1.13.~)* contains TCP connection-specific information such as *tcpConnState (1.3.6.1.2.1.6.13.1.1)*, *tcpConnLocalPort (1.3.6.1.2.1.6.13.1.3)*, and so on.

In short, it is not possible to describe all of groups here in this thesis, and it is also not necessary to collect all of the MIB data; it actually depends on monitoring functions of required systems. Table C.1 [Appendix C] lists some parts of data collection based on MIB-II, which are actually used for dynamically monitoring in this architecture.

There are many kinds of software development tools for NMS (network management systems) monitoring. Based on prior research results [LiJi99] [GuMF20] [GuRW99] at University of Manitoba, a Web-based 3-tier Client/Server architecture using SNMP/CORBA/Database approach is strongly recommended.

Our research results show that the SNMP model is a good way to implement network monitoring. However, there are several limitations of SNMP protocol. It may not be suitable for the management of truly large networks, it is not well suited for retrieving large volumes of data (such as entire routing table), its traps are unacknowledged, it does not support manager-to-manager communications, and the SNMP MIB model is limited because it does not readily support applications that make sophisticated management queries based on object values or types, and so on. Thus, SNMP management is not perfect. In addition, based on the nature of this research project, we need more features of other approaches to network monitoring, such as sysLogs monitoring.

4.3.2 System Logs Monitoring

Auditing is the process of monitoring the behavior of a system on which protection mechanisms have been established. Both UNIX and Linux maintain a number of log files that keep track of what has been happening to the computer. Early versions of Unix used the log files to record who logged in, who logged out, and why they did so. Newer versions of Unix provide expanded logging facilities that record such information as what files are transferred over the network, attempts by users to become the superuser, electronic mail, and much more [WeKa96] [GrSp96].

To take advantage of this sort of system logs monitoring, we present three ways to do so: basic Log files, sysLog facilities, and creating our own sysLog

- BASIC LOG FILES

LOG FILES are an important part of building a secure system because they form a recorded history (or *audit trail)* of a computer's past in order to make it easier to track down intermittent problems or attacks. Log files could be used for discovering the cause of a bug, the source of a break-in, and the scope of the damage involved, which may help us to rebuild our systems safely.

Most log files are text files that are written line by line by system programs. Different versions of UNIX store log files in different directories. The most common locations are: */usr/adm* (early version of Unix), */var/adm* (newer versions of Unix), and */var/log* (some versions of Solaris, Linux, BSD, and free BSD). Within these directories, some or all of these files are found: *acct* or *pacct, auclog, lastlog, loginlog, messages, sulog, utmpl, utmpx, wtmp2, wtmpx, vold.log,* and *xferlog.* Table D.1 [Appendix D] lists the functions of these Log files.

lastlog File: This file is used to record the last time that each user logged into the system. It is designed to provide quick access to the last time that a person logged into the system, but it doesn't provide a detailed history recording the use of each account. A script program needs to be developed to read the contents of this file; an example in Perl script is given in Appendix D.

utmp and wtmp Files: The file */etc/utmp* is used to keep track of who is currently logged into the system. The file */var/adm/wtmp* is used to keep track of both logins and logouts. The extended *wtmpx* is used by Solaris, including such information as Username, inittab id, Terminal name, Device name, Process ID of the login shell, Code that denotes the type of entry, Exit status of the process, Time that the entry was made, Session ID, and Unused bytes for future expansions. By scanning these files, Unix programs (such as *who, whodo, users, finger, ps, write, last,* etc.) can report the information to users.
Logging Network Services: Some versions of the *inetd* Internet services daemon have options that can be used for logging incoming network services. To enable *inetd* logging, we locate the startup file from which *inetd* is launched and added the "*-t*" (trace) option. For example, for logging of

incoming TCP connections, we need to launch: */usr/sbin/inetd -t -s* . (s option means "standalone").
There are also many other possible log files on Unix systems that result from third-party software such as Usenet news programs, gopher servers, database applications, and others. These programs often generate log files both to show usage and to indicate potential problems.

- UNIX SYSTEM LOG (*SYSLOG*) FACILITY

In addition to the various logging facilities mentioned above, many versions of Unix provide a general-purpose logging facility called *syslog*, originally developed at the University of California at Berkeley for the Berkeley sendmail program. Since then, *syslog* has been ported to several System V-based systems, and is now widely available.

The *syslog* is a host-configurable, uniform system logging facility. The system uses a centralized system logging process that runs the program */etc/syslogd* (or */etc/syslog*), and individual programs that need to have information logged send the information to *syslog*. Messages can then be logged to various files, devices, or computers, depending on the sender of the message and its severity.

Any program can generate a ***syslog message***. Each message consists of four parts: ***program name***, *facility*, ***priority***, and ***log message*** itself. Tables D.2 [Appendix D] and D.3 [Appendix D] summarize the *syslog* facilities and *syslog* priorities, respectively, which are potentially available on Unix or Linux, although not all facilities are presented on all versions of Unix or Linux.

When *syslogd* starts up, it reads its configuration file, usually */etc/syslog.conf* [Appendix D], to determine what kinds of events it should log and where they should be logged. The *syslogd* then listens for log messages from the following three sources: */dev/klog*, */dev/log*, and *UDP port 514*, which are listed in Table D.4 [Appendix D]. We can then get and analyze log messages.

For example, there is a log message: (login: Root LOGIN REFUSED on ttya). It means this log message is generated by the *login* program, indicating that somebody tried to login to an unsecure terminal as *root*. The message's facility (*authorization*) and priority of error level (*critical error*) are not shown. Tables D.5 and D.6 [Appendix D] list some typical critical messages of *syslog* and typical info messages of *syslog*, respectively.

- CREATE OUR OWN *SYSLOG*

We may want to insert *syslog* calls into our own programs to record information of importance. As well some third-party software often has the capability to send log messages into the *syslog* if configured corrected. For instance, Xyplex terminal servers and Cisco routers both can log information to a network *syslog* daemon; Usenet news and POP mail servers also log information.

If writing shell scripts, we can also log to *syslog*. Usually, systems with *syslog* come with the *logger* command. For example, if we include the following: logger -t ThisProg -p user.notice "Called without required # of parameters", we can log a warning message about a user trying to execute shell file with invalid parameters.

There is a log file tool named *Swatch*, which is a simple program written in the Perl programming language that is designed to monitor log files. It allows us to automatically scan log files for particular entries and then take appropriate action, such as sending a mail, printing a message on screen, or running a program. *Swatch* was developed by E. Todd Atkins at Stanford's EE Computer Facility. It is not currently included as standard software with any Unix distribution, but it is available via [Swatch].

4.3.3 Data Analysis

It is absolutely prerequisite and definitely important to measure a network's performance. We cannot expect to manage and control a system or an activity well unless we can monitor its performance properly. There are some typical measurement indicators of network performance that can be classified into the following two categories: service-oriented measures including *Availability*, *Response time*, *Accuracy*, and efficiency-oriented measures including *Throughput* and *Utilization* [Stal93].

4.3.3.1 Availability

Availability can be expressed as the percentage of time that a network system, a component, or an application is available for a user. Depending on the application, the value of high availability can be significant.

Availability is based on the reliability of the individual components of a network. The reliability is the probability that a component will perform its specified function for a specified time under certain conditions, and component failure is usually represented by the *mean time between failures* (MTBF), so the *availability* (A) can be expressed as: A = MTBF/(MTBF+MTTR), where MTTR is the *mean time to repair* following a failure.

The availability of a system depends on the availability of its individual components plus the system organization. For example, for some redundancy components, the failure of just one component does not affect system operation; depending on system configurations, the loss of a component may result in reduced capability, yet the system still functions.

4.3.3.2 Response Time

The response time generally means the time it takes for the system to respond to a particular task. For different types of applications, a slightly different definition is needed for each specific purpose. For example, in an interactive transaction, response time may be defined as the time of the last keystroke by the user and the beginning of the display of a result by the computer; in packet-switched networks, the end-to-end delay may possibly be defined as the total time that packets are transmitted over all communication links from source to destination.

To measure response time, we need to examine a number of elements. Fig 4.3 describes a typical networking situation, indicating the seven elements of the response time that are common to most interactive applications. Each of these elements is one step in the overall path that an inquiry takes through a communication configuration. These elements are: ITDelay = inbound terminal delay, IQTime = inbound queuing time, ISTime = inbound service time, CPU = CPU and 2Front-end Processor delays (processing time), OQTime = outbound queuing time, OSTime = outbound service time, and OTDelay = outbound terminal delay.

Each element contributes a portion of the overall response time, so we get the overall response time (ORT) expressed as:

$$ORT = ITDelay + IQTime + ISTime + CPU + OQTime + OSTime + OTDelay$$

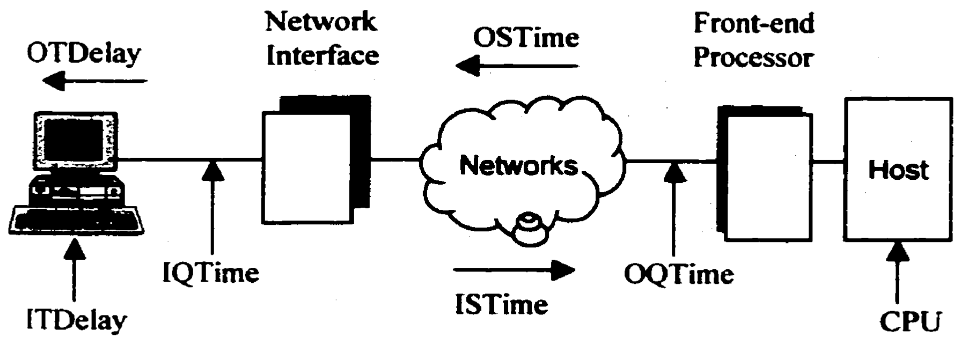

Fig. 4.3. Network system response time analysis.

Inbound terminal delay (ITDelay) and *Outbound terminal delay (QTDelay): ITDelay* is the delay in getting an inquiry from the terminal to the communications line. The terminal itself has no noticeable delay, so the delay is directly dependent on the transmission rate from terminal to controller. Similarly, *QTDelay* is the delay at the terminal itself; this is primarily due to line speed. As a router, this is line termination delay of the input/output port processing. It could be expressed as *transmission delay*.

Inbound queuing time (IQTime): is the time required for processing by the controller that deals with inputs from a number of terminals as well as inputs from the network to be delivered to the terminals. Thus, an arriving message received on the input port will be placed in a buffer to be served in turn. The busier the controller is, the longer the delay for processing will be. In a router architecture, this actually includes two portions of delay during the input port processing, i.e., data link processing (protocol, decapsulation) and queuing processing (lookup, forwarding). Hence, this time contains the processing delay and queuing delay.

Outbound queuing time (OQTime): is the time a reply spends at a output port in the front-end processor waiting to be dispatched to the network or communications line. As with the controller, the front-end processor will have a queue of replies to be serviced, and the delay increases as the number of replies waiting increases. In a router architecture, it actually consists of two portions delay during the output port processing including queuing processing (buffer management), data link processing (protocol, encapsulation), and queuing processing (lookup, forwarding). Hence, this time contains processing delay and queuing delay.

Processing time (CPU): is the time the front-end processor, the host processor, the disk driver, etc., in the computer spend preparing a reply to an inquiry. As a router, this is similar to the routing processor. This element is usually outside the control of the network manager.

Inbound service time (ISTime) and *Outbound service time (OSTime): ISTime* is the time to transmit the communications link, network, or other communications facility from the controller to the host's front-end processor. Similarly, *OSTime* is the time to transmit the communication facility from the host's front-end processor to the controller. This element is itself made up of a number of elements, based on the structure of the communications facility. If the facility is a public-switched network, it must be treated as a single element. However, if it is a private network (wide area or local area), leased line, or other user-configured facility, then a breakdown of this element will be needed for network control and planning.

In particular, although it may be possible to directly measure the total response time in a given network environment, Fig. 4.3 alone is of little use in correcting problems or planning for the growth of the network. For these purposes, a detailed breakdown of response time is needed to identify bottlenecks and potential bottlenecks.

Figure 4.4 illustrates an example of the end-to-end delay analysis through switching networks.

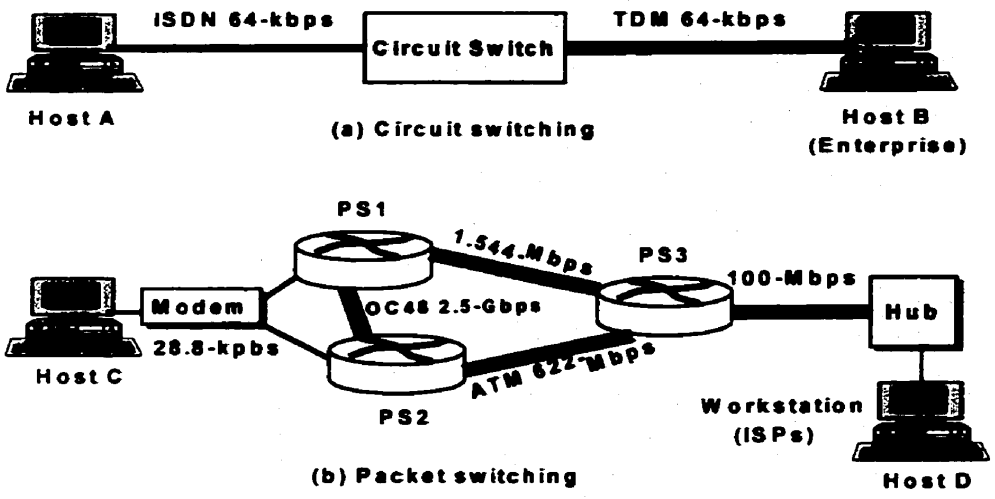

Fig. 4.4. The end-to-end delay analysis through switching networks

Fig 4.4(a) shows a circuit switching system. Host A (as a client) accesses the Circuit Switch through a digital communication link (e.g., ISDN 64Kbps) (Integrated Service Digital Network) and another link of Circuit Switch is implemented by TDM (e.g., 64Kbps) (Time-Division Multiplexing), connected by an Enterprise (e.g., Host B).

Fig 4.4(b) shows a packet switching system. There are three packet switchings (e.g., PS1, PS2, and PS3) with three different links. A T1 link (e.g.,1.554Mbps), an optical OC 48 link (e.g., 2.5Gbps), and a (IP over) ATM link (e.g., 622Mbps) are between PS1 and PS3, PS1 and PS2, and PS2 and PS3, respectively. At the user-end, Host C is connected by a modem to access the packet-switched network through an analog link (e.g., 28.8Kbps). The modem is available to connect to PS1 and PS2. At the ISPs (Internet Service Providers) end, Host D (on a workstation with a Hub) is connected to PS3 through a 100 based Ethernet (e.g., 100Mbps).

Processing delay: is the time required for the packet switching (e.g., a router) to examine the packet's header and determine where to direct the packet; it also includes other factors such as the time needed to check for bit-level errors in the packet that occurred in transmitting the packet's bits. It is simply expressed here as D_{proc}.

Queuing delay: is the time the packet spends at the queue as it waits to be transmitted across the link. The queuing delay of a specific packet will depend on the number of other earlier-arriving packets that are queued and waiting for transmission across the link, and it can vary significantly from packet to packet. It is expressed here by D_{queue}.

Transmission delay: is the amount of time required to transmit all of the packet's bits across the link. We denote the length (or size) of the packet by *L* bits and the transmission rate of the link by *R* bits per second. For example, for a 1.554 T1 link, the rate is *R*=1.544 Mbps. The transmission delay is expressed by $D_{trans} = L/R$ (in seconds).

Propagation delay: is the time required for the bit to propagate from the beginning of the link to the end. The bit propagates at the propagation speed of the link, and the propagation speed depends on the physical medium of the link (i.e., multimode fiber, twisted-pair copper wire, etc.), and is in the range of $2*10^8$ to $3*10^8$ m/s. We denote the distance of the link by *d*, and the propagation speed of the link by *s*, so the propagation delay is expressed by $D_{prop} = d/s$ (in seconds).

Therefore, we get the total delay for packet switching networks, i.e., $T_{total} = D_{proc} + D_{queue} + D_{trans} + D_{prop}$.

Although we include all these delay components here, not all of them contributes equally. Some factors might be negligible, depending on the network.

Ideally, the response time for any application is expected to be short; however, shorter response time imposes greater cost. This cost comes from two sources: computer processing power and competing requirements. Firstly, the faster the computer is, the shorter the response time will be run on, and increased processing power means increased cost. Secondly, providing rapid response time for some processes may penalize other processes.

Response time is relatively easy to measure and is one of most important classes of information needed for network management. We also can use simulation tools, such as OPNET to analyze a variety of delay elements.

4.3.3.3 Accuracy, Throughput and Utilization

Accuracy is expressed as the percentage of time that no errors occur in the transmission and delivery of information. The accurate transmission of data between user and host or between two host is essential for any network. Indeed, some protocols, such as the data-link and transport protocols include certain error-correction mechanisms, so a user generally does not need to be concerned about accuracy.

Throughput is expressed as the rate at which application events (e.g., transactions, messages, file transfers) occur. This is an application-oriented measure. These measures include, for example, the number of transactions of a given type during a certain period of time, the number of customer sessions for a given application during a certain period of time, and the number of calls for a circuit-switched environment. It is useful to track these measures over time to get to know likely performance trouble spots.

Utilization is expressed as the percentage of the theoretical capacity of a resource (e.g., multiplexer, transmission line, and switch) that is being used. It refers to determining the percentage of time that a resource is in use over a given period of time. Perhaps the most important use of utilization is the search for potential bottlenecks and areas of congestion, because the response time usually increases exponentially as the utilization of a resource increases. This is a well-known result of queuing theory. For this exponential behavior, congestion can quickly get out of hand if it is not spotted early and dealt with quickly.

Basically there are two kinds of invaluable tools for performance management: Queuing Theory concepts and Statistical Analysis concepts.

4.3.4 Feedback Information Control

Once we expect to make networks being smarter, it is a good time to consider adding control mechanisms of feedback information. Two methods are discussed in this section: Connection Quality and Cisco Dynamic Feedback Protocol.

4.3.4.1 Connection Quality

The quality of a network connection is described in terms of availability, latency, jitter, and capacity. The availability is the assurance that traffic will reach its destination successfully, and forms the basis of most service-level agreements (SLAs) today. The latency is the delay that traffic experiences as it travels across the network. The jitter is the change in this latency over time. The capacity is the total amount of bandwidth available on a link.

It is a complex process to establish a particular quality of service for a connection, in part because of the stateless and best-effort model upon which the Internet is based. Taking into account QoS strategies, there are two main approaches to doing so:

The integrated services model, or *IntServ*, negotiates a particular QoS at the time it is requested. Before exchanging traffic, the sender and receiver request a particular QoS level from the network. Upon acceptance, the intermediate network devices associate the resulting traffic flow with a specific level of jitter, latency, and capacity. RSVP is an example of such a model.

The differentiated services model, or *DiffServ*, takes a different approach. A few coarse classes of traffic handling, similar to gold, silver, or bronze levels of frequent flyer cards, are established by the network administrator. When the sender needs a particular kind of handling, it marks the individual packets accordingly.

Both methods have their benefits and drawbacks, but used properly they can deliver a specific QoS level for different business applications. For example, the large number of distinct flow states that *IntServ* creates in the core of the network require some kind of flow aggregation, possibly using MPLS. Similarly, *DiffServ* systems require policying to ensure that greedy users don't monopolize available bandwidth at the expense of business-critical traffic.

Along the network path, each node enforces the level of service to which it has agreed by a number of forwarding algorithms such as queues, packet discards, TCP rate control, leaky-buckets, and so on. Coordinating all of these mechanisms across multiple devices is the job of a policy system.

4.3.4.2 Cisco Dynamic Feedback Protocol

Currently most of the IP load-balancing products' approach to distribute traffic is based on the IP address of the destination server. Cisco Dynamic Feedback Protocol (DFP) is the mechanism by which servers provide feedback to IP load-balancing products.

DFP is implemented with workload agents that reside on IP server platforms. Workload agents communicate with the load-balancing manager, which is resident on the Cisco LocalDirector [CiLD20] chassis or Catalyst® 6000 series switch [Cisc20].

The system flow of DFP works as follows: To begin the system flow, a TCP connection is initiated from the load-balancing manager to the DFP workload agent on the server. After this TCP connection is established, the workload agent must periodically send update information to maintain the connection. This status information is used by the load-balancing manager to aid in load balancing the real servers, as well as acting as a keep alive for the DFP connection. If an agent has no information to send and an inactivity timeout has been specified, the DFP workload agent must send an empty report to prevent the connection from being torn down. The defined messages and vectors [Appendix E] need to operate appropriately.

DFP is the first protocol that allows servers to provide feedback and input into the IP load-balancing decision through the use of workload agents. The server workload agent can help to instruct the load balancer that the server is congested, to tell the load balancer that the server is underutilized, and to inform the load balancer that this server should not be used for load balancing for a period of time. The application workload agent can help to instruct the load balancer that this application should take precedence over generic applications. In shortly, DFP allows servers and applications to be more scalable and highly available.

In summary, in this chapter, we provide one kind of system architecture for intelligent switching, including some details of system functions: network monitoring, data analysis, switching decision, and feedback information control. The designed system with dynamic management has actually combined a variety of new techniques, so it could be a good reference model for the next generation of intelligent switching designs.

Chapter 5

System Implementation

This chapter talks about some issues pertaining to system implementation. Section 5.1 discusses the Linux implementation approach of virtual load server clusters (VLSC), including the architecture of Linux VLSC and three VLSC strategies over IP networks. Section 5.2 presents an application example of system design, namely SmartSwitch Router, where a kind of software design prototype and its programming implementation are provided.

5.1. Linux Implementation Approach

Generally speaking, implementation platforms of a prototype for system design include a variety of elements, such as operating systems (OS), programming languages (e.g., Java, C++, C, Perl), network infrastructures (e.g., edge/backbone, IP/Optical/wireless, hub/tunnel, etc.), and other resource availability (e.g., hardware and financial investment).

Currently, popular operating systems include: Unix (Solaris, BSD, System V, etc.), Linux (Slashware, Redhat, etc.), Windows (95/98/2000/NT), DOS, and RTOS (VxWork, QNX, etc.). Among all of them, Linux is a free resource having most of the features of Unix, and more and more software products are tending to implement on Linux Boxes. Let's take a look at the Linux implementation of Virtual Load Server Clusters (VLSC).

5.1.1 Architecture of Linux VLSC

Figure 5.1 illustrates a three-tier architecture of virtual server clusters implemented on a Linux system, primarily including the following three parts: load balancer, server pool (also called server farm), and backend storage.

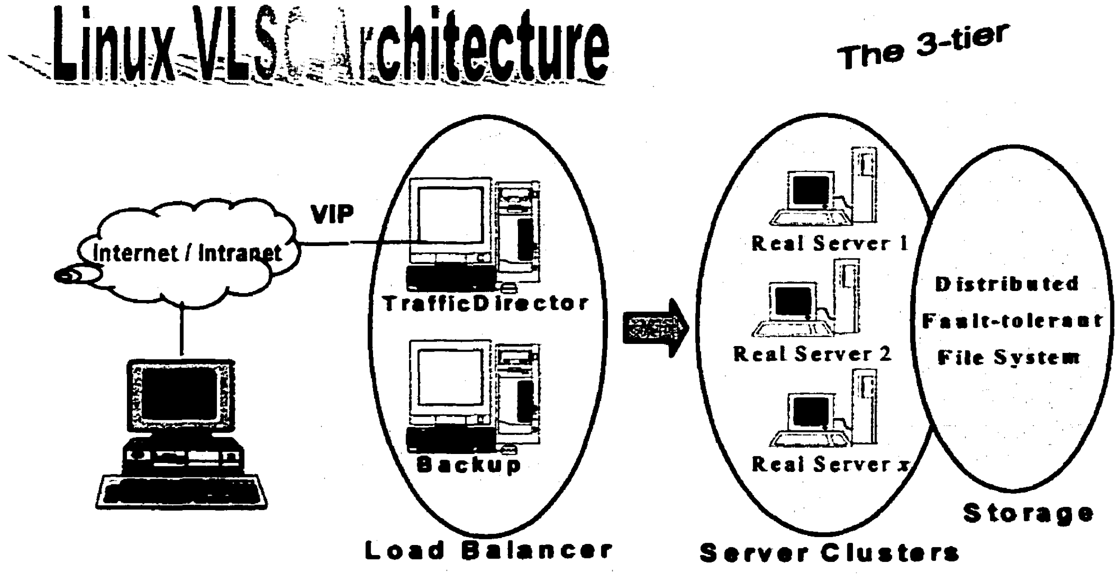

Fig. 5.1. The three-tier architecture of Linux virtual load server clusters (VLSC)

A load balancer acts as a front-end that can be seen by the outside world, i.e., it publishes a variety of services through a single IP address. We refer to this single IP address the virtual IP address, or simply VIP. The load balancer device accepts requests from clients who know this single IP address, and directs network connections to a set of servers that actually perform the work of real services. When accessing services, clients never know the real servers' IP addresses; they just believe that all of the services are available on this VIP address, so they are blindly being "fooled" by the VIP and "virtual" services. This technique is actually a sort of transparency.

A machine *TrafficDirector* handlers incoming connections using IP load balancing techniques, selects servers from the server pool, maintains the state of concurrent connections, and forwards packets. A machine *Backup* is used to prevent failure. The load balancer may become a single failure point of the whole system. In order to prevent the failure of the load balancer, we need to setup a backup of the load balancer.

The *Server pool* consists of a cluster of servers that actually implement real services, such as Web, FTP, Mail, DNS, and so on. The server nodes in the above architecture may be replicated for either scalability or high availability. Scalability is achieved by transparently adding or removing a node in the cluster. When the load to the system saturates the existing capacity, server nodes can be added to handle the increasing workload.

Backend storage provides shared storage for the servers, so that it is easy for servers to keep the same content and provide the same services. It is usually provided by distributed fault-tolerant file systems which also take care of availability and scalability of file system accesses, so that server nodes access the distributed file system like a local file system.

5.1.2 Linux VLSC Strategies over IP Networks

There are different ways to implement Linux VLSC. Fig. 5.2 shows three kinds of strategies over IP networks, i.e., VLSC/NAT, VLSC/TUN and VLSC/DR.

5.1.2.1 Linux Virtual Server via NAT (VLSC/NAT)

In this approach, the load balancer and real servers are interconnected by a switch or a hub. The workflow of VLSC/NAT is as follows. When a user accesses a virtual service (using VIP) provided by the server cluster, a request packet destined for the VIP address arrives at the load balancer. The load balancer examines the packet's destination address and port number; if they are matched for a virtual service according to the virtual server rule table, a real server is selected from the cluster by a scheduling algorithm, and the connection is added into the hash table which records connections. Next, the destination address and port of the packet are rewritten to those of the selected server, and the packet is forwarded to the server.

The load balancer handles both inbound traffic and outbound traffic. When an incoming packet belongs to an established connection, the connection can be found in the hash table and the packet will be rewritten and forwarded to the right server; when response packets come back from servers, the load balancer rewrites the source address and port of the packets to those of the virtual service (VIP address), and sends the packets back to clients. When a connection terminates or times out, the connection record is removed in the hash table. This approach requires the processing of network address translation (NAT).

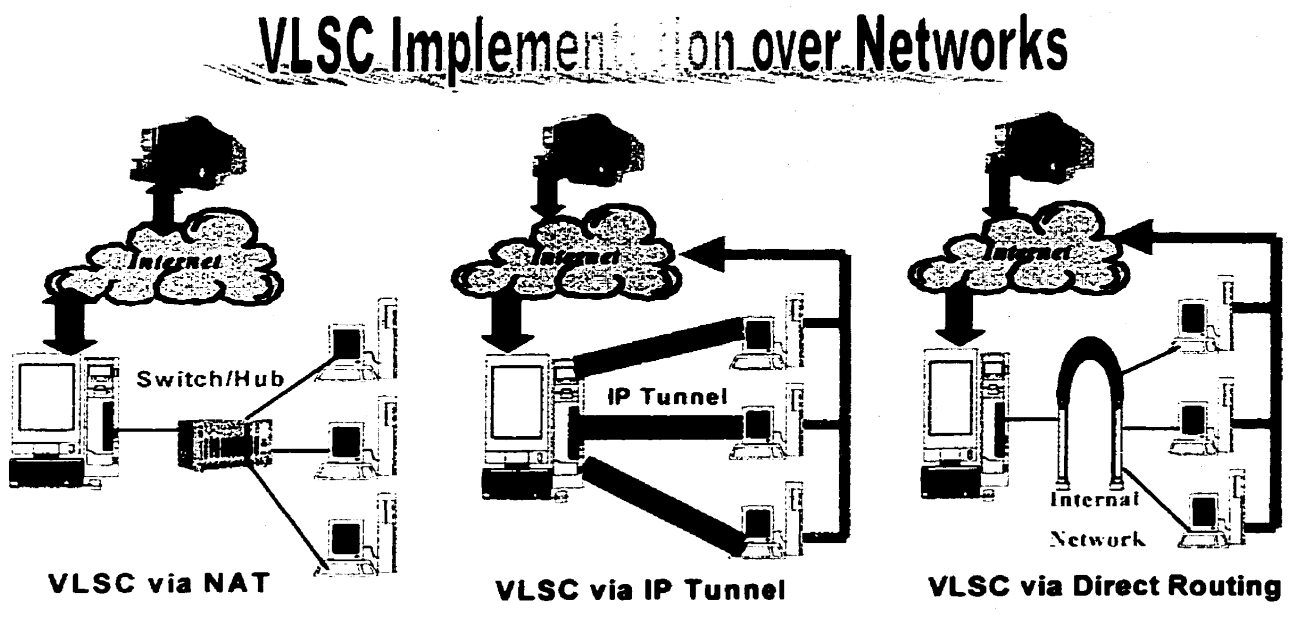

Fig. 5.2. Linux VLSC implementation over IP networks.

5.1.2.2 Linux Virtual Server via IP Tunnel (VLSC/TUN)

IP tunneling, or IP encapsulation, is a technique to encapsulate IP datagram within IP datagram, which allows datagrams destinated for one IP address to be wrapped and redirected to another IP address.

This technique is used to build a virtual server. The load balancer tunnels the client request packets to the different servers, and the servers then process the requests and return the results to the clients directly; thus, the service can still appear as a virtual service on a single IP address.

The workflow of VLSC/TUN is the similar to that of VLSC/NAT, except that the load balancer only deals with inbound traffic. When an incoming packet is destined for the virtual IP address arrives at the switching, the load balancer encapsulates the packet within an IP datagram and forwards it to a dynamically selected server; when the server receives the encapsulated packet, it decapsulates the packet and finds that the inside packet is destined for a VIP that is on its tunnel device, so it processes the request and returns the results to the user directly.

In this approach, real servers can have any real IP address in any network and they can be geographically distributed, but they must support IP tunneling protocol and they all have one of their tunnel devices configured with VIP (virtual IP address).

5.1.2.3 Linux Virtual Server via Direct Routing (LVSC/DR)

In the Direct Routing (DR) approach, there is a sort of internal network to interconnect the load balancer and real servers. That is, the load balancer and real servers must have one of their interfaces physically linked by an uninterrupted segment of LAN such as a Hub/Switch. All real servers have their loopback alias interfaces configured with the virtual IP address, and the load balancer has an interface configured with the virtual IP address to accept incoming packets, so the VIP (virtual IP address) is shared by both real servers and the load balancer.

The workflow of VLSC/DR is the same as those of VLSC/NAT and VLSC/TUN. The load balancer routes a packet directly to the selected server, i.e., the load balancer simply changes the MAC address of the data frame into that of the server, and then retransmits it on the LAN. When the server receives the forwarded packet, it finds that the packet is destined for the address on its loopback aliased with its interface. It then processes the request, and finally returns the result directly to the user.

Using this approach, the real server's interfaces that are configured with virtual IP address should not do ARP (Address Resolution Protocol) response, otherwise there would be a collision if the interface to accept incoming traffic for VIP and the interfaces of real servers are in the same network.

5.1.2.4 Comparison of the Three Approaches

In the VLSC/NAT approach, only one IP address is needed for the load balancer and some private network IP addresses can be used for real servers. Real servers can run any operating system that supports TCP/IP protocol, so it is easy to operate. On the other hand, the load balancer needs to deal with both inbound and outbound traffic, i.e., both request and response packets need to be rewritten by the load balancer. The load balancer may become a bottleneck of the whole system when the number of server nodes increases to around 20 which depends on the throughput of servers. So, the disadvantage is that scalability is limited.

In the VSC/TUN approach, the load balancer simply directs requests to real servers and real servers reply to clients directly, so it may schedule over 100 general real servers; for instance, for most of the Internet service (such as Web service) in which request packets are short and response packets usually carry large amount of data, and it won't become a bottleneck of the system. Because of such scalability, it is extremely good to use LVSC/TUN to build a virtual server that takes a huge load, i.e., the proxy server receiving requests from clients can access the Internet directly, and return services to clients directly. However, LVSC/TUN requires that servers support IP Tunneling protocol; since the IP tunneling protocol is becoming a standard of all operating systems, this approach should be applicable to servers running other operating systems.

Similarly in VLSC/DR approach, the load balancer processes only the client-to-server half of a connection, and the response packets can follow separate network routes to the client. This can greatly increase the scalability of the virtual server. However, there are some specific network configurations required, i.e., the server OS (Operating System) has loopback alias the interface that doesn't do ARP response, and the load balancer and each server must be directly connected to one another by a single uninterrupted segment of a local-area network.

Table 5.1 lists some characteristics of these three strategies; the comparison is based on the server-applied conditions, including server type (i.e., the requirements of servers applicable for these approaches), server network (i.e., applicable and required network type), server number (i.e., server node number that the load balancer could handle), and server gateway (i.e., gateway type).

Table 5.1. The comparison of VLSC/NAT, VLSC/TUN and VLSC/DR.

Server-applied condition	VLSC/NAT	VLSC/TUN	VLSC/DR
server type	any	tunneling	non-arp dev
server network	private	LAN/WAN	LAN
server number	low (10 ~ 20)	high (100)	high (100)
server gateway	load balancer	own router	own router

Basically there are four kinds of scheduling algorithms implemented on LVSC, i.e., Round Robin, Weighted Round Robin, Least Connection, and Weighted Least Connection, which all are discussed in Chapter 3. Other issues, such as cluster management, are omitted due to limited space.

5.2. Application Example of SmartSwitch Router

Based on what we have discussed previously, we now use all of these advanced technologies, methodologies, system architecture and system implementation. In this section, we provide a practical application example of system design, called SmartSwitch Router, which is illustrated in Fig. 5.3.

5.2.1 Concept of SmartSwitch Router

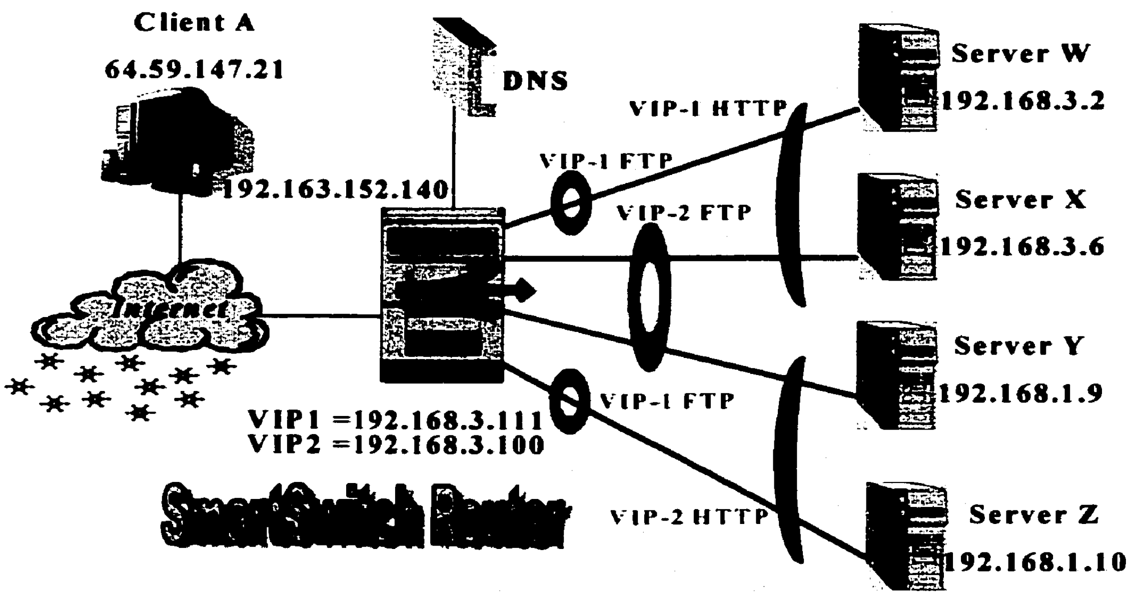

Fig. 5.3. An application example of SmartSwitch Router.

Figure 5.3. shows a kind of distributed environment across different network domains. There are three kinds of different host machines, i.e., the client machine, switching machine, and server machines, with each machine having its own IP address. The client machine (Client A) has IP address 64.59.147.21 (carltn.mb.wave.com.ca) which is belonged to Videon Cable Systems. The switching machine (i.e., SmartSwitch Router) has IP address at 190.163.152.140 (celato.win.trlabs.ca) and is owned by TR*Labs*; this switch sits outside the TR*Labs* firewall and is registered with DNS server. There are four server machines that alias the switching machine, i.e., Server W (192.168.3.2, caviar.win.trlabs.ca), Server X (192.168.3.6, truffle.win.trlabs.ca), Server Y (192.168.1.9, chekov.win.trlabs.ca), and Server Z (192.168.1.10, mudd.win.trlabs.ca). All of these servers reside inside the firewall.

Some specific configurations are needed for both switching machine and server machines. For example, SmartSwitch is registered here by two virtual server clusters (expressed by VIP), i.e., VIP1 (192.168.3.111, maggie.win.trlabs.ca), and VIP2 (192.168.3.100, moe.win.trlabs.ca). Each VIP is "wired" (registered) by a number of servers which belong to this VIP subnetwork, i.e., VIP1 affiliation members are Server W and Server X, and VIP2 is associated with Server Y and Server Z. All of these registrations should be configured beforehand. In addition, all of the servers (W, X, Y, and Z) are capable of providing a variety of Internet services (such as Web, HTTP, etc.), and they all hide their IP addresses from the public (i.e., clients); In this case, all servers most be installed and configured in a specific manner.

The SmartSwitch Router basically functions by accepting incoming requests from local or Internet clients, classifying application traffic based on port number, selecting the best servers to make connections based on intelligent switching decisions, and finally redirecting entire traffic to different server clusters. For example, one session of *ftp* traffic could be distributed to Server W (15% traffic load), Server Z (50%), Server X (25%), and Server Y (10%). Similarly, if a client sent a *http* request only to VIP-1, then redirected traffic might be allocated to Server W (40%) and Server X (60%); if the request was only sent to VIP-2, then traffic distribution could possibly be Server Y (20%) and Server Z (80%).

5.2.2 Software Development Design

Software engineering, generally speaking, is the study of the principles and techniques involved in the systematic development of programs to solve problems. Broadly, it is concerned with improving software reliability, and it appears to meet the need for successful development of large software systems that came along with the increased computer power. Software engineering [WaMi98] has exploded in recent years and now common involves the following areas: a) Structured programming methodologies, i.e., Top-down development (TDD), Abstract data types (ADTs), and Object-oriented programming (OOP). b) Testing of software using white box and black box testing. c) Software quality metrics relevant to the quantitative measurement of the integrity of programs. d) Formal verification of software for pre and post conditions and proofs of correctness. e) Formal specification of software concerning the executable and non-executable specification languages and refinement techniques.

As application software developers within the context of this thesis, we need to focus more on structured programming methodologies. The Top-down development attempts to solve a large problem in terms of smaller sub-problems that are functionally decomposed. ADTs were developed to support "programming in the large", and are effectively abstracting the problem so that there is no need to consider all the details of solving it; with a large program, further functional decomposition may be required; that is, functions might be further divided and conquered, and this process continues until coding becomes "easy", which is known as step-wise refinement. With OOSE (Object-oriented Software Engineering), recently the concepts of OOD (Object-oriented Design) and OOP have become more useful because of some distinguishing characteristics such as encapsulation, inheritance and polymorphism, etc.

For achieving a good modular design, we should take care of the following three concepts:

- ABSTRACTION AND INFORMATION HIDING: We define an abstract data type (or ADT) which is a collection of data and a set of operations on data. Then we will implement the ADT by using a data structure, which is a construct that we can define within a programming language to store a collection of data. Information hiding limits the ways in which we need to deal with functions and data; by making specifications *private* (as opposed to *public*), it presents its information outside itself by using abstraction.

- OBJECT-ORIENTED DESIGN: One way to achieve a modular solution is by identifying within a problem components, called objects, that combine data and operation on data. Such an object-oriented approach to modularity produces a collection of objects that have behaviors. A set of objects of the same type is called a class (such as, in C++ and Java, an object is an instance of a class). Object-oriented programming, or OOP, has the following principles: a) Encapsulation: objects combine data and operations; b) Inheritance: classes can inherit properties from other classes; c) Polymorphism: objects can determine appropriate operations at execution time.
- TOP-DOWN DESIGN: When we need to design an algorithm for a particular function, or sometimes when the emphasis of the problem is on algorithms and not data, a top-down design will lead to a modular solution. The philosophy of a top-down design is that we should address a task at successively lower levels of details. A structure chart can be used to illustrate the hierarchy of and interaction among the modules.

There are two kinds of object-oriented programming languages: C++ and Java. Basically, C++ is beneficial for building data structure and using less computing memory, while Java is platform independent and more suitable for network programming such as portable servers, etc. Since we are primarily considering network programming, we use Java as our programming language.

5.2.3 TCP Traffic Intelligent Redirector

Recall that a layer-3 router makes switching decisions based on the IP addresses of packets, and a layer-4 switch uses not only IP addresses, but also the TCP/UDP header information to make routing decisions. The layer-7 switch forwards packets using more information, i.e., the payload of packets in the application layer.

The TCP/UDP header contains specific source and destination ports on which well-known applications are running (e.g., FTP server runs on port 20/21, HTTP server runs on port 80). We even can create our own protocols and applications running on some specific ports available for us.

The layer-4/7 switching can handle application requests received on a virtual IP address, and redirect the traffic stream to different servers based on the real servers' IP addresses, so it manages a number of backend servers which provide a certain service.

In this section, we provide the prototype and programming of one kind of TCP traffic intelligent redirector.

5.2.3.1 Prototype of TCP Redirector

The workflow of this sort of TCP traffic intelligent redirector is as follows: 1) open a server socket to receive requests from clients, and put them into the input receiving queue; 2) classify traffic packets based on the port number and request packet information, and store classified packets into the input multiple queues; 3) call server load balancing algorithms to make intelligent switching decisions; 4) perform network address translation; 5) create the new format packets that will be sending out (being redirected) and put these packets into the output multiple queues; 6) open a number of sockets to make connections to selected servers. 7) send packets to server clusters.

Fig. 5.4 shows the designed prototype of TCP Intelligent Switching, including the following modules: Ports, four Queues (Input Queue, Traffic mQueues, Ready Queue, Output mQueues), Switching Decisions, three database (Farms DB, Files DB, Conns DB), SLB, Monitoring Functions, Feedback Information, Connections Agent, and two Policies (Classification Policy, QoS Pol icy).

The traffic flow in Fig. 5.4 starts from the client sending requests across the Internet, going into Intelligent Switch. It is then redirected to Server Clusters. Finally, the outbound traffic with different bandwidth from different servers are sent out directly to the client. Within Intelligent Switch, the incoming traffic first goes into the *Input Queue*, then packets are classified and cached into the *Traffic mQueues*. When *Switching Decisions* receives packets from the *Traffic mQueues*, it makes intelligent decisions, creates the packets with new formats and sends them into the *Ready Queue*. Next, these ready packets are cached into the *Output mQueues* operating based on *QoS Policy*. Lastly, the *Connection Agent* sends these packets into Server Clusters.

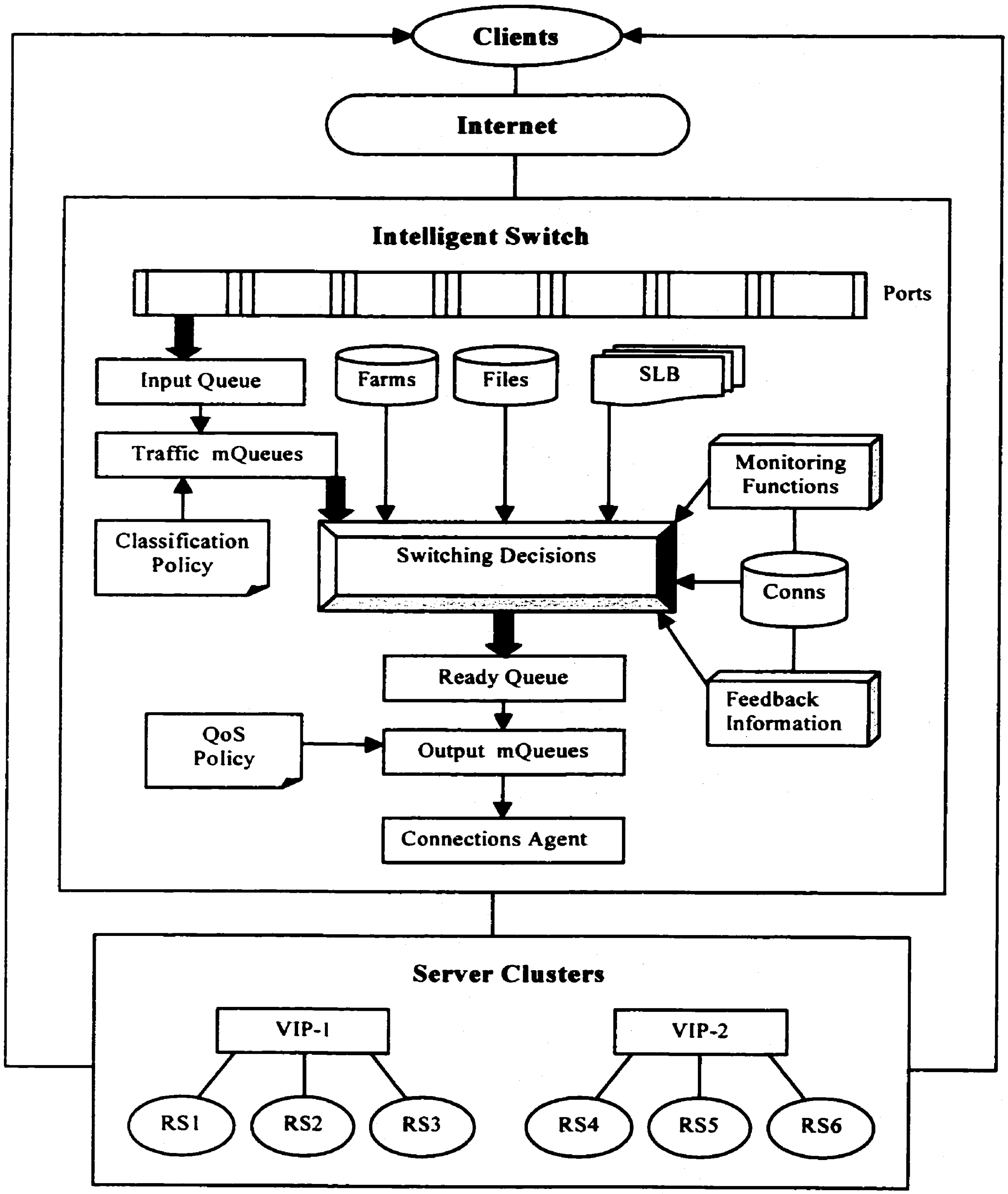

Fig. 5.4. A prototype of the TCP traffic intelligent redirector.

5.2.3.2 Prototype Functions

Ports: In the context of computer and telecommunication networks, ports refers to the port number relevant to some specific network applications. They can be used for receiving client requests and for intelligent traffic classification. RFC1700 [RePo94] defines some well-known port numbers (0-1024), as the software developers of network applications, we usually pick up the port number to be in the range of 1024-65535. For instance, I choose port 5000 as my intelligent switch server program.

Database Farms stores some registration information of server clusters (farms) that register with the switching machine and will bind to it when connected. The information includes: server host name (or host IP address), host port, server weight, and server connection number. These data will be applied to the server load balance algorithm (SLB), network address translation, and forming new sending packets and making connections to servers.

Database Files stores some registration information of files available on server clusters. The information includes: file name, and file size. This sort of data is used to form the new kind of ready packets that are prepared to be sent into the *Ready Queue*.

Database Conns stores some connection information that includes server clusters' connections to the switch machine and clients' request connections to the switch machine, such as, server binding table, server connection number and client request number. This sort of data is used for *SLB*.

Input Queue caches client requests and packets being prepared for traffic classification.

Traffic mQueues: These multiple queues cache different traffic incoming requests and packets from clients. These incoming requests of traffic are classified by different priorities based on *Classification Policy* rules. This part is used for a portion of QoS implementation, i.e., incoming traffic classification.

Ready Queue caches the ready packets that are handled by Switch Decisions. This cache contains some kind of redirection information, such as farm connections (host name, host port), and send requests (file name, file byte, start position, end position, client received port).

Output mQueues: These multiple queues cache different ready packets that will be sent out, or distributed, to server clusters. These outgoing streams of traffic are classified by different priorities based on *QoS policy* rules. This part is used for a portion of QoS implementation, i.e., outbound traffic bandwidth management.

Classification Policy generates some classification policies based on *Ports* by which the incoming requests could be recognized. These policies are used by *Traffic mQueues*.

QoS Policy creates some QoS policies based on bandwidth management. These policies are used by *Output mQueues*.

Connections Agent is used to make connections, and to send requests and packets to server clusters.

SLB implements the server load balancing scheduling, including four kinds of algorithms: Round Robin (RR), Weighted Round Robin (WRR), Least Connection (LC), and Weighted Least Connection (WLC).

Switching Decisions makes intelligent switching decisions. It first fetches the incoming packets from *Traffic mQueues*, then it makes switching decisions based on the client request information, database *Farms*, *Files* and *Conns*, *Monitoring functions*, *Feedback information*, and *SLB*. After having created the new format of sending packets, it puts these ready packets into the *Ready Queue*.

Monitoring Functions provides some monitoring information of LAN devices, such as SNMP data and sysLogs data.

Feedback Information records and provides some information about feedback monitoring and connection quality, such as connection number, availability, latency, jitter, and capacity.

5.2.4 Programming Environment

"Java: A simple, object-oriented, network-savvy, robust, secure, architecture neutral, portable, high-performance, multithread, dynamic language." [Haro96]

Our TCP traffic intelligent redirector is implemented in the Java language. Fig 5.5 shows the flow-chart of the Java implementation within the distributed environment.

5.2.4.1 Flow chart of the Java Implementation

There are three kinds of host machines in this distributed environment, i.e., Client Machine, Switch Machine, and RS M-1, RS M-2, and RS M-3 (real server machines). On Client Machine, two kinds of ports are needed: Port1 for sending requests to Switch Machine, and Port2 for receiving traffic from RS M-1, RS M-2, and RS M-3. On the RS Machines, one kind of server socket (RSS) is opened for accepting the redirected traffic from Switch Machine and another kind of socket (RSO) is opened for sending services back to Client Machine. On Switching Machine, one server socket (SSocket on port 5000) is opened for receiving requests from Client Machine, another three sockets (S1, S2, S3) are opened for distributing relocated traffic to the RS Machines.

Switch Machine runs a main program SwitchServer with multiple threads: thread 1 (T1: Incoming Handler), thread 2 (T2: SLB), thread 3 (T3: Outgoing Handler), and thread 4 (T4: Feedback Handler). *IncomingHandler* deals with the inbound traffic, such as client connections, packets cache into and between Input Queue and Traffic mQueue; *OutgoingHandler* handles the outbound traffic, such as Ready Queue, Output mQueues, and Connections Agent; *SLBHandler* copes with server load balance algorithms, using database (DBs) information and monitoring functions; *FeedbackHandler* is responsible for recording and updating server and client connections.

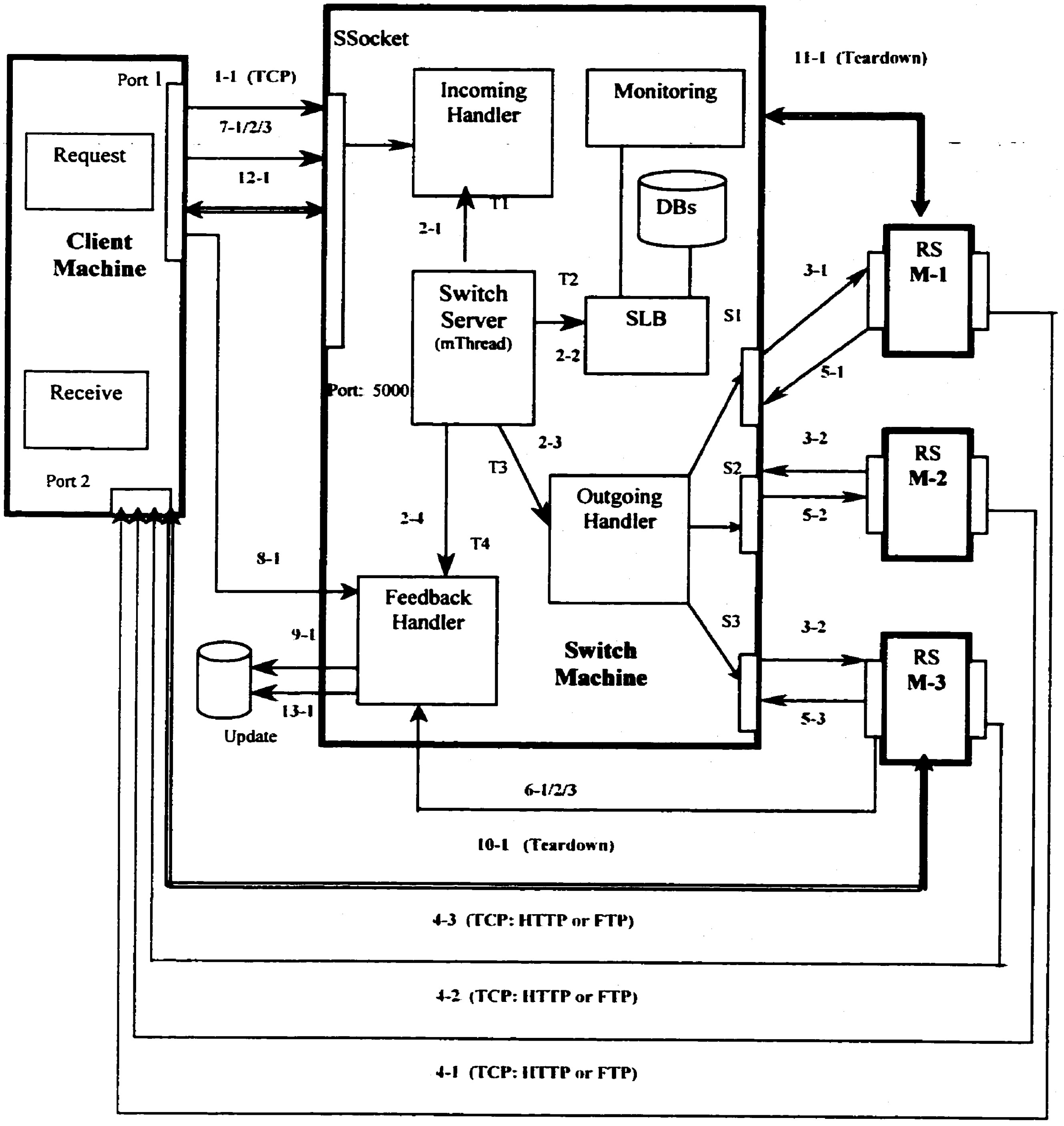

Fig. 5.5. The flow chart of the Java implementation in the distributed environment.

The workflow of operations for this programming is described as follows:

Step 1: Client Machine opens a socket, and sends a TCP request to Switch Machine.

Step 2: Switch Machine opens a server socket to wait for client requests. The SwitchServer program starts a main thread, then it installs and starts Incoming Handler in step 2-1, installs and starts SLB Handler in step 2-2, installs and starts Outgoing Handler in step 2-3, and installs and starts Feedback Handler in step 2-4.

Step 3: After switching decisions, Outgoing Handler picks up three sockets to make connections to the three best real servers selected. In step 3-1, Outgoing Handler starts a thread, opening a socket, and then sending a TCP request and packet to RS M-1. In step 3-2, it starts a second thread, opens a second socket, and sends a TCP request and packet to RS M-2. In step 3-3, it starts the third thread, opens the third socket, and sends a TCP request and packet to RS M-3.

Step 4: The RS machines open server sockets (waiting for the Switch Machine distribution requests) and complete server services, then open new sockets to send TCP streams (e.g., http or ftp files) back to Client Machine. In steps 4-1, 4-2, and 4-3, RS M-1, RS M-2, and RS M-3 will send back to Client Machine, respectively.

Step 5: After getting the sending jobs done, the RS Machines notify Switch Machine for Outgoing Handler to yield running threads. In steps 5-1, 5-2, and 5-3, RS M-1, RS M-2, and RS M-3 do so, respectively.

Step 6: After RS Machine get the sending jobs done, they will notify Switch Machine for Feedback Handler to update the records of connection data. In steps 6-1, 6-2, and 6-3, RS M-1, RS M-2, and RS M-3 do so, respectively.

Step 7: When Client Machine receives the TCP stream from each RS Machine, it will notify Switch Machine. In steps 7-1, 7-2, and 7-3, the received message is from RS M-1, RS M-2,and RS M-3, respectively.

Step 8-1: Once all of TCP streams are received from all three RS machines, Client Machine will notify Switch Machine for Feedback Handler to update the records of client connection data.

Step 9-1: Feedback Handler updates the connections information on the central database.

Step 10-1: Client Machine and RS machines close sockets, disconnect from each other, and tear-down communications.

Step 11-1: Switch Machine and RS Machines close sockets, disconnect from each other, and tear-down communications.

Step 12-1: Client Machine and Switch Machine close sockets and disconnect each other, teardown communications.

Step 13-1: Feedback Handler updates all of the teardown information.

5.2.4.2 Java Programming Functions

Some primary programming functions on different host machines are provided.

- ON CLIENT MACHINE:

GenericClient.java is a generic client program, being able to connect to a server on the port at a specified host. It reads text from the console and sends it to the server, and also reads text from the server and sends it to the console.

ClientReceiver.java provides receiving functions, such as opening a server socket to accept real server clusters connections and transmission traffic, receiving the transferring files with different sizes from different IP address machines, and arranging different portions of a file in the right order to form an entire file.

- ON SWITCH MACHINE:

SwitchServer.java is a multithread server program, providing a main method including a main thread sserver, installing multiple handlers, i.e., IncomingHandler, OutgoingHandler, SLBHandler, and FeedbackHandler, and starting multiple thread instances, i.e., inHanlder, outHandler, slbHandler, fbHandler.

IncomingHandler.java: This class deals with clients' connections, opening a server socket and socket, accepting client requests, getting some information from client input packets (e.g., host-name or IP address, portnum1, portnum2, filename), creating a new kind of packets (Received-Packet), and adding them *into inputBuf Queue*.

OutgoingHandler.java: This class deals with real server clusters' connections. It fetches "ready packets" (CachePacket) from outputBuf Queue, opens sockets to send the redirected requests and packets to RS Machines, and receives "completion notifications" from the RS Machines.

SLBHandler.java: This class implements four kinds of SLB algorithms, i.e., RRobin, WeightR-Robin, LeastConnect, WeightLeastConnect. It fetches the input packets from inputBuf, calls functions of SLB algorithms, forms the new "CachePacket" and puts them into the output Queue.

FeedbackHandler.java: This class takes care of feedback information. It listens to feedback from the RS Machines and Client Machine, and records and updates server clusters' and clients' con-nections, e.g., connectCount.

PACKAGE NODE: This package includes the following Java classes: *Hostnode.class*, *Filen-ode.class*, *ClientInfo.class*, *LinkableHost.class*, *LinkableFile.class*, *ReadHostfile.class*, and *Read-Filefile.class*.

Hostnode.java: This class takes care of the registration information of a real server machine as a host node. An object instantiated using this class can be used to represent a node that is organized according to the following format:

| hostServer Name | hostServer Port | connection Weight | connection Count |

Filenode.java: This class takes care of the registration information of files available on a real server machine as a file node. An object instantiated using this class can be used to represent a node that is organized according to the following format:

| registeredFile name | registeredFile Size |

ClientInfo.java: This class stores client information of connections and disconnections.

LinkableHost.java: This class extends LinkedList class and implements the Linkable interface.

LinkableFile.java: This class also extends LinkedList class and implements the Linkable interface.

ReadHostfile.java: This class reads a file (HOST.dat) to get registered-host information, and inserts the records into HostLinkList.

ReadFilefile.java: this class reads a file (FILE.dat) to get registered-file information, and inserts the records into FileLinkList.

PACKAGE PACKET: this package includes two classes: *CachePacket.class* and *ReceivedPacket.class*.

CachePacket.java deals with creating an output packet (CachePacket). An object instantiated using this class can be used to represent a packet that is organized according to the following format:

| farmName | farmPort | sentTofarmRequest |

ReceivedPacket.java deals with creating a incoming packet (ReceivedPacket). An object instantiated using this class can be used to represent a packet that is organized according to the following format:

| clientName | clientPort | clientReceivePort | fileName |

PACKAGE TRAFFIC includes the following classes: *ChatListener.class*, *ListenerMore.class*, *Chatalker.class* and *TalkerMore.class*.

ChatListener.class listens on a socket and receives InputStream from other host machine.

ListenerMore.class listens on a socket and receives InputStream from other host machine, given a threadName.

ChatTalker.class talks on a socket and sends OutPutStream to another host machine, using screen input.

TalkerMore.class talks on a socket and sends OutPutStream to another host machine, using both screen input and InputStream.

PACKAGE UTIL: This package includes three classes: *Queue.class*, *LinkedList.class and Globle-Data.class*.

Queue.class implements a FIFO (first in, first out) data-structure, by which Objects are added to the front of the queue and removed from the back.

LinkedList.class provides a data structure called Linked List, which is used here for database storage. This class needs to implement the Linkable interface when called.

GlobalData.java defines some global information as static final variables, such as maxHost, maxConnection, maxFile and maxSwitch.

- ON SERVER FARMS MACHINES

FarmServerD.java includes a main method. It opens a server socket to receive a request and packets from Switch Machine, and starts a farmRequest handler thread.

FarmRequestHandler.java starts another thread to deal with file transfer service. It opens a socket to make connections to Client Machine.

SendLocalFile.java implements a sending job. It fetches a file of a certain number of bytes and sends it to Client Machine.

The software tests and system experiments will be presented in the next chapter (System Experiments and Analysis).

Chapter 6

System Experiments & Analysis

This chapter presents system experiments and analysis, including test environments, experiments and results, and results analysis. Section 6.1 describes the software test environment of the TCP traffic intelligent redirector. Section 6.2 provides experiments and results, including two kinds of tests: system test and SLB algorithms performance tests. Section 6.3 gives some results analysis based on both experiment results and theoretical prediction.

6.1. Test Environments

The TCP traffic intelligent redirector, discussed in Section 5.2, is tested on Unix Workstations (Sun Microsystem Solaris 5.7) at the Department of Electrical and Computer Engineering, University of Manitoba. Fig. 6.1 shows the diagram of the test environment, an example of a distributed environment. There are five host machines, one computer setup as a client machine, one computer as a switch machine, and others act as three real server (RServer) machines. This kind of multiple network communication starts with client sending requests to the switch machine; traffic flow is shown as one biggest light yellow arrow between the client machine and the switch machine. The switch machine then makes intelligent decisions and redirects traffic to the different RServer machines (real servers); traffic flows are shown as three blue arrows between the switch machine and RServer machines. Finally, three RServer machines each distribute a specific amount of bandwidth back to the client machine; traffic flows are represented by three dark yellow lines and six arrows. Different port numbers are chosen here for implementing this multi-communication, i.e., ports:9100/9200/9300 on the client machine, port:5000 on the switch machine, and port:7000 on RServer. We could not use some well-known port numbers defined in [RePo94] because these port numbers are already being used for the specific server services installed on most Unix workstations. Some details of various machines are described as the follows:

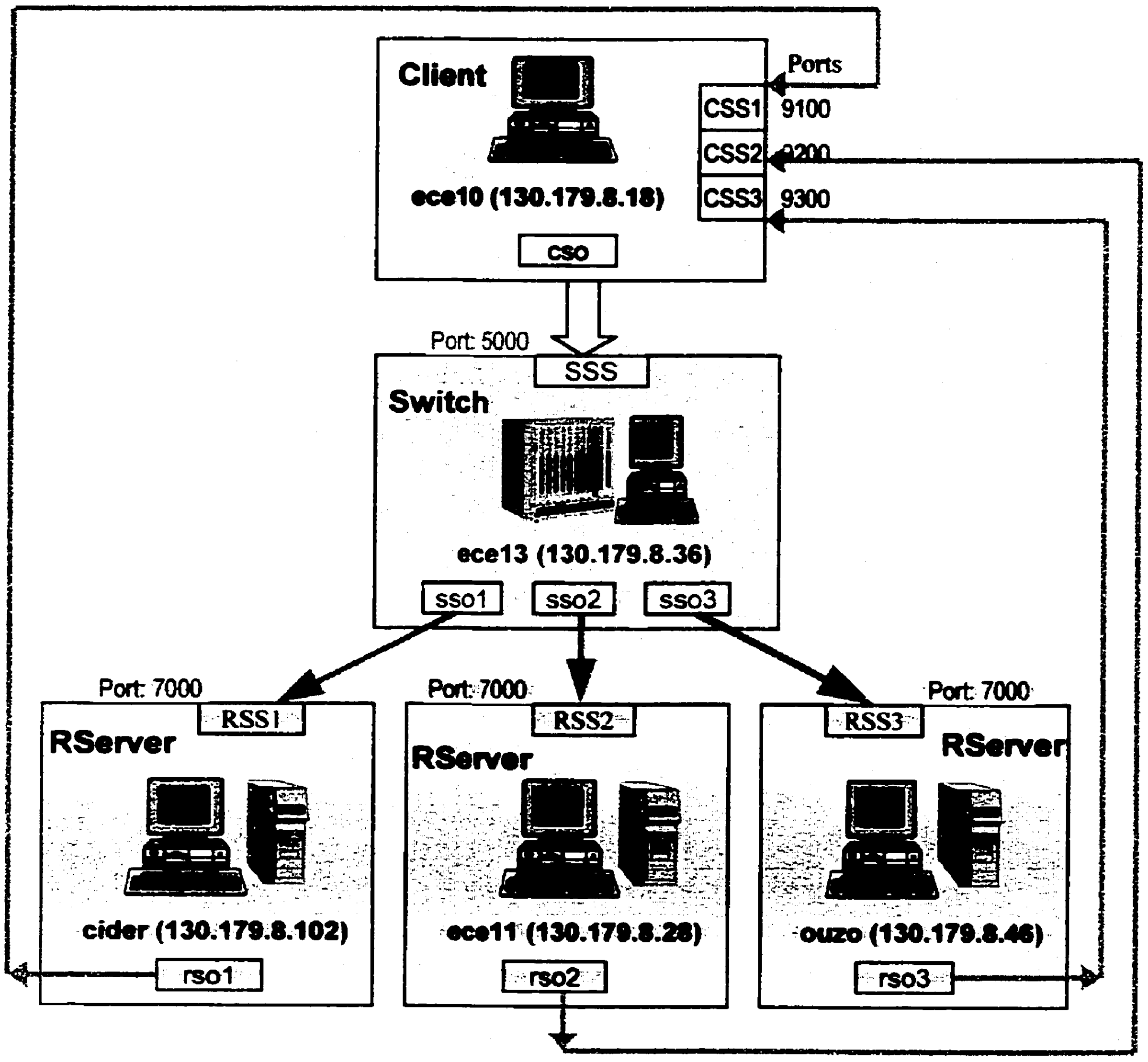

Fig. 6.1. Test environments of the TCP traffic intelligent redirector.

Notes:

CSS1, CSS2, CSS3: Client (machine) Server Sockets 1/2/3.

SSS: Switch (machine) Server Socket.

RSS1, RSS2, RSS3: Real Server (machine) Server Sockets 1/2/3.

cso: client (machine) socket.

sso1, sso2, sso3: switch (machine) sockets 1/2/3.

rso1, rso2, rso3: real server (machine) sockets 1/2/3.

CLIENT MACHINE: **ece10 (IP address: 130.179.8.18)** acts as a client host, opening a socket (cso) to make a connection to the switch machine; in the meantime, the client machine also needs to open three Server Sockets (CSS1, CSS2, CSS3) at ports: *9100, 9200, and 9300* (default port:*9000*) in order to receive connections from RServer machines. Client host will launch a client program to send a TCP request to the switch machine when all of the server programs on the switch machine and RServer machines start and are ready to accept client requests.

SWITCHING MACHINE: **ece13 (130.179.8.36)** is used as a TCP traffic redirector. Firstly, a server socket (SSS) on this host is opened for receiving client requests at port:*5000.* Once the client request is accepted, the switching server programs then call some SLB algorithms that will choose the best server IP address among the RServers (real server clusters) for making the next connection to RServer. After intelligent decisions are made, the switching server programs run the substitution of IP address translation; that is, they replace (swap) the client machine IP address (130.179.8.18) with the switch machine IP address (130.179.8.36). Next, the switch machine needs to open three sockets (sso1, sso2, and sso3) to make connections to the RServers (i.e., cider, ece11, and ouzo), if three real server hosts are selected for one session TCP traffic switching. These processes run automatically on the switch machine when the client sends a request to it, and the RServer server programs start and are ready for server services.

SERVER CLUSTER MACHINES: **cider (130.179.8.102)**, **ece11 (130.179.8.28)**, and **ouzo (130.179.8.46)** are used as real servers (RServer1, RServer2, and RServer3, respectively). The RServer machines open three Server Sockets (RSS1, RSS2, RSS3, respectively, each for each machine), in order to receive connections and requests from the switching machine. After accepting request packets, the RServer machines launch server programs, processing service requests (e.g., send a *ftp* or *http* file), and translating the IP address in the packet header in preparation for sending to clients (i.e., replace the switch machine IP address:130.170.8.36 by the RServer machine IP address: 130.179.8.102, 130.179.8.28, and 130.179.8.46, respectively). Next, the RServer machines need to open three sockets (rso1, rso2, and rso3, respectively) to communicate with the client machine. Finally, various traffic distributed from RServer1, RServer2, and Rserver3 will go to the client machine at port: 9100, 9200, and 9300 respectively.

In this sort of distributed test environment of a multi-communication network system, there are three kinds of host machines that run different program to function as client, switching, and server clusters. However, each of these machines act as not only a simple client, switching, or service server, but as a proxy machines as well; that is, the whole system actually forms a kind of connection-oriented circular link when all programs launch. Therefore, any interruption of programs on any machine can cause partial teardown of communications. For example, if the switching server programs (or machine) shut down, the connection between Client and Switch, and the connection between Switch and RServers will be disconnected. If the client's requesting program stops running, then Switch does not receive any request; it will thus not make any connection to the real server machines. If the client's receiving program suspends, then the traffic from the real server machines will not reach the client machine. If one or all of the real server programs do not launch, then the client machine will not get all of the bandwidth requested. Test1 (system test) in section 6.2 presents and verifies that this sort of multi-linked communication across on entire network system works out well.

The software development of this TCP intelligent traffic redirector is basically designed to implement a kind of TCP session switching, therefore it is a session-based traffic redirector. That is, all tests are session-based. In other words, over the course of one test, an entire TCP session traffic is being processed from its beginning to its teardown. Each test run is equivalent to one session. More details on session-switching tests will be presented through Test2 (SLB Algorithms Performance Tests) in section 6.2. Nevertheless, this software design and implementation also include some concepts and methodologies of application-layer switching, e.g., TCP splicing, which is one portion of layer-7 intelligent switching mentioned in Section 3.4. in this thesis.

6.2. Experiments and Results

Based on the test environment illustrated above, we have two kinds of experiments tested, namely, Test1 (system test) and Test2 (SLB algorithms performance tests). The next sections will describe these tests in detail.

6.2.1. Test1: System Test

The purpose of this test is to demonstrate that designed traffic redirector of a TCP session is able to build multiple connections intelligently across the whole communication network system illustrated in section 6.1. The intelligence of switching on selection decisions to the best servers can be presented by running different SLB algorithms.

The procedure of this test as follow. Firstly, we launch a variety of client/server programs on different host machines. Next, we monitor messages on several consoles to verify successful connections.

The following Java programs are launched on Client host machine, Switch host machine, and RServer host machines:

Client Machine:

```
ece10> java GenericClient <host> <port>
ece10> java Receive1/2/3 <port>
```

i.e.:

```
ece10> java GenericClient ece13.ee.umanitoba.ca 5000
ece10> java Receive1 9100
ece10> java Receive2 9200
ece10> java Receive3 9300
```

Note: <host> is the host name, <port> is the port number.

Switch Machine:

```
ece13> java SwitchTest <port> <SLB> <w1> <w2> <w3> <c1> <c2> <c3>
```

i.e.:

```
ece13> java SwitchTest 5000 RR 1 1 1 0 0 0
ece13> java SwichTest 5000 WRR 2 5 3 0 0 0
ece13> java SwichTest 5000 LC 0 0 0 4 7 5
ece13> java SwichTest 5000 WLC 2 5 3 4 7 5
```

Note: <port> is the port number; <SLB> is the SLB algorithm (RR: Round Robin, WRR: Weighted Round Robin, LC: Least Connection, WLC: Weighted Least Connection); <w> is the weight value currently assigned or registered to the real server (w1: RServer1 weight, w2: RServer2 weight, and w3: RServer3 weight); <c> is the current connection number to the real server (c1: RServer1 connection, c2: RServer2 connection, and c3: Rserver3 connection).

Server Farms:

cider> java MRServer <port> <clientIP> <clientReceivedPort>

i.e.:

```
cider> java MRServer 7000 130.179.8.18 9100
ece11> java MRServer 7000 130.179.8.18 9200
ouzo> java MRServer 7000 130.179.8.18 9300
```

Note: <port> is port number, <clientIP> is client machine IP address, <clientReceivedPort> is the port number on client machine for receiving connection from RServer.

If we have already started all of the Java programs, then we can get some monitoring information on the following eight consoles. For example, we use WRR (weighted round robin) as SLB algorithm, and w1, w2, and w3 are 3, 4, and 2, respectively. The best server machines selected by intelligent switching are: 130.179.8.28, 130.179.8.28, and 130.179.8.102, so the next three connections will be to ece11, ece11, and cider, respectively. There is no traffic directing to machine ouzo during this TCP session because IP address 130.179.8.46 is not chosen.

```
ece13> java SwitchTest 5000 WRR 3 4 2
SLB Algorithm works out. WRRobin done.
The best selected servers are: 130.179.8.28 , 130.179.8.28, 130.179.8.102
 Hello 130.179.8.28
 How are you?
Switch is connecting to Server: 130.179.8.28
 Hello 130.179.8.28
How are you?
Switch is connecting to Server: 130.179.8.28
 Hello 130.179.8.102
How are you?
Switch is connecting to Server: 130.179.8.102
... ...
```

```
ece11> java MRServer 7000 130.179.8.18 9200
Real Server is using port 7000
 Hello 130.179.8.28
How are you?
RealServer ece11/130.179.8.28 is sending back to client.
RS ipAddress ece11/130.179.8.28
 Hello Client, I am RS from ece11/130.179.8.28
How about our services?
Client is receiving at port 9200
... ...
Done
```

```
cider> java MRServer 7000 130.179.8.18 9100
Real Server is using port 7000
 Hello 130.179.8.102
How are you?
RealServer cider/130.179.8.102 is sending back to client.
RS ipAddress cider/130.179.8.102
 Hello Client, I am RS from cider/130.179.8.102
How about our services?
Client is receiving at port 9100
.........
```

```
ouzo> java MRServer 7000 130.179.8.18 9300
Real Server is using port 7000
......
```

```
ece10> java GenericClient ece13.ee.umanitoba.ca 5000
Connected to ece13.ee.umanitoba.ca/130.179.8.36:5000
......
```

```
ece10> java Receive2 9200
The Chat Server is using port 9200
 Hello Client, I am RS from ece11/130.179.8.28
How about our services?
.......
```

```
ece10> java Receive1 9100
The Chat Server is using port 9100
 Hello Client, I am RS from cider/130.179.8.102
How about our services?
.......
```

```
ece10> java Receive3 9300
The Chat Server is using port 9300
......
```

6.2.2. Test2: SLB Algorithms Performance Tests

6.2.2.1 Test Procedure

The purpose of this test is to compare the performance of various SLB algorithms and different approaches by statistic data collection and analysis. The procedure of this test includes the following six steps.

Step 1. Making assumptions:

It is necessary to make certain assumptions for this group of tests. Firstly, we assume that all of the real server host machines in the server clusters have the same capability of handling each TCP session at the same speed, and that no hardware interruptions or configuration problems occur during a session test. We label S1 (cider: 130.179.8.102), S2 (ece11: 130.179.8.28), and S3 (ouzo:130.179. 8.46) for the results of tests in Table 6.1-6.7. We also regard these three servers (S1, S2, and S3) as potential candidates that have the same opportunity to be selected by switching programs to make the next connection; however, the successful choice to each candidate totally depends on intelligent decisions of the SLB algorithms. We also assume each server (S1, S2, and S3) has only one IP address effectively to register the switching database named Farms where some useful information data of server clusters (or farms) are stored for intelligent switching decision. Next we also assume the maximum connection number of each session switching is three for these tests, that is, each TCP client request allows switching SLB algorithms to choose exactly three real server IP addresses for traffic redirection. Finally, we assume each TCP traffic session has the same traffic payload.

Step 2. Running SwitchTest programs:

Based on the test environment and assumptions above, we launch various SwitchTest programs (e.g., *RRtest.java*, *WRRtest.java*, *LCTest.java*, and *WLCTest.java*) augmented with different options (e.g. <port>, <w1>, <w2>, <w3>, <oper>, <c1>, <c2>, <c3>) and/or different values (e.g., w1, w2, w3, c1, c2, c3), then try to run each program for sufficient times (Test Number) to obtain enough test data, for example, 100 times is used in this thesis. ***Test Number*** is shown in the 1st column of Table 6.1-6.7.

Step 3. Collecting data:

The data we need here is the collection of selected RServer's (S1, S2, and S3) IP addresses. The outstanding IP address actually represents the successful opportunity for each RServer that the SLB algorithms intelligent decision gives as the best server for making the next connection during one session switching. For example, if in our first test (Test Number = 1), we get IP address data: 130.179.8.102, 130.179.8.102, 130.179.8.46, then this means during this session test, S1 is selected twice for making connections, S3 is selected once, and unfortunately S2 is not selected, so successful opportunities for S1, S2, and S3 are 2/3, 0/3, and 1/3, respectively. This sort of data collection will be shown in the second column (***Best Selected Servers***) of Table 6.1-6.5 and in the third column of Tables 6.6-6.7.

Step 4. Calculating SLB%:

SLB% represents the percentage of server load balancing (SLB). We have three real server candidates (S1, S2, and S3), and each TCP session switching can pick up three real server IP addresses. If the total SLB value is equal to100% (S1/SLB + S2/SLB + S3/SLB), then Si/SLB (i=1, 2, 3) is the shared percentage of the total SLB for each server (Si) during one TCP session switching. For example, from data collection, we have already shown: 130.179.8.102, 130.179.8.102, 130.179.8.46; this implies that the servers' opportunities are: S1(2/3), S2(0/3), and S3(1/3), so S1/SLB = 2/3 = 66.7%, S2/SLB = 0/3 = 0.0%, and S3/SLB = 1/3 = 33.3%. This kind of data SLB% is shown in the 3rd column (***SLB%, S1, S2, S3***) of Table 6.1~6.5.

Step 5. Obtaining SLB Points:

SLB Points represents the degree of server load balancing (SLB) achieved for each TCP session switching. We classify three kinds of SLB Points: *vu*, *mb*, and *eb**. For example, if SLB% is: S1/SLB=66.7%, S2/SLB=0.0%, and S3=33.3%, then SLB Points is defined as *mb* (moderately balanced); if SLB% is: S1/SLB=33.3%, S2/SLB%=33.3%, and S3/SLB=33.3%, then SLB Points is defined as *eb** (extremely balanced); if SLB% is: S1/SLB=0.0%, S2/SLB=0.0%, and S3/SLB

=100%, then SLB Points is defined as *vu* (very unbalanced). Therefore, *vu* means that server load balance (SLB) is very unbalanced because all of three connections are assigned to the same IP address, that is, only one server is carrying all of the traffic load for this TCP session, which of course results into SLB being very unbalance. In the case of *mb*, there are two servers sharing traffic load, one takes 66.7% and another takes 33.3%, and a third server is idle. Finally, *eb** means all of three servers equally share the entire traffic payload; it is extremely SLB balanced, and it is what we want. This kind of SLB points data is shown in the 4th column (***SLB Points***) of Table 6.1~6.5.

Step 6. Counting statistic SLB points:

Finally, after all session tests are completed, we need to count the number of *vu*, *mb*, and *eb** based on statistic data collection of total test number (i.e., 100 times in this case), so that we can get the statistic SLB points (or probability). For example, in Test 2-1, during 100 test runs, *vu* appeared 15 times, *mb* 63 times, and *eb** 21 times, so the statistic SLB points are: *vb* (15/100), *mb* (63/100), and *eb** (21/100). This sort of statistic data is shown in the bottom rows (***Statistic SLB points***) of Table 6.1~6.5.

6.2.2.2 Results of Tests

The results of tests are given in Tables 6.1-6.7.

Test 2-1 is the Random Round Robin Algorithm Test. We launch program: *RRTest.java*, where w1=w2=w3=1, and c1=c1=c3=0, simply by running: `java RRTest <port> <SLB>` (i.e., `ece13> java RRTest 5000 RR`). After 100 times runs (Test Number = 100) of collection and statistic analysis, then we get the result of Table 6.1. For thess 100 times runs, the statistic SLB points are: *vu*=15/100, *mb*=63/100, and *eb**=21/100; this indicates that when running Round-Robin algorithm 100, there are 15 instances where server load balance (SLB) is very unbalanced, 63 instances where that SLB is moderately balanced, and 21 instance where SLB arrives extremely balanced.

Tests 2-2 to 2-5 are the Weighted Round Robin Algorithm Tests (Approach-1,2,3,4). We launch the program: *WRRTest.java*, where w1=2, w2=5, w3=3, and c1=c2=c3=0, simply by running: `java WRRTest <port> <SLB> <w1> <w2> <w3> <oper>` (i.e., `ece13> java WRRTest 5100 WRR 2 5 3 100/200/300/`

400). If oper=100, then the program is run using approach-1; similarly, if oper=200, then approach-2 is used; if oper=300, then approach-3 is used; if oper=400, then approach-4 is used. After 100 times runs of collection and statistic analysis, we get the results of Table 6.2-6.5. In Table 6.2, Test 2-2 (Weighted Round-Robin: Approach-1, or simply WRR: App1), gives statistic SLB points: *vu*=30/100, *mb*=70/100, and *eb**=0/100. In Table 6.3, Test 2-3 (WRR: App-2) gives statistic SLB points: *vu*=10/100, *mb*=40/100, and *eb**=50/100. In Table 6.4, Test 2-4 (WRR: App-3) gives statistic SLB points: *vu*=10/100, *mb*=30/100, and *eb**=60/100. In Table 6.5, Test 2-5 (WRR: App-4) gives statistic SLB points: *vu*=10/100, *mb*=30/100, and *eb**=60/100. We can see that the value of *eb** has apparently improved by using different approaches, e.g., 0% up to 50% from WRR-1 to WRR-2, and 50% up to 60% from WRR-2 to WRR-3 and WRR-4. More details about approach-1 to approach-4 will be discussed in section 6.3.

Test 2-6 is the Least Connection Algorithm Test. We launch the program: *LCTest.java*, where wt1=wt2=wt3=1, and c1, c2, c3 will vary dynamically, simply running: `java LCTest <port> <SLB> <c1> <c2> <c3>` (e.g., `ece13> java LCTest 5200 LC 4 7 5`). The key point of this algorithm is that the next connection will be distributed to the best server who has the minimum value of current connections, and the connection number (c) is dynamically changed and updated when a new connection is made or an old connection is completed. This is a sort of dynamic algorithm. Test results are shown in Table 6.6. The weights and connections are shown in the second column (***w1 w2 w3 c1 c2 c3***) of Tables 6.6 and 6.7.

Test 2-7 is the Weighted Least Connection Algorithm Test. We launch the program: *java WLCTest.java*, simply by running: `java WLCTest <port> <w1> <w2> <w3> <c1> <c2> <c3>` (e.g. `ece13> java WLCTest 5300 WLC 2 5 3 4 7 5`). All weights (w1, w2, w3) should be assigned ahead of this time or could be later changed by network administrators. The connection number (c1, c2, c3) should be dynamically changed in the same manner as *LCTest*. The key point for this algorithm is that the next connection should be assigned to the best server that holds the minimum value of (Ci/Wi). When the default value of weights is given, i.e. w1=w2=w3=1, then this test becomes *LCTest*. Thus, the Least Connection algorithm is a special case of the Weighted Least Connection algorithm where the weight values are equal to 1. The Weighted Least Connection algorithm has more dynamica characteristics because it actually addresses more factors: weights and connections, which are all network dynamically sensitive. Test results are shown in Table 6.7.

Table 6.1. The result of Test 2-1 (Random Round-Robin Algorithm).

Test Number	Best Selected Servers			SLB%			SLB Points
				S1	S2	S3	
1	130.179.8.102	130.179.8.102	130.179.8.46	66.7	0	33.3	mb
2	130.179.8.46	130.179.8.102	130.179.8.28	33.3	33.3	33.3	eb *
3	130.179.8.102	130.179.8.102	130.179.8.28	66.7	33.3	0	mb
4	130.179.8.46	130.179.8.46	130.179.8.46	0	0	100	vu
5	130.179.8.102	130.179.8.102	130.179.8.46	66.7	0	33.3	mb
6	130.179.8.28	130.179.8.28	130.179.8.28	0	100	0	vu
7	130.179.8.46	130.179.8.46	130.179.8.46	0	0	100	vu
8	130.179.8.46	130.179.8.46	130.179.8.28	0	33.3	66.7	mb
9	130.179.8.46	130.179.8.28	130.179.8.46	0	33.3	66.7	mb
10	130.179.8.46	130.179.8.46	130.179.8.46	0	0	100	vu
....							

100 test runs data collection average:
Statistic SLB points: (vu) very unbalanced (15/100)
(mb) moderately balanced (63/100)
(eb*) extremely balanced (21/100)

Table 6.2. The result of Test 2-2 (Weighted Round-Robin: Approach-1)

Test Number	Best Selected Servers			SLB%			SLB Points
				S1	S2	S3	
1	130.179.8.28	130.179.8.28	130.179.8.28	0	100	0	vu
2	130.179.8.28	130.179.8.28	130.179.8.46	0	66.7	33.3	mb
3	130.179.8.46	130.179.8.46	130.179.8.102	33.3	0	66.7	mb
4	130.179.8.102	130.179.8.28	130.179.8.28	33.3	66.7	0	mb
5	130.179.8.28	130.179.8.28	130.179.8.28	0	100	0	vu
6	130.179.8.46	130.179.8.46	130.179.8.46	0	0	100	vu
7	130.179.8.102	130.179.8.102	130.179.8.28	66.7	33.3	0	mb
8	130.179.8.28	130.179.8.28	130.179.8.28	0	100	0	vb
9	130.179.8.28	130.179.8.46	130.179.8.46	0	33.3	66.7	mb
10	130.179.8.28	130.179.8.102	130.179.8.102	66.7	0	33.3	mb
....							

100 test runs data collection average:
Statistic SLB points: (vu) very unbalanced (30/100)
(mb) moderately balanced (70/100)
(eb*) extremely balanced (0/100)

Table 6.3. The result of Test 2-3 (Weighted Round-Robin: Approach-2).

Test Number	Best Selected Servers			SLB%			SLB Points
				S1	S2	S3	
1	130.179.8.28	130.179.8.28	130.179.8.28	0	100	0	vu
2	130.179.8.46	130.179.8.28	130.179.8.46	0	33.3	66.7	mb
3	130.179.8.102	130.179.8.28	130.179.8.46	33.3	33.3	33.3	eb *
4	130.179.8.102	130.179.8.28	130.179.8.28	33.3	66.7	0	mb
5	130.179.8.28	130.179.8.46	130.179.8.28	0	66.7	33.3	mb
6	130.179.8.46	130.179.8.102	130.179.8.28	33.3	33.3	33.3	eb *
7	130.179.8.46	130.179.8.102	130.179.8.28	33.3	33.3	33.3	eb *
8	130.179.8.28	130.179.8.28	130.179.8.46	0	66.7	33.3	mb
9	130.179.8.28	130.179.8.46	130.179.8.102	33.3	33.3	33.3	eb *
10	130.179.8.28	130.179.8.46	130.179.8.102	33.3	33.3	33.3	eb *
....							

100 test runs data collection average:
Statistic SLB points: (vu) very unbalanced (10/100)
(mb) moderately balanced (40/100)
(eb*) extremely balanced (50/100)

Table 6.4. The result of Test 2-4 (Weighted Round-Robin: Approach-3).

Test Number	Best Selected Servers			SLB%			SLB Points
				S1	S2	S3	
1	130.179.8.28	130.179.8.28	130.179.8.46	0	66.7	33.3	mb
2	130.179.8.28	130.179.8.102	130.179.8.46	33.3	33.3	33.3	eb *
3	130.179.8.28	130.179.8.102	130.179.8.46	33.3	33.3	33.3	eb *
4	130.179.8.28	130.179.8.28	130.179.8.28	0	100	0	vu
5	130.179.8.46	130.179.8.28	130.179.8.102	33.3	33.3	33.3	eb *
6	130.179.8.46	130.179.8.28	130.179.8.102	33.3	33.3	33.3	eb *
7	130.179.8.46	130.179.8.28	130.179.8.28	0	66.7	33.3	mb
8	130.179.8.28	130.179.8.46	130.179.8.28	0	66.7	33.3	mb
9	130.179.8.102	130.179.8.46	130.179.8.28	33.3	33.3	33.3	eb *
10	130.179.8.102	130.179.8.46	130.179.8.28	33.3	33.3	33.3	eb *
....							

100 test runs data collection average:
Statistic SLB points: (vu) very unbalanced (10/100)
(mb) moderately balanced (30/100)
(eb*) extremely balanced (60/100)

Table 6.5. The result of Test 2-5 (Weighted Round-Robin: Approach-4).

Test Number	Best Selected Servers			SLB%			SLB Points
				S1	S2	S3	
1	130.179.8.28	130.179.8.46	130.179.8.102	33.3	33.3	33.3	eb *
2	130.179.8.28	130.179.8.46	130.179.8.102	33.3	33.3	33.3	eb *
3	130.179.8.28	130.179.8.46	130.179.8.28	0	66.7	33.3	mb
4	130.179.8.28	130.179.8.28	130.179.8.46	0	66.7	33.3	mb
5	130.179.8.102	130.179.8.28	130.179.8.46	33.3	33.3	33.3	eb *
6	130.179.8.102	130.179.8.28	130.179.8.46	33.3	33.3	33.3	eb *
7	130.179.8.28	130.179.8.28	130.179.8.28	0	100	0	vu
8	130.179.8.46	130.179.8.102	130.179.8.28	33.3	33.3	33.3	eb *
9	130.179.8.46	130.179.8.102	130.179.8.28	33.3	33.3	33.3	eb *
10	130.179.8.46	130.179.8.28	130.179.8.28	0	66.7	33.3	mb
....							

100 test runs data collection average:
Statistic SLB points: (vu) very unbalanced (10/100)
(mb) moderately balanced (30/100)
(eb*) extremely balanced (60/100)

Table 6.6. The result of Test 2-5 (Least-Connection Algorithm).

Test Number	w1	w2	w3	c1	c2	c3	Best Selected Servers		
1	1	1	1	4	7	5	130.179.8.102	130.179.8.46	130.179.8.46
2	1	1	1	6	3	4	130.179.8.28	130.179.8.46	130.179.8.28
3	1	1	1	1	3	9	130.179.8.102	130.179.8.102	130.179.8.28
4	1	1	1	3	2	3	130.179.8.28	130.179.8.102	130.179.8.46
5	1	1	1	8	10	5	130.179.8.46	130.179.8.46	130.179.8.46
....									

Comment: The next connection goes to the server with the minimum value of connection count number.

Table 6.7. The result of Test 2-7 (Weighted Least-Connection Algorithm).

Test Number	w1	w2	w3	c1	c2	c3	Best Selected Servers		
1	2	3	5	1	2	2	130.179.8.46	130.179.8.28	130.179.8.102
2	1	1	1	2	5	3	130.179.8.102	130.179.8.46	130.179.8.28
3	5	7	10	1	2	15	130.179.8.102	130.179.8.46	130.179.8.28
4	3	2	6	3	2	3	130.179.8.28	130.179.8.102	130.179.8.46
5	9	4	2	8	10	2	130.179.8.102	130.179.8.46	130.179.8.28
....									
Comment: The next connection goes to the server with the minimum ratio of connection number and weight.									

6.3. Result Analysis

Based on the above test results, statistic SLB points in Tables 6.1-6.5 are primarily used here Fig. 6.2 in the following, which is a graph comparing the results of the SLB algorithms performance.

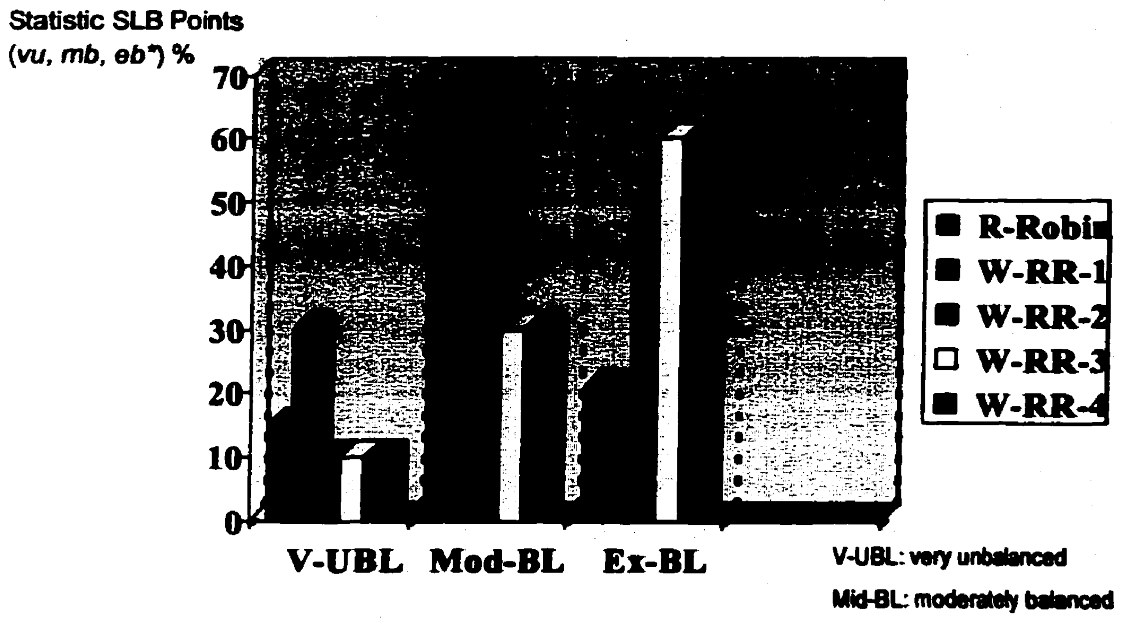

Fig. 6.2. Comparison of the results of the SLB algorithms' performance

Note1:

V-UBL: Very Unbalanced (Server Load), statistic SLB points from *vu*.

Mod-BL: Moderately Balanced (Server Load), statistic SLB points from *mb*.

Ex-BL: Extremely Balanced (Server Load), statistic SLB points from *eb**.

Note2:

R-Robin: Round Robin algorithm, statistic SLB points from Table 6.1.

W-RR-1/2/3/4: Weighted Round Robin algorithm, approach-1/2/3/4, statistic SLB points from Table 6.2, Table 6.3, Table 6.4, and Table 6.5, respectively.

In Figure 6.2, we compare five SLB algorithm approaches: the Random Round Robin algorithm (R-Robin, in dark green), the Weighted Round Robin algorithms (WRR) including four approaches, i.e., W-RR-1 (in dark blue), W-RR-2 (in red), W-RR-3 (in yellow), and W-RR-4 (in purple). We consider three kinds of SLB points (these three areas are separated by vertical dash lines): V-UBL (very unbalanced, shown in the left most area), Mod-BL (moderately balanced, shown in the middle area), and Ex-BL (extremely balanced, shown in the right most area). There are five color bars in each part from the left side to the right side, i.e., green, purple, orange, yellow, and pink, whose values represent statistic SLB points that are labeled in the Y direction on x-y plane. The value of Y-axis is the statistic value of SLB points for *vu*, *mb*, and *eb** in V-UBL, Mod-BL, and Ex-BL, respectively. For example, the red bar (third column in each part) represents W-RR-2 approach, with *vu* (10%) in V-UBL, *mb* (40%) in Mod-BL, and *eb** (50%) in Ex-BL.

We can see that, among of all three dark blue bars in the second column of each area, the value *vu* of V-UBL is 30%, and the value *eb** of Ex-BL is 0%, this means that this is the worst result because there is no server load balance (SLB) point reached by W-RR-1 in this experiment, and very unbalanced server load points is the highest of all. On the other hand, upon exampling the yellow and pink bars, we get the highest value *eb** for EX-BL:60%, and the the lowest value *vu* for V-UBL:10%. This result indicates that W-RR-3 and W-RR-4 are the best approaches in this experiment because we obtain the highest server load balance (SLB) points (60%) and the lowest very unbalanced points (only 10%).

From the view of the SLB algorithm and system design optimization, the best result we expect is to get the maximum value *eb** of Ex-BL and the minimum value *vu* of V-UBL. Now let's take a look at these different SLB algorithms and approaches to this goal. The following analysis is based on experiments results and theory prediction.

R-ROBIN APPROACH: Schedule decisions are made based on the round-robin mode; it could be either one by one (taking turns), or a random number selection (this is random round-robin in our test). The method of taking turns is good for the case where we have a small number of real servers, the number of servers selected each time is almost equal to the number of real servers, all server machines have the same capacity, and all session traffic has almost the same payload. For instance, there are only four real servers (S1, S2, S3, and S4) with the same capacity, and in each session switching three servers are supposed to be chosen for traffic switching, so scheduling should be: (session1: S1, S2, S3), (session2: S4, S1, S2), (session3: S3, S4, S1), and so on. The server loads are not very badly balance. However, if there are 20 real servers (S1-S20) with greatly differing capacities, and session traffic payload is also highly varied (e.g., downloading a large movie or fetching a very small *ftp* file), then in this case, the scheduling method of taking turn will result into server load being very unbalanced. If we use the method of random number selection, it will not get good SLB points either, because it depends on chance; maybe sometimes having a very good result, sometimes a very bad one. Therefore, R-Robin is not a good approach for SLB.

Before discussing WRR, we assume, wt1=2, wt2=5, wt3=3, for three server machines named C, A, B, respectively; i.e, the total weight is 10 (totalWT=wt1+wt2+wt3). So, we get 10 instances of selection when running WRR once: 2C, 5A, 3B.

WRR APPROACH-1: Scheduling (or picking up) the best servers is based solely on their weights; in other words, the server with the highest weight definitely has the highest priority. So the selection order is: AAAAA BBB CC. This approach, of course, causes serious server load unbalancing because A is always served first, especially when the selected number of servers is less than five, and it will be more serious when the difference in weight values becomes larger. Our test results show: V-UBL (30%), Mod-BL (70%), and Ex-BL (0%). However, this these ineffective SLB points can be slightly improved by assigning appropriate weight values to different servers with different power capacities. For example, server C (wt2=5) has three times the capacity of server B (wt3=3), and two times that of server A (wt1=2). Unfortunately, my literature reviews indicate that the most of current server load balance products just simply use this approach once using the Weighted Round Robin algorithm because they assume that weights as the only important factor.

WRR APPROACH-2: Scheduling is based servers' weights, but the execution order is related to the difference in value of the weights. In this example, if wt1=2 (minWT), wt2=5 (maxWT), wt3=3 (nextWT), then t1 is the difference value between maxWT and nextWT, i.e., t1=wt2-wt3=5-3=2, and t2 is the difference value between nextWT and minWT, i.e, t2=wt3-wt1=3-2=1. Then the selection order should be: AA AB ABC ABC (t1 times A with highest weight, then t2 times AB taking turns, then ABC taking turns). Our experiment results show: V-UBL (10%), Mod-BL (40%), and Ex-BL (50%). This approach is relatively fair, improving SLB points siginficently because it gives other servers certain opportunities before the server with the highestly priority (A here) completes all of its turns (wt2=5).

WRR APPROACH-3: This approach is similar to WRR-2, but the execution order is slightly improved: AA BA CBA CBA. Our experiment results show: V-UBL (10%), Mid-BL (30%), and Ex-BL (60%). This approach is more effective, further improving SLB points because it provides more fairness to other servers with lower priority (servers B and C) before serving to A completely.

WRR Approach-4: Scheduling is based servers' weights, and the execution order is different than that of WRR-3, although it is still related to the difference of weights. We still have wt1=2, Wt2=5, wt3=3, t1=2, and t2=1, which are same as WRR-2. Now the execution order is: ABC ABC AB AA (wt1 times ABC taking turns each, then t2 times AB taking turns, then t1 times A). Our experimental results show: V-UBL (10%), Mod-BL (30%), and Ex-BL (60%). This approach seems to be more fair, and works effectively because it gives every server the same opportunities at first.

In short, WRR algorithm is a basic and classic SLB algorithm currently used by most of SLB products. It works well for server clusters with equal capacity, and its efficiency to some degree depends on approaches being taken appropriately. However, the WRR algorithm is not smart enough to deal with dynamic network traffic. So, we next discuss other two dynamic algorithms.

Least Connection Algorithm: Selecting the best server for the next connection is based on the fewest currently active connections, so counting, recording, and updating the current connection number is required. This is a dynamic algorithm because a network connection is based on more than a single parameter (related to TCP wait-time, time-out, server power processor, etc.). Indeed, any of the load balancing algorithms can be augmented with a minimum-connections option, where a maximum number of connections into a server can be configured. If this threshold is reached, no more connections will be made to that server until the number of active connections drops below the threshold.

In practice, it is probably easiest and best to use the Least Connection algorithm. By its nature, this approach, when used in session switching, tends to direct more connections to the servers with the highest capacity, because these servers try to set up, provide services and tear down connections more quickly than other servers.

Weighted Least Connection Algorithm: Selecting the best server for the next connection is based on the minimum value of (Ci/Wi). This is a dynamic algorithm, and in my view is the most complicate algorithm for current SLB products, although [Luce20] states: "Bell Labs's own patent-pending Weights TCP Redirected Method" is used for the first really pure Layer-7 switching. Actually, even the WLC algorithm has yet to be used by any of SLB products currently available.

To obtain the improved performance of the WLC algorithm, it is necessary and very important to monitor and analyze network connections, server weight and other relevant parameters. The system architecture presented in chapter 4 is designed for and beneficial to this purpose.

In summary, to get good server load balance, there are a number of heuristics, or server load balancing algorithms that can be used today. A key question is to how to determine the best, or the most available, server to which a connection request should be forwarded. This sort of driving load balancing decision, is not as easy as some papers describe; it actually depends on a variety of factors, such as server processing, server response time, CPU utilization, number of users, desired application availability, traffic payload, etc. What has been done for this research project is the design and implementation of one kind of network intelligent switching, and this chapter presents some experiments and analysis of available SLB algorithms. Hopefully, this work will be a useful reference for the creation of a new kind of advanced SLB algorithm and a new kind of system architecture using for the next generation of intelligent network switching.

Chapter 7

Conclusions & Recommendations

7.1. Conclusions

This project, supported by TR*Labs'* sponsors is application-based in nature. This relative large project includes many different technologies and methodologies. My thesis work involves: (1) the study and understanding of these technologies; (2) the research and systematical presentation of these methodologies; (3) the design of a system architecture using advanced technologies and methodologies; (4) the development and implementation of a prototype in software; (5) system tests and experiments using statistical methods; (6) system analysis based on experimental results and theoretical methods.

- This thesis reviews the background and technologies, including intelligent gateway, network management, network QoS, and network switching. The concept of an intelligent gateway is given, which is a device sitting between WAN and LAN, or WAN and WAN, taking care of both inbound traffic and outbound traffic, and containing certain intelligent functionalities, such as network monitoring, security service, QoS services, and intelligent switching. There are two areas of network management: network monitoring and network control. The SNMP management model provides an effective way to monitor LAN devices, which can collect information for intelligent traffic management. Basically there are three kinds of QoS service models: Best-Effort, IntServ, and DiffServ; the current research targets DiffServ over an IP network. Different techniques are provided for QoS, including multiple queuing methods for congestion management (such as FIFO, PQ, CQ, and WFQ), random early detection using WRED for congestion avoidance on outbound traffic, traffic shaping (or traffic limiting) for controlling bandwidth, and traffic coloring for intelligent packet classification.

- This thesis reviews network switching technology. The origins of network switching goes back about one hundred years ago, and today's data link layer (Layer-2) switching uses two technologies: the digital circuit switching used in most voice networks, and the digital packet switching used mostly in routing data and Internet traffic through packet networks. The evolution of packet networks includes X.25 networks, frame relay networks, cell relay networks, ISDN networks, ATM networks, and now ATM combining SONET. Network core switching includes the circuit-switched networks that is linked by FDM and/or TDM implementation, and the packet-switched networks that is classified into the VCs and datagram networks. The VCs is a kind of connection-oriented switching which needs to set up a path between the source and destination hosts and maintain connection states. The datagram networks is an IP address based routing switch, i.e., each packet switch, being a router that has a routing table mapping destination addresses (or portions of destination addresses) to an outbound link, extracts the header of incoming IP packets and examines the address and indexes of the routing table to find the appropriate outbound link, then send the packets into this outbound link. This sort of router switching in the network layer (Layer-3) is actually implemented based on the best-effort QoS model.

- This thesis reviews Layer-3/4/7 switching technologies, and provides different intelligent switching methodologies based on the TCP/IP reference model. The layer-3 switching methodologies include the classic switching, NetFlow switching, CEF switching and Tag switching. Tag switching is an implementation of the IETF standard for multi-protocol label switching (MPLS). MPLS currently is being used more and more for QoS over IP networks and optical networks more and more by both large industry-leading companies and small start-up companies now. Layer-4 switching, also called session switching, is designed for network system scalability, high performance, high availability and fault tolerance; the technical methods for layer-4 switching design are based on server clustering, that is, using server clusters, server load balancing algorithms (SLB), and load sharing network address translation (LSNAT). Layer-7 switching, or content switching, can provide more intelligent application services; there are two advanced approaches for application-layer switching: transparent switching of HTTP traffic to a proxy cache, and URL-aware switching.

- This thesis designs an architecture of intelligent switching system featuring dynamic network traffic management. This completely designed system consists of five modules: network monitoring, data analysis, switching decision, feedback information control, and server clusters. The system functionalities include: SNMP monitoring, system logs monitoring, network performance analysis (availability, response time, accuracy, throughput, and utilization), intelligent switching decisions (SLB algorithms), feedback information control based on connection quality and Cisco Dynamic Feedback Protocol (CDFP). The approach of the system design is intended to achieve increasing scalability, higher performance, higher availability, and disaster recovery. The advantage of the system architecture is that the core module of switching decision takes full advantage of system functionalities and feedback information approaches. SNMP monitoring provides MIB-II data collection which presents system information of a LAN device and inbound and outbound dynamic traffic, performance analysis provides network optimization information, connection quality and CDFP targets QoS, and SLB algorithm achieves server cluster load balancing. Therefore, this sort of dynamic network traffic network management meets the requirements of today's networks and the Internet which respond to increased traffic load, new application demands, and increasing network capabilities and services.

- This thesis presents system implementation concepts, including the Linux implementation of a virtual load server cluster (VLSC), and an application example of the SmartSwitch Router. A three-tier architecture of Linux VLSC has scalability, higher availability and performance, and distributed fault-tolerance. There are three kinds of strategies to implement VLSC over IP networks: VLSC/NAT, VLSC/TUN and VLSC/DR. Comparing these strategies shows different advantages and disadvantages: the VLSC/NAT approach is compatible with the TCP/IP protocol stack, but its scalability is limited; the VLSC/TUN approach is to avoid a bottleneck system, so its scalability is increased, but it requires servers that support the IP tunnel protocol; likewise, the VLSC/DR approach has better scalability, however there are some specific network configurations required. Realistic implementation strategies could be chosen based on system requirements, financial resources, and available network resources.

- This thesis provides the prototype and implementation of the SmartSwitch Router. Based on the concept of the SmartSwitch Router and software development design using TDD, ADTs, and OOD/OOP, one kind of TCP intelligent traffic redirector is created. The prototype contains different functions for achieving QoS, and it is implemented in the Java programming language. For this distributed system, different server/client programs are developed and launched on various machines as client, switch and server cluster. System tests and experiments are performed on Unix (Solaris 5.7) Workstations at Electrical and Computer Engineering, University of Manitoba; the results show that the prototype design is reasonable and the programming implemation is appropriate.

- This thesis provides system analysis based on experiment results. The comparison of the results of the SLB algorithms performance is given in graphical manner. Considering three kinds of SLB points (V-UBL, Mod-BL, and Ex-BL), for five SLB algorithms approaches (R-Robin, W-RR-1, W-RR-2, W-RR-3, and W-RR-4), W-RR-2 obviously improves Ex-BL, and W-RR-3 and W-RR-4 are the best approaches. The Least Connection and Weighted Least Connection algorithms are two kinds of dynamic algorithms that are more complicated but they are also very adaptive for use in dynamic traffic management. There are still large amounts lots of potential research and development work that could be done in order to get these dynamic algorithms to more work effectively in practice.

7.2. Recommendations for Future Works

Whole system architecture includes many different functions. Because of time limitation, I have only implemented some parts of these functions, especially these related to intelligent switching using SLB algorithms; because of thesis size limitation and the nature of this project, I have restricted my research work to a specific range. However, we definitely can take full advantage of these system functions and extensions for future works in both research and implementation approaches. My suggestions are as follows:

- As far as the network management module is concerned, if interested in SNMP monitoring for specific traffic such as TCP, UPD, and/or IP, then a *tcpDatabase*, *updDatabase*, and/or *IPDa-*

tabase could be built to store different statistic and/or dynamic data. A *RealTrafficMIB* could also be created to monitor real-time traffic. Alternately, network management could be extended to use RMON, SNMP v2 or SNMP v3. If one is interested in security monitoring or auditing, some of the approaches and tools provided in chapter 4 could be beneficial for complicated implementation of sysLog monitoring.

- In my thesis, the analysis of network availability and response time is discussed in detail, including some typical examples of telecommunication networks. There are other three kinds of network performance indicators, i.e., accuracy, throughput, and utilization, which are also important and interesting. There some potential for further research work, for example, the use of queuing theory to analyze network utilization.
- As far as feedback information module is concerned, it is a good way to try to implement the Cisco Dynamic Feedback Protocol, if the network and financial resources are available to buy and install Cisco equipment.
- Further research work and system implementation of the TrafficDirector included in the Linux LVSC architecture could be done by using modifying the TCP/IP stack inside Linux kernel.
- Further research work on the cluster management of Linux LVSC, such as the client-side approach, server-side Round-Robin DNS approach, server-side application-level scheduling approach, and server-side IP-level scheduling approach could be done.
- The implementation of the TCP intelligent traffic redirector could be done in the C++ programming language to a certain degree.
- Research into the development of a new kind of advanced SLB algorithm could be done, such as, for example, Nonlinear Weighted Least Connection algorithm, which could be achieved using a global nonlinear optimization approach.
- Research work into industry-leading technology to look for a sort of combination solution to optimize intelligent switching performance could be done, i.e., a combination of software and hardware approaches. This is a real challenge because this sort of product must have all the advantages of both Lucent Layer 4/7 switching using NEPPI (Network Element for Programmable Packet Injection), a software infrastructure developed by Bell Labs, and Web switching using the new WebIC (L2-7 network processor ASICs (Application-Specific Integrated Circuit)) architecture included in Nortel Networks' Alteon Series 700. Nevertheless, we are expecting the next generation of intelligent switching will be born soon.

References

[ACEd20] "ACEdirector", http://www.alteon.com.

[Alte20] Alteon Web Systems, "WEBWORKING Networking with the Web in mind", White paper, http://www.alteon.com, 2000.

[Arro20] "The Content Smart Internet", http://www.arrowpoint.com/solutions/whitepapers/CSI.asp.

[Bell20] http://www.bell-labs.com

[BeLc20] Bell Laboratories, Lucent Technologies, "Layer4/7 Switching and Other Custom IP Traffic Processing using the NEPPI API", 2000.

[CAID99] National Science Foundation and Cisco Systems, the cooperative as so cation for Internet data analysis, "CAIDA Measurement Tools Taxonomy", Summary paper, http://www.caida.org., 1999.

[CaFl20] "High-Performance Web Caching White Paper", http://www.cacheflow.com/technology/wp/hp_cache.html.

[CDFP20] Cisco Systems, "The Cisco Dynamic Feedback Protocol (Executive Summary)", 2000, http://www.cisco.com.

[CiDF20] Cisco Systems, "Cisco Dynamic Feedback Protocol---Platform Computing's SiteAssure", Solutions Brochure, http://www.cisco.com/warp/public/cc/so/cuso/sp/webhost/platf_cp.html, 2000.

[CiEF20] Cisco Systems, "Cisco Express Forwarding Switching".

[CiLD20] "Cisco LocalDirector", http://www.cisco.com.

[Cisc20] http://www.cisco.com.

[CoRH99.1] Ariel Cohen, Sampath Rangarajan, and Hamilton Slye: "On the Performance of TCP Splicing for URL-aware Redirection". Presented in the Proceedings of the USENIX Symposium on Internet Technologies and System, October 1999.

[CoRS99.2] Cohen, A., Rangarajan, and S., Siingh, N.: Support Transparent Caching with Standard Proxy Cache. In: Proceedings of the 4th International Web Caching Workshop (1999).

[Danz98] Danzi, P.: NetCache Architecture and Deployment. http://www.networkappliance.com/technology/level3/3029.html. Presented at the 3rd International WWW caching Workshop (1998).

[EgFr94] Egevang, K. and Francis, P., "The IP Network Address Translator (NAT)", RFC 1631, May 1994.

[Foun20] "ServerIron Switch", http://ww.founfrynet.com/cerviceironspec.html.

[GrSp96] Simson Garfinkel and Gene Spafford, Practical UNIX & Internet Security, 2nd Edition, O'Reilly & Associates, Inc., 1996.

[GuRM20] R.W. Gu, J.A. Rueda and R.D. McLeod, "Design of a Dynamic Network Traffic Allocation Gateway", TR*Labs* Research & Development Project Proposal, No.20DN17.

[GuMR20] R.W. Gu, R.D. McLeod and J.A. Rueda, "Dynamic Network traffic Allocation Gateway/ Switching", TR*Labs* TechForum 2000 in Saskatoon, Saskatchewan, Canada, October 2000.

[GuMF20] R.W. Gu, R.D. McLeod, and K. Ferens, "Web-based Network Management System: An Approach with SNMP/CORBA/Database and Traffic Analysis", TR*Labs* research project proposal, Department of Electrical and Computer Engineering, University of Manitoba, January 2000.

[GuRW99] R.W. Gu, "Design & Technology of Component-based E-Commerce", research project report on course Distribute Database Systems, Department of Electrical and Computer Engineering Department, University of Manitoba, December 1999.

[GuRW20] R.W. Gu, "Literature Review on Layer 4/7 Intelligent Switching", TR*Labs* research project report, Department of Electrical and Computer Engineering, University of Manitoba, October 2000.

[Haro96] Elliotte Rusty Harold, JAVA Networking Programming, O'Reilly, 1996.

[HuTc99] Hughes Technology, "User Manual for mSQL 2.0.11", 2000, http://www.huges.com.au.

[InCA99] "Internet Commerce Appliances", http://www.ipivot.com.

[Inkt20] "Deploying Transparent Caching with Inktomi's Traffic Server", http://www.inktomi-com/products/traffic/tech/deploy.html.

[JaSo99] "The Java Language: An Overview", http://java.sun.com/docs/overviews/java/java-overview-1.html.

[KuRo20] James F. Kurose and Keith W. Rose, Computer Networking: A Top-Down Approach Featuring the Internet, 2nd Edition, the Addision-Wesley WWW site, 2000.

[LiJi99] Jizong Li, "Web-based Network Management using SNMP and CORBA", Master thesis, Electrical and Computer Engineering Department, University of Manitoba, August 1999.

[Luce20] http://www.lucent.com.

[MaBh98] Maltz, D.A. and Bhagwat, P.: TCP Splicing for Application Layer Proxy Performance. IBM Research Report RC 21139 (1998).

[NoAl20] Nortel Networks Alteon WebSystems, "Scaling Next Generation Web Infrastructure with Contend-Intelligent Switching", White paper, 2000, http://www.nortelnetworks.com.

[OrHa98] R. Orfali and D. Harkey, "Client/Server Programming with JAVA and CORBA", Second Edition, John Wiley & Sons, Inc., 1998.

[PosU80] Postel, J., "User Datagram Protocol (UDP)", STD 6, RFC 768, August 1980.

[PosI80] Postel, J., "Internet Control Message Protocol (ICMP) Specification", STD 5, RFC 792, September 1981.

[Post81] Postel, J., "Transmission Control Protocol (TCP)", STD 7, RFC 793, September 1981.

[Post95] Postel, J., and J. Reynolds, "File Transfer Protocol (FTP)", STD 9, RFC 959, October 1985.

[RePo94] J. Reynolds and J. Postel, "Assigned Numbers", STD 2, RFC 1700, October 1994.

[RFCs] RFC web site, http://www.faqs.org/rfcs/.

[ScGR20] T. Scroeder, S. Goddard, and B. Rammamurthy, "Scalable Web Server Clustering Technologies", IEEE Network, pp. 38-45, May/June 2000.

[SrGa98] P. Srisuresh and D. Gan, "Load Sharing using IP network Address Translation (LSNAT)", RFC 2391, August 1998.

[Stal93] William Stallings, “SNMP, SNMPv2, and CMIP: The Practical Guide to Network Management Standards”, Addison Wesley Publish Company, 1993.

[Stll97] Stallings, William, Data and Computer Communications, Fifth Edition, Prentice-Hall Inc., 1997.

[Swatch] Stanford University, anonymous FTP web site *ftp://sierra.stanford.edu/swatch* or *ftp://coast.cs.purdue.edu/pub/tools/swatch/*.

[Tane96] Tanenbaum, Andrew S., Computer Networks, Third Edition, Prentice Hall PTR, Prentice-Hall Inc., 1996.

[Tris95] Brisco, T., “DNS Support for Load Balancing”, RFC 1794, April 1995.

[ViBr20] http://www.inprise.com/visibroker,1999.

[WeKa96] Matt Welsh and Lar Kaufman, Running LINUX, 2nd Edition, O’Reilly & Associates, Inc., 1996.

[WaMi98] Walls and Mirrors, Data Abstraction and Problem Solving with C++, 2nd Edition, Addison Wesley, 1998.

Appendix A

Transport Protocols (TCP/UCP) Header Formats

This appendix provides TCP and UDP segment header formats.

Figure A.1 and A.2 illustrate the Transmission Control Protocol (TCP) segment header format [Post81] and User Datagram Protocol (UDP) segment header format [PosU80], respectively.

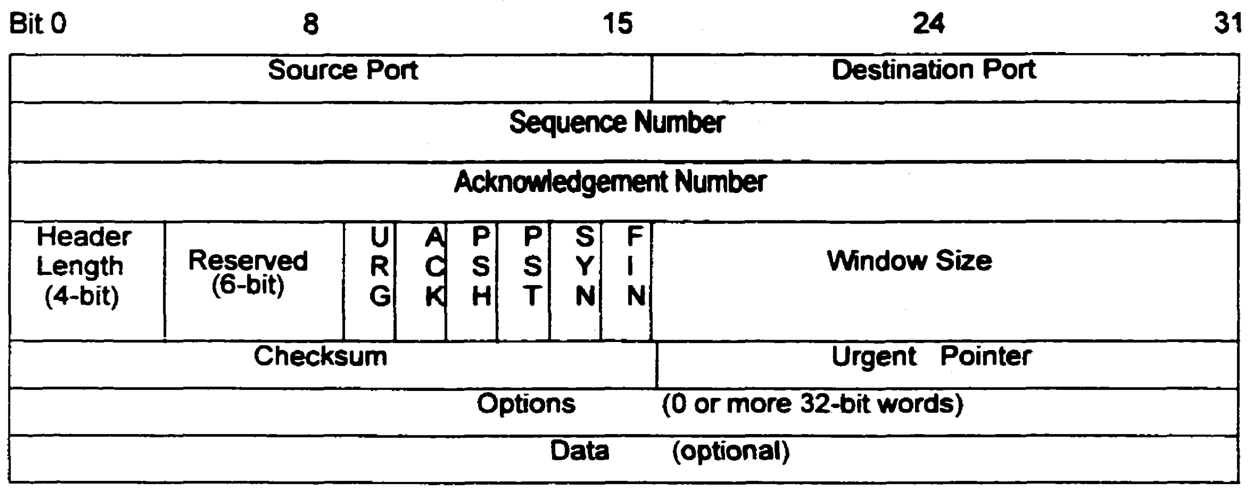

Fig. A.1. TCP segment header format.

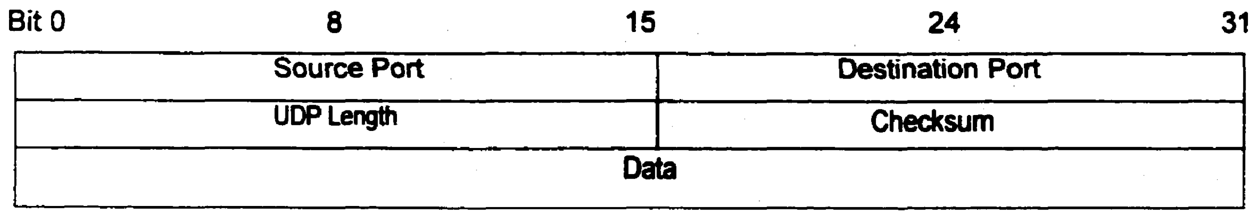

Fig. A.2. UDP segment header format.

Some points in Fig. A.1. and Fig. A.2. are explained as follows.

- *Source and Destination ports: (32 bits)* the application port from which the TCP (or UDP) segment is sent (source) and which it is sent to (destination).
- *Sequence Number: (32 bits)* the sequence number of the first data octet in this segment (except when SYN is present). If SYN is present, the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.
- *Acknowledgment Number: (32 bits)* If the ACK control bit is set, this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established, this is always sent.
- *Header Length (or Data Offset): (4 bits)* the number of 32 bit words in the TCP Header. This indicates where the data begins. The TCP header (even one including options) is an integral number of 32 bits in length; similar for *UDP Length: (16 bits)*.
- *Reserved: (6 bits)* reserved for future use. Must be zero.
- CONTROL BITS: 6 bits (from left to right):

  URG: urgent pointer field significant.

  ACK: acknowledgment field significant.

  PSH: pushes data, meaning the sender buffer was flushed to speed up transmission.

  RST: reset the connection, used after crashes.

  SYN: synchronize sequence numbers, used to establish connections.

    (SNY = 1, Ack = 0) Connection request.

    (SYN = 1, Ack = 1) Connection accepted - one half of full duplex.

    (SYN = 0, Ack = 1) Connection accepted - second half.

  FIN: no more data from sender, release the connection.
- *Window Size: (16 bits)* The number of data octets, beginning with the one indicated in the acknowledgement field, which the sender of this segment is willing to accept.
- *Checksum: (16 bits)* The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header and text, using for error detection.
- *Urgent Pointer: (16 bits)* This field communicates the current value of the urgent pointer as a positive offset from the sequence number in this segment. The urgent pointer points to the

sequence number of the octet following the urgent data. This field is only be interpreted in segments with the URG control bit set.

- *Options: (variable)* Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. All options are included in the checksum. An option may begin at any octet boundary. The content of the header beyond the End-of-Option option must be header padding (i.e., zero). A TCP must implement all options.

Appendix B

The Structure of SNMP MIB-II

This appendix provides the structure of SNMP MIB-II (RFC 1213) [RFCs].

Each information resource to be managed in the SNMP model is represented by an object and the Management Information Base (MIB) that is a structured collection of such objects. In the MIB structure, each type of object in an MIB is associated with an identifier of the ASN.1 type expressed by OBJECT IDENTIFIER (OID). The identifier is used to name the object and the structure of object types. The set of defined objects has a tree structure, shown in Fig. B.1.

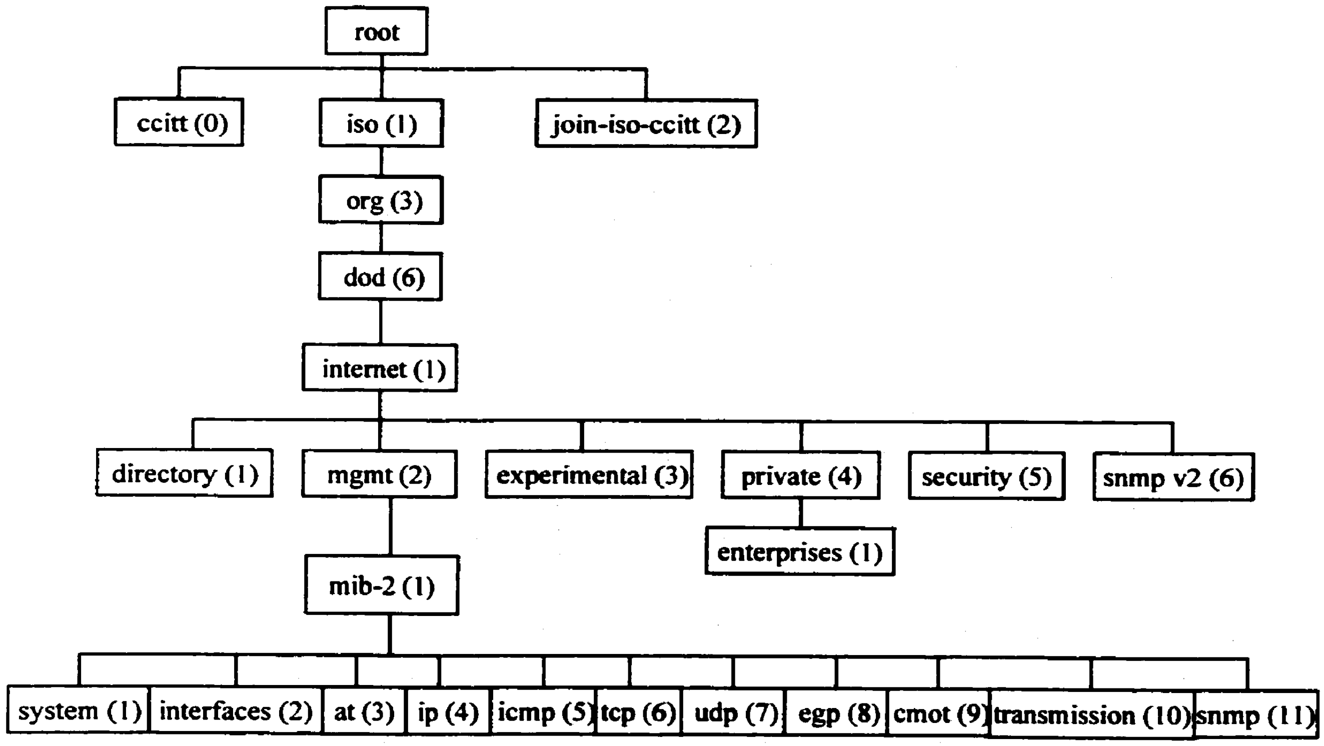

Fig. B.1. The Structure of SNMP MIB and MIB-II object groups.

In Figure B.3, starting from the root of the tree structure, there are three nodes at the first level: *ccitt* (0), *iso* (1), and *joint-iso-ccitt* (2). The *ccitt* stands for the International Consultive Committee on Telegraphy and Telephone, *iso* for International Organization for Standardization, and *joint-iso-ccitt* for jointly administrated by the IOS and CCITT.

Under the *iso* node, there is one subtree used for other organizations (org), labelled *org* (3), and one subtree of the *org* node used for the U.S. Department of Defense (dod), labelled *dod* (6). RCF1155 [RFCs] appoints that one subtree under *dod* will be allocated for administration by the Internet Activities Board (IAB) as follows:

```
internet OBJECT IDENTIFIER ::= { iso org(3) dod(6) 1 }
```

Thus, the *internet* node in Fig. B.3 has the object identifier value of 1.3.6.1. This value is used as the prefix for the nodes at the next lower level of the tree.

The subtree *mgmt* (2) under the *internet* node contains the objects defined in IAB-approved documnets. Further, the derived child of *mgmt* node is *mib-2* (1), an extension of the first MIB version *mib-1*, referred to as called MIB-II and defined in RFC1213 [RFCs]. The Object ID for *mib-2* is (*1.3.6.1.2.1*).

The structure of management information (SMI) document also defines other nodes (subtrees) under the *internet* node: *directory (1)* is reserved for future use with the OSI directory (X.500), *experimental (3)* is used to identify objects used in Internet experiments, and *private (4)* is to identify objects defined unilaterally, and so on.

Appendix C

Monitoring Data Collection Based on SNMP MIB-II

This appendix provides a table that lists some aspects of SNMP monitoring data collection based on MIB-II, as discussed in chapter 4.

Table C.1. SNMP monitoring data collection based on MIB-II.

Object Group	Group Node	Access & Status	Description
system *(1.3.6.1.2.1.1.)*	sysServices (~1.7)	RO M	This object has a value that is interpreted as a 7-bit code, and it represents the relationship that a system offers a service at a particular layer.
interface *(1.3.6.1.2.1.2.)*	ifSpeed (.5)	RO M	An estimate of the interface's current data-rate capacity.
	ifPhysAddress (.6)	RO M	The interface's address at the protocol layer immediately below the network layer.
	ifOperStatus (.8)	RO M	Current operational interface state: up(1), down(2), testing(3).
	ifInOctets (.10)	RO M	Total number of octets received on the interface, including framing characters.
	ifInDiscards (.13)	RO M	Number of inbound packets discarded even though no errors had been detected to prevent their being deliverable to a higher-layer protocol (e.g., buffer overflow).
	ifOutOctets (.16)	RO M	Total number of octets transmitted on the interface, including framing characters.
interface *(1.3.6.1.2.1.2.)*	ifOutDiscards (.19)	RO M	Number of outbound packet discarded even though no errors had been detected to prevent their being ransmitted (e.g., buffer overflow).

Object Group	Group Node	Access & Status	Description
at *(1.3.6.1.2.1.3.)*	ifOutQLen (.21)	RO M	Length of the output packet queue.
	atTable (.1.)	NA D	Contains the NetAddress to the equivlent physical address.
	atEntry (.1.1.)	NA D	Each entry contains one NetAddress to physical address equivalence.
	atIfIndex (.1.1.1)	RW D	Interface on which this entry is effective.
	atPhysAddress (.1.1.2)	RW D	Media-dependent physical address.
	atNetAddress (.1.1.2)	RW D	the NetAddress (e.g., IP address) corresponding to the media-dependent physical address.
ip *(1.3.6.1.2.1.4.)*	ipAddrTable (.20.1.~)		Contains information relevant to the IP addresses assigned to this entity, with one row for each IP address.
	ipRouteTable (.21.1.~)		Contains information used for internet routing.
	ipNetToMediaT-able (.22.1.~)		An address-translation table that provides a correspon-dence between physical addresses and IP addresses.
tcp *(1.3.6.1.2.1.6.)*	tcpRtoAlgorithm (.1)	RO M	retransmission time: other(1), constant(2), MIL-STD-1778(3), Van Jacobson's algorithm(4).
	tcpRtoMin (.2)	RO M	Minimum value for the retransmission timer.
	tcpRtoMax (.3)	RO M	Maximum value for the retransmission timer.
	tcpMaxConn (.4)	RO M	Limit on local number of TCP connections the entity can support.
	tcpActiveOpens (.5)	RO M	Number of active opens that have been supported by this entity.
	tcpPassiveOpens (.6)	RO M	Number of passive opens that have been supported by this entity.
	tcpAttemptFails (.7)	RO M	Number of failed connection attempts that have occurred at this entity.
	tcpEstabResets (.8)	RO M	Number of resets that have occurred at this entity.

Object Group	Group Node	Access & Status	Description
tcp *(1.3.6.1.2.1.6.)*	tcpCurrEstab (.9)	RO M	Number of TCP connections for which the current state is either ESTABLISHED or CLOSED-WAIT.
	tcpInSegs (.10)	RO M	Total number of segments received, including those received in error.
	tcpOutSegs (.11)		Total number of segments sent, excluding those containing only retransmitted octets.
	tcpRetransSeg (.12)	RO M	Total number of retransmitted segments.
	tcpConnTable (.13.~)	NA M	Contains TCP connection-specific information.
	tcpConnEntry (.13.1.~)	NA M	Information about a particular current TCP connection.
	tcpConnState (.13.1.1)	RW M	Close(1), listen(2), synSent(3), synReceived(4), established(5), finWait1(6), finWait2(7), closeWait(8), lastAck(9), closing(10), timeWait(11), deleteTCB(12).
	tcpConnLocalAddress (.13.1.2)	RO M	Local IP address for this connection.
	tcpConnLocalPort (.13.1.3)	RO M	Local port number for this connection.
	tcpConnRemAddress (.13.1.4)	RO M	Remote IP address for this connection.
	tcpConnRemPort (.13.1.5)	RO M	Remote port number for this connection.
	tcpInErrs (.14)	RO M	Total number of segments received in error.
	tcpOutRets (.15)	RO M	Number of TCP segments sent containing the RST flag.

Notes: The access characteristic of an object may be read-only (RO), read-write (RW), write-only (WO), or not-accessible (NA). The status of an object may be mandatory (M) or deprecated (D).

The remaining groups of icmp, udp, egp, transmission and snmp are omitted here due to limited space.

Appendix D

System Logs Monitoring Information

This appendix provides some useful information on system log monitoring as discussed in Chapter 4, including Tables D.1-D.6, and some script examples.

Table D.1. The basic Log File on Unix / Linux

File name	Function
acct or pacct	Records commands run by every user.
aculog	Provides records of dial-out modems (automatic call units).
lastlog	Logs each user's most recent successful login time, and possibly the last unsuccessful login too.
loginlog	Records bad login attempts
messages	Records output to the system's "console" and other messages generated from the *syslog* facility.
sulog	Logs use of the *su* command.
utmp[1]	Records each user currently logged in
utmpx	Extended *utmp*.
wtmp[2]	Provides a permanent record of each time a user logs in and logs out, also records system shutdowns and startups.
wtmpx	Extended *wtmp*.
vold.log	Logs errors encountered with the use of external media, such as floppy disks or CD-ROMs.
xferlog	Logs FTP access.

Table D.2. *syslog* Facilities.

Program	Facilities
kern	Kernel
user	Regular user processes
mail	Mail system
lpr	Line printer system
auth	Authorization system, or programs that ask for user names and passwords (*login, su, getty, ftpd,* etc.).
daemon	Other system daemons
news	News subsystem
uucp	UUCP subsystem
local0...local7	Reserved for site-specified use
mark	A timestamp facility that sends out a message every 20 minutes

Table D.3. syslog Priorities.

Priority	Meaning
emerg	Emergency condition, such as an imminent system crash, usually broadcast to all users.
alert	Condition that should be corrected immediately, such as a corrupted system database.
crit	Critical condition, such as a hardware error.
err	Ordinary error.
warning	Warning.
notice	Condition that is not an error, but possibly should be handled in a special way.
info	Informational message.
debug	Messages that are used when debugging programs.
none	Do not send messages from the indicated facility to the selected file.

Table D.4. syslog Log message sources.

Source	Meaning
/dev/klog	Special device, used to read messages generated by the kernel.
/dev/log	Unix domain socket, used to read messages generated by processes running on the local machine.
UDP port 514	Internet domain socket, used to read messages generated over the local area network from other machines.

Table D.5. Typical critical messages of *syslog*.

Program	Message	Meaning
halt	halted by <user>.	<user> used the */etc/halt* command to shut down the system.
login	ROOT LOGIN REFUSED ON <tty> [FROM <hostname>].	*root* tried to log onto a terminal that is not secure.
login	REPEATED LOGIN FAILURES ON <tty> [FROM <hostname>] <user>.	Somebody tried to log in as <user> and supplied a bad passwords more than five times.
reboot	rebooted by <user>.	<user> rebooted the system with the */etc/reboot* command.
su	BAD SU <user> on <tty>.	Somebody tried to *su* to superuser and did not supply the correct password.
shutdown	reboot, halt, or shutdown by <user> on <tty>.	<user> used the */etc/shutdown* command to reboot, halt, or shut down the system.

Table D.6. Typical Info messages of *syslog*.

Program	Message	Meaning
date	date set by <user>.	<user> changed the system date.
login	ROOT LOGIN <tty> [FROM <hostname>]	root logged in.
su	<user> on <tty>.	<user> used the *su* command to become the superuser.
getty	<tty>	/bin/getty was unable to open <tty>.

An example of reading a lastlog File:

The following developed program written in Perl script works on SunOS systems.

```
#!/usr/local/bin/perl

$fname = (shift || "/var/adm/lastlog");

$halfyear = "60*60*24*365.2425/2;  # pedantry abounds

setpwent;

while (($name, $junk, #uid) = getpwent) {

     $names{$iud} = $name;

}

endpwent;

open(LASTL,  $fname);

for ($uid = 0;  read(LASTL, $record, 28);  $uid++) {

   ($time, $line, $host) =  unpack('l A8 A16', $record);

    next unless $time;

    $host = "(host)"  if host;

    ($sec,  $min,  $hour,  $mday,  $mon,  $yaer)  =  localtime($time);

    $year +=1900 + ($yaer < 70 ? 100 : 0);

    print  $names{$uid},  $line, "$mday/$mon";

    print (time - $time > $halfyear) ? "/$year" : "  "$hour:$min";

    print "    $host\n";

}

close LASTL;
```

A typical example of *syslog.conf* file used for the Unix syslog facility is listed here.

```
*.err; kern.debug;auth.notice   /dev/console

daemon,auth.notice              /var/adm/messages

lpr.*                           /var/adm/lpr-errs

auth.*                          root, nosmis

auth.*                          @maggie.win.trlabs.ca

*.emerg                         *

*.alert                         |dectalker

mark.*                          /dev/console
```

Appendix E

Cisco DFP Defined Messages and Vectors

This appendix provides the defined messages and vector names used by the Cisco DFP system flow as discussed in chapter 4.

The Cisco DFP system flow defines the following MESSAGES between the workload agent and the load-balancing manager:

- The *preference information message* is used to report the status or weight of an IP server and is sent from the workload agent to the load-balancing manager.
- The *server state message* is used to inform the workload agent that the load-balancing manager has decided to take the server in or out of service and is sent from the load-balancing manager to the workload agent.
- The *DFP parameters* are used to send the configuration information from the workload agent to the load-balancing manager. Currently the only configuration parameter passed is the keep-alive interval.

DFP message also uses vectors for defined messages. VECTORS are optional commands that can exist in the defined messages. The *vector header*, the first part of each vector in the DFP message, allows the receiver of the message to skip over any vectors or commands that are not understood by the workload agent. Each vector header contains a vector opcode and the overall length of the vector. The following vector names can be used in defined messages:

- The *load vector* contains the actual load information being reported for the real servers and it represents the servers' preferred capability. The real servers are first grouped by their port numbers and protocol types, and the individual servers are then listed with their individual weights. A Bind ID is included in the load vector to allow a single physical machine to be bound to multiple virtual machines that have been weighted differently.

- The *security vector* allows each DFP message to be verified. It is expandable to specify which security algorithm is to be used, so that it enables MD5 (Message Digests) and DES (Data Encryption Standard) to be supported today while enabling future security algorithms to be supported in a nondisruptive fashion.
- The *keep-alive vector* is the part of the DFP connection's configuration. The keep-alive vector allows the ioad-balancing manager to inform the DFP workload agent of the minimum time interval by which the workload agent must send information over the DFP connection.