DESIGN AND IMPLEMENTATION OF MULTIPLE ADDRESS PARALLEL TRANSMISSION ARCHITECTURE FOR STORAGE AREA NETWORK

MENG BIN
(M.Eng. , Xi'an Jiaotong Univ.)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2004

Acknowledgements

I am sincerely grateful to my supervisor Prof. Chong Tow Chong for giving me the privilege and honor to work with him over the last two years. Without Prof. Chong's constant support, insightful advice, excellent judgment, and, more importantly, his demand for top-quality research, this thesis would not be possible. I am continuously amazed and humbled by his infinite knowledge and unmatched wisdom.

I would also like to thank Yong Khai Leong and Patrick B. T. Khoo for introducing me to the subject of network storage, giving a necessary direction to my research, providing continuous encouragement throughout my M.Eng.

This work would not be possible without a long-lasting support and infinite patience of DSI MCSA group. I would like to thank Core Technology members, Don Lee, Alvin Koy, Wei Minglong, Li Zhixiang and Implementation members, Wilson Wang, Jimmy Jiang, Law Sie Yong, Ng Tiong King, Wang Haichen, Huang Xiaogang, Yeo Heng Ngi. Since a large part of my work was experimental, I am further indebted to MCSA Application team for their extreme generosity in providing the abundant resources needed for completing this M.Eng. Many thanks to Vincent Leo, Wang Donghong, Daniel Khoo, Premalathe Naidu and Han Binhua.

Furthermore, I would like to thank my friends and fellow students at the National University of Singapore. I am especially grateful to Wu Daowei, Lin Song, Shen Yatao and Chen Li for being absolutely awesome friends and treating me to frequent philosophical discussions.

I am thankful to 20th *IEEE Symposium on Mass Storage Systems* reviewers for providing their helpful comments on earlier versions of this work.

Last, but not least, I would like to thank my parents. Without their continuous support this work would be simply impossible.

Contents

Summary

The demand for high availability and high performance in Storage Area Network (SAN) drives more and more effort on network architecture design. To build such a Storage Area Network, the use of parallel channels is attractive. At present, most parallel solutions heavily depend on hardware components and do not provide end-to-end parallel connection. This thesis provides a theoretical model and an implementation demonstrating the multiple address parallel transmission architecture. This architecture provides end-to-end parallel connections between network storage devices without any addition of components.

At first, the theories of the network topology and parallel data transmission model are studied. This parallel data transmission architecture is based on packet switching network. SAN topology using this structure will have good scalability and fault tolerance but without much additional complexity. The model of this transmission architecture is composed of two stage load balancing algorithms, data flow priority algorithms, data flow restoring, reordering and fault tolerance algorithms.

Three load balancing algorithms, Reverse Weight Round Robin (RWRR), Reverse Deficit Round Robin (RDRR) and Reverse Weighted Fair Queuing (RWFQ), are proposed and implemented with small computation overhead. The mechanism of out-of-order in parallel architecture is analyzed in detail. An efficient out-of-order estimation algorithm is used to setup retransmission threshold and allocate reordering buffer. Furthermore, since this transmission architecture uses redundant channels, the scheduler can provide fault tolerance by using either hardware-based, software local loop back andor remote acknowledge detection methods.

To demonstrate the multiple address parallel transmission protocol (MAP-TP) in SAN, we implement the protocol as network layer in the Linux kernel module of the HyperSCSI and measure the performance by testing the disk read speed of the virtual SCSI device with three popular benchmark tools of HDPARM, DD and IOZONE.

The result shows that HyperSCSI bandwidth can be increased above 380% when 4 links are used, which is acceptable for high speed transmission. Interestingly, we also found that out-of-order rate does not affect the disk read performance very much, which can be explained by the theory of schedule delay and out-of-order distance.

In addition, by supporting Ethernet MAC and IP packets, the network storage devices can work in both LAN and WAN environments. This architecture can be a key factor in deploying SANs, which can provide a highly scalable bandwidth with full redundancy over switched and parallel data paths.

List of Figures

Chapter One

1 Introduction

1.1 Background and Motivations

A Storage Area Network (SAN) is a specialized, high-speed network attaching servers and storage devices. It is sometimes called “the network behind the servers”. A SAN allows “any to any” connection across the network, using interconnect elements such as routers, gateways, hubs and switches.

A Storage Area Network (SAN) is any high-performance network whose primary purpose is to enable storage devices to communicate with computer systems and with each other.

The demand for high availability and high performance in Storage Area Network (SAN) drives more and more effort on network architecture design. The SAN’s performance must be able to grow as the organization’s information storage and processing needs grow. A successful SAN should provide an extra performance for the high-volume data and message traffic of client and server. Thus, the scalability of SAN is quite important, because it is not practical to replace a SAN very often for enterprise applications. To build a scalable Storage Area Network, the use of parallel data path network architecture is attractive.

Current parallel network architectures can be classified into two categories: non packet switching based and packet switching based architectures. The non packet switching parallel network architectures emerged at 1980s’. First, two popular non packet switching architectures are demonstrated below.

One traditional research on parallel network architecture concentrates on Multi-channel Local Area Network (MLAN) architecture [1]. The MLAN is based on a bus sharing architecture (Figure 1.1). Although no middle gateways or switches are required, this bus sharing architecture has the disadvantage that only two devices can communicate at one time.

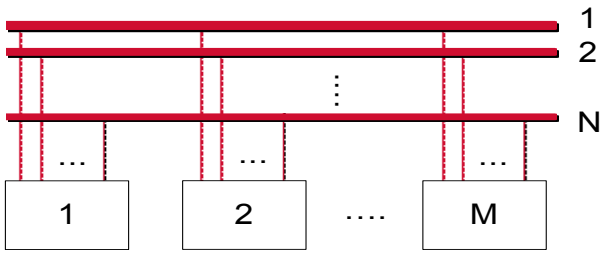

Figure 1.1 Traditional Multi-channel Local Area Network

The other architecture is Direct Attached Storage (DAS) network, where storage area network uses directly connected parallel SCSI network [2]. The current maximum data transmission performance of DAS is 200MB/s. However, faster speed means shorter distance, normally no more than 30 meters. Limited by the distance, DAS is not a good solution for parallel SAN architecture.

With the rapid evolution of the switching technology in 1990s', research in parallel network gradually focus on packet switching network (as showed in Figure 1.2). The algorithms which schedule data streams to parallel paths are also studied as inverse multiplexing [3]. The parallel packet switching network is more scalable and flexible than the MLAN and DAS. Additionally, two packet switched storage devices can communicate without affecting other devices' performance.

The parallel packet switching network architecture can be built on one particular link or end-to-end communication [4]. The aggregate bandwidth built on one

particular link is commonly used to increase local area network performance. These scheduling algorithms are relatively simple because of the symmetrical parallel architecture. However, it is highly dependent on switches to provide load balancing services. Most contemporary parallel implementations work on one particular link, for instance:

Link Aggregation Control Protocol (LACP) [5] is a sub layer for CSMA/CD MAC. Designed for IEEE 802.3 MAC, it can not adapt to the load balancing based on IP layer address; and LACP can not operate across multiple data rates, which means all network interface are restricted to the same transfer speed. Furthermore, LACP transfer one conversation on one physical network interface to avoid packets disorder, bandwidths on other interfaces are wasted.

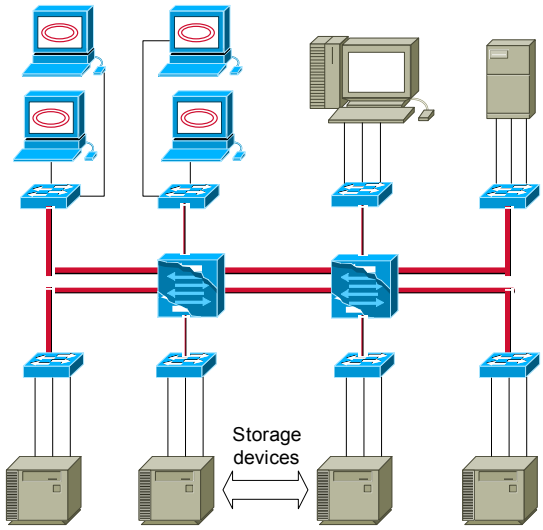

Figure 1.2 Asymmetrical Parallel Architecture in Packet Switching Network

Linux Bonding, Cisco EtherChannel and Sun Trunking [6]**:** All these technologies refer to using multiple network interfaces to compose a virtual link, which use one pair of MAC address. This technology is similar to the LACP except that it heavily relies on switch's ability to provide traffic load balancing (known as

virtual LAN technology). This will increases the cost and complexity of the SAN topology.

The parallel packet switching architecture based on end-to-end parallel transmission routes different packets belonging to a single end-to-end application along multiple distinct network paths [7]. This end-to-end parallel technology can provide a highly scalable bandwidth over long distances, and with the ability to provide full redundancy and parallel data paths. However, there are several significant challenges to end-to-end protocol, which must be responsible for managing the multiple flows and resequencing packets with large skew. Due to these challenges, there is no available end-to-end parallel protocol at present.

On the storage area network application side, many contemporary SAN solutions such as, HyperSCSI, FCIP, iSCSI, iFCP and iSNS [8], are working on packet switching network. For these solutions, parallel packet switching technology is in great need to meet the new requirements for high scalability and reliability over long distance. Recently, GridFTP claims that it provides end-to-end parallel TCP streams to improve performance [9,10]. However, it actually provides multiple data flows on the same path. This only increases the channel utilization not the physical channel bandwidth.

As we discussed here, the end-to-end parallel transmission architecture will be a key factor in SAN solutions. Thus, we start a study on multiple address parallel transmission protocol (MAP-TP) for SAN to meet the new requirement of high scalability and reliability over long distances. This protocol is also referred as “Multi-channel” technology throughout the thesis. The aim of the study is to design and implement an end-to-end parallel architecture that can increase the network I/O performance between network storage modules by employing more communication

channels in packet switching network. Moreover, a built-in fault tolerance strategy for surviving and restoring from network interface and network failures should be developed.

1.2 An Introduction to HyperSCSI Protocol

To demonstrate the multiple address parallel transmission protocol in SAN, MAP-TP protocol is developed and combined with the HyperSCSI network storage protocol. In this section, we will outline some of the key features and basic technical details of HyperSCSI. The architecture relationship between MAP-TP and HyperSCSI is also demonstrated.

Small Computer Systems Interface (SCSI) is the predominant mechanism for various storage and even non-storage devices. To make SCSI “network-enabled”, the HyperSCSI is invented for the transmission of SCSI family of protocols across a network.

Since the requirements of local network storage (SAN) and wide-area network storage (SWAN) are quite different, HyperSCSI protocol supports multiple modes of operation. Two such modes are currently being developed, one for local access, Local HyperSCSI over Ethernet (HS/eth), and the other for wide-area connectivity, Wide-Area HyperSCSI over IP (HS/IP). The basic protocol structure is essentially the same, thus allowing devices to speak local or wide-area storage seamlessly.

Many advanced functions and capabilities were built into the HyperSCSI protocol to support other requirements like dynamic management, dynamic flow control and in-band management capabilities. HyperSCSI can provide a minimum level of connectivity for interoperability operations and while supporting advanced vendor-specific or implementation-specific functions. Other possible device specific options include read-only access, removable media locking and data compression.

The HyperSCSI connection setup is a three-step handshaking procedure between a HyperSCSI client and server pair. Typically, in a storage network, the host machine (HyperSCSI client) is responsible for locating and initiating connections to storage devices (HyperSCSI servers). During this process, the HyperSCSI client issues a HCC_DEVICE_DISCOVERY via Ethernet broadcast or IP packet, to locate devices on the network. For IP-based situations, a client must specify an IP address (or DNS name) and a HCC_DEVICE_DISCOVERY packet is sent over IP directly to the server. Once the HyperSCSI server receives this packet, it checks the client address for authentication purposes and transmits the HCC_ADN_REQUEST packet back to the HyperSCSI client. In order for the HyperSCSI client to establish a connection with the HyperSCSI server, it must then send the correct response through a HCC_ADN_REPLY command and add the ID numbers of the devices that it has access to into its own registry. If the server successfully authenticates the HCC_ADN_REPLY, the connection is accepted and the HyperSCSI client can now send commands to the server.

An ACK mechanism has been adopted to support flow control of data between a HyperSCSI client and server pair. The ACK window size refers to the number of packets that the transmitter may continuously send before waiting for an acknowledgement. This window size must be negotiated and agreed upon before data flow can take place and is set by the requestor through an FC_ACK_SNR command. This packet is issued as a separate message and typically, the server will be the one to issue this command so that the server has the ability to balance loads or priorities across multiple clients, although this does not mean that the client may not issue one either. The ACK window size can be set based on traffic loads, or buffer capacities and can be set at start-up or changed dynamically during run time.

When there is a SCSI request from the local OS SCSI upper layer of the host machine, the HyperSCSI client software is responsible for converting the OS-specific SCSI command block together with any relevant data (as in a write command) into a platform independent HyperSCSI Command Block (HCB). The client then encapsulates and fragments the HCB into one or more HCBE_REQUEST packets that it sends to the HyperSCSI server. SCSI command blocks and user data will therefore be transmitted together in the same packet. The HyperSCSI server receives the data stream, re-assembles the HyperSCSI command block and relevant user data, converts it back to an OS-specific SCSI command block and passes it to the relevant hardware for execution. When the result of this SCSI request is ready, the HyperSCSI server will send the result together with the requested data back to HyperSCSI client by issuing the HCBE_REPLY packet stream in a similar manner as the request. The HyperSCSI client reassembles the HyperSCSI command block and converts it back to a OS-specific SCSI command block before passing it on to the local OS SCSI upper layer.

During a HyperSCSI connection, the HyperSCSI server will regularly (timer-based) issue a HCC_ADN_REQUEST command for three purposes, re-authentication of clients and key-exchange for security, re-negotiation of device options (if permitted), and as a form of “keep-alive”. Through this method, servers not only poll the client’s status, but also check its identity. Furthermore, if HyperSCSI encryption options are turned on for data transmission, the HCC_ADN_REQUEST and HCC_ADN_REPLY use an authenticated exchange mechanism to update and change encryption keys. This scheme also allows a device’s options to be modified dynamically.

The HyperSCSI client can close a connection by sending an HCC_DISCONNECT command to the HyperSCSI server. The server will then remove

this client from its connection list and close the connection. Servers do not need to acknowledge disconnect requests from clients because SCSI connections are host-target based.

Feature-wise, the HyperSCSI reference implementation already supports standard SCSI hard drives, IDE hard drives, software RAID / virtualized drives, optical disks (like DVDROM and CDRW), USB devices (like Iomega Zip Disk) and SCSI tape drives (like HP DAT40). We have even successfully used HyperSCSI to write CDs remotely over DSI live corporate LAN. File systems like Microsoft's FAT16/FAT32, SGI's XFS, IBM's JFS and Linux Ext2/Ext3 have all been successfully tested on HyperSCSI drives. HyperSCSI clients and servers have been successfully implemented on Linux, while client versions on Windows 2000 and Solaris 8 are currently in development. Encryption schemes that have already been implemented include 64-bit Blowfish and 128-bit Rijndael. HyperSCSI has been assigned its own IEEE Ethertype Number, and will soon receive a registered IP port for HS/IP implementations.

HyperSCSI provides an opportunity to address various concerns and open up new possibilities for network storage. The Local HS/eth protocol allows the construction of high-speed Ethernet based SANs while the use of Wide-Area HS/IP permits mobile devices like laptops to access the corporate SAN directly (bypassing servers if needed). However, storage devices that only use single network path are not efficient enough. In the HyperSCSI performance testing, it is found that the storage devices have low utilization rate when working on Fast Ethernet network. In this application, the network bandwidth becomes the bottleneck of the whole SAN system.

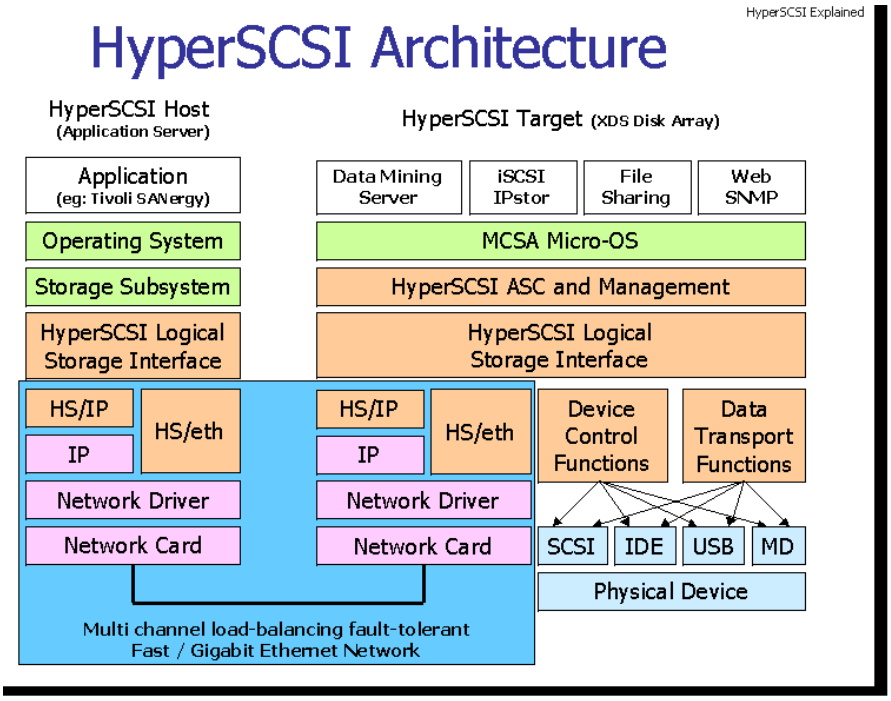

Figure 1.3 Multi-Channel Network Architecture for HyperSCSI Protocol

To improve the scalability and reliability of HyperSCSI, the MAP-TP protocol is combined with HyperSCSI protocol (Figure 1.3). Multiple network interfaces and network paths are used to enhance the performance and balance the traffic load. Furthermore, the packetisation and virtualization options of HyperSCSI allow us to implement Multi-channel communications with fault tolerance and reliability.

1.3 Contribution and Organization of Thesis

In this thesis, our research emphasis is on the end-to-end multiple address parallel transmission architecture for SAN application. This architecture provides not only scalability, but also the simplicity and reliability. The contributions of this thesis can be summarized as:

1. A theoretical multiple address parallel transmission model is studied. This model provides end-to-end parallel connections between network storage devices without extra hardware.

2. Three load balancing algorithms, Reverse Weight Round Robin (RWRR), Reverse Deficit Round Robin (RDRR) and Reverse Weighted Fair Queuing (RWFQ), are proposed and implemented with small computation overhead.

3. The mechanism of the out-of-order in parallel architecture is analyzed in detail. An efficient out-of-order estimation algorithm is deduced to setup retransmission threshold and allocate reordering buffer.

4. Fault tolerance methods of hardware-based, software local loop back and/or remote acknowledge are studied on parallel redundant channels.

5. The multiple address parallel transmission protocol (MAP-TP) is implemented as network layer in the Linux kernel module of the HyperSCSI. The performance is measured by testing the disk read speed of the virtual SCSI device with three popular benchmark tools of HDPARM, DD and IOZONE.

Organization of this thesis is as follows:

In Chapter 2, we will discuss the theoretical model of the end-to-end multiple address parallel transmission architecture. First the parallel network topology is studied, and then an asymmetrical parallel data transmission model is demonstrated. This transmission model is composed of two-stage load balancing algorithm, data flow priority algorithm, data flow restoring, reordering and fault tolerance algorithm. Three load balancing algorithms, Reverse Weight Round Robin (RWRR), Reverse Deficit Round Robin (RDRR) and Reverse Weighted Fair Queuing (RWFQ) are proposed. The mechanism of the out-of-order in parallel architecture is analyzed in detail. An efficient out-of-order estimation algorithm is used to setup retransmission threshold and allocate reordering buffer. Furthermore, since this transmission architecture uses

redundant channels, three fault detection methods of hardware-based, software local loop back and/or remote acknowledge are studied.

In Chapter 3, the multiple address parallel transmission protocol (MAP-TP) will be implemented as network layer in the Linux kernel module of the HyperSCSI. Three major modules: data module, control module and global state machine module are demonstrated. The data module is responsible for sending and receiving data flow packets through multiple network channels. The control module is used to manage multiple channels and selecting load balancing algorithm. The fault tolerance is also considered in the control module. The global state machine inspects global parameters and synchronizes the control and data transmission states transition. Additionally, detail functions of the modules are presented in software design flow charts.

In Chapter 4, we will measure the parallel transmission performance and out-of-order delivery by testing the disk read speed of the virtual SCSI device with three popular benchmark tools of HDPARM, DD and IOZONE. Different parallel symmetrical and asymmetrical architectures are studied under all three scheduling algorithms, RWRR, RDRR and RWFQ. All experiments are based on parameters of channel numbers, channel bandwidth, packet size and system resource. By discussing the experimental results, suggestions for MAP-TP design will be provided.

Chapter Two

2 Theoretical Models

Research on parallel resource modeling is emerging in recent years. Several papers have proposed theoretical models [11, 12, 13, 14, 15, 16, 17] for multiple resources scheduling in parallel architecture network. These theories are quite helpful for designing and evaluating the throughput, delay, and load-balancing algorithm of multiple address parallel transmission architecture. In this chapter we first discuss the theories of the network topology, and then demonstrate an asymmetrical parallel data transmission model in the following sections. This transmission model is composed of two-stage load balancing algorithm, data flow priority algorithm, data flow restoring, reordering and fault tolerance algorithm.

2.1 Network Topology - Ethernet and IP Packet Switching Network

In a SAN, the topology is arbitrary. The storage devices are expected to have variable bandwidths and packets can have variable sizes. To meet these requirements, we will design a model which can provide asymmetrical parallel channels between the sender and the receiver.

This multiple address parallel transmission architecture, which is called Multi-Channel, is designed to work on both symmetrical and asymmetrical packet switching network. In the switching network, storage devices are connected with links and switches, where links are assumed to have bound delay and switches are assumed to be “non-blocking”. With these topologies, a connection in such a network can be modeled as traversing a number of queuing servers, while each server modeling the output link of a switch. Each communication component of the network has a bounded delay.

Between a node and a switch, multiple links can be used and these links can have different bandwidth. A general structure is shown in Figure 1.2. Each storage device (one node) can have any number of network interface cards (NIC). Every network interface can access any other network interface. To build a connection between two end-to-end nodes, the number of the network interfaces of two nodes need not to be equal. For the implementation, two most popular protocols, Ethernet and IP protocol are chosen as the network channels sub layer. The NIC interfaces can be Fast Ethernet or Gigabit Ethernet.

Unlike other NIC aggregation solutions, this design does not demand for special hardware in storage devices or switches. Each NIC uses their original address naming system, i.e., each NIC card is bonded with unique MAC address or IP address. The parallel communication channels between two nodes are built up by scheduling network packets based on these addresses. Many other parallel transmission solutions assign the same address to multiple NIC interfaces, which cannot build an end-to-end parallel channel. This requires that both nodes and switches install special load balancing and address translating service, which is complicated to setup. Normally, the switch components must support VLAN function to support such parallel transmission. In our architecture, only the sender and receiver nodes schedule the data flows, the required network in between is a normal network.

So far we have a multiple paths architecture built by normal packet switching network. Next, multiple communication channels are built on these paths. Some designs use part of the communication channels as backup channels [18], which preserve spare resource in case of failure. This is actually a resource waste. All the available channels should be utilized on communication simultaneously. Additionally, some design keep one data flow on one channel [19] to avoid packet out-of-order, this

also does not fully utilize the resource. Because information is required to maintain a data flow on a particular channel, it may cost more effort than out-of-order handling. In our design, one data flow or multiple data flows are distributed on all channels. By employing multiple channels, not only load balancing and fault tolerance are provided, the security feature is also enhanced in some distance because of the information dispersal [20].

With the improvement of the computation power, traffic load scheduling can be implemented based on software [21]. The whole multiple address parallel transmission protocol can be implemented as a software layer, where the sender can do all the scheduling. Multiple data flows are dispersed into packets at sender, and then the traffic load is sent to channels evenly by schedulers.

2.2 Multiple Address Parallel Transmission

The communication model of Multi-Channel is different from a normal parallel transmission model. In SAN, network storage devices are expected to have variable capacity and bandwidth. At present, only the storage capacity has good scalability, while the bandwidth cannot be scaled easily. One purpose of Multi-Channel model is to efficiently increase the end-to-end bandwidth scalability of the network storage devices. To apply the algorithms more precisely to a real SAN environment, channel numbers are not necessarily symmetrical at sender and receiver nodes. For instance, if the storage server needs more bandwidth to serve multiple client stations, the server will be installed more NIC interfaces.

The theoretical model of a connection between a sender and a receiver is composed of five sequential components as shown in Figure 2.1: (1) the first stage load balancing scheduler, (2) data flow priority controller, (3) the second stage load balancing scheduler, (4) hashing table for address restoration and (5) data flow reorder.

With this architecture, the bandwidth of each network storage device can be easily increased or decreased independently. Such model can work in a normal packet switching network without the use of special hardware.

In the sender node, two schedulers are used to balance the traffic load on sender and receiver respectively. To distinguish the two schedulers, we call the first stage scheduler “Marker”. Each data flow has a Marker, however all data flows share one global second stage scheduler. Between first and second scheduler, a service algorithm is used to grant different priorities for each data flow.

In the receiver node, data flows are restored and reordered when packets arrive at different network interface. A hashing selection and an out-of-order estimation algorithm are used to maintain the data flow.

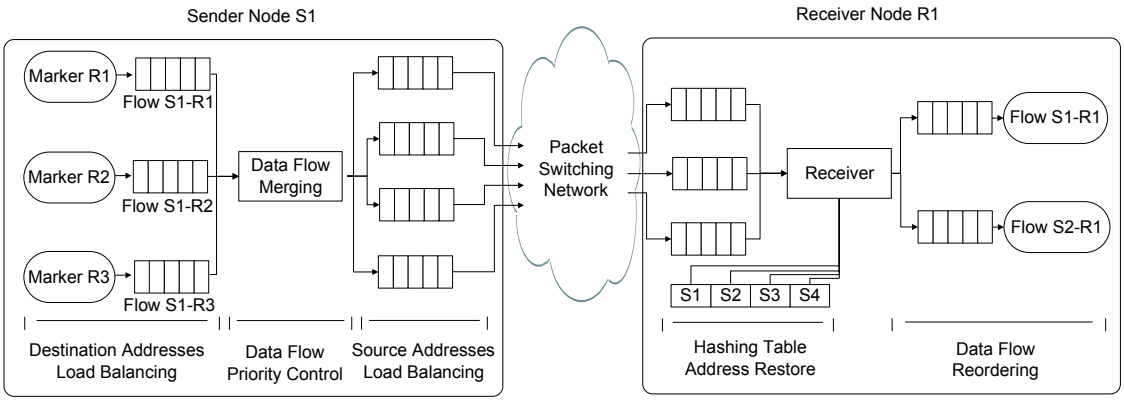

Figure 2.1 Model for Multiple Address Parallel Transmission Architecture

The following parts will demonstrate each component in detail. First, two cases are studied to reveal the distribution of traffic flows in the network.

2.2.1 Case One: Single Data Flow on Asymmetrical Networks

To demonstrate how the architecture can balance the traffic load of asymmetrical network, first a case of single data flow between sender node S and

receiver node A is considered. To simplify the illustration, both schedulers use Weighted Round Robin load balancing algorithm, other load balancing algorithms will be discussed later. Since there is only one data flow, no data flow priority selection algorithm is needed between two stage schedulers. In Figure 2.2, the data flow on sender node S is shown to explain how the packets are scheduled.

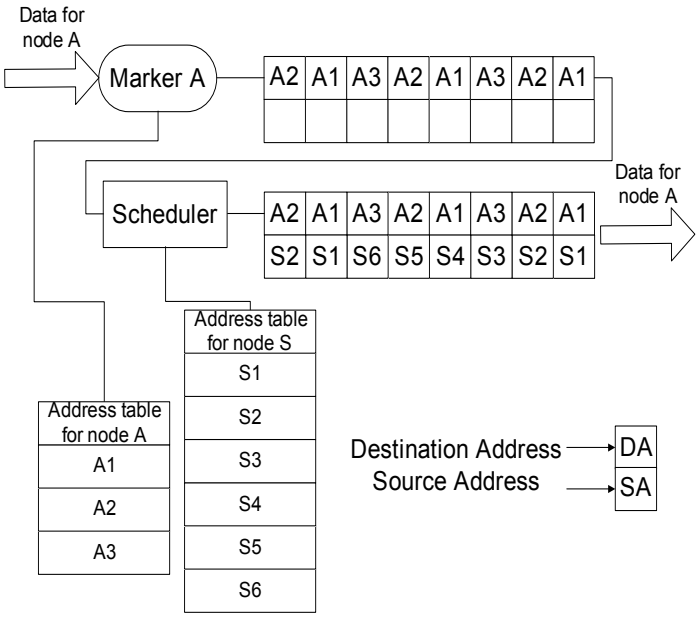

Figure 2.2 Single Data Flow Scheduling on Two Stage Address Table Scheduler

In this case, the sender node S has six network addresses (or network interfaces) from S1 to S6, and the receiver node A has three network addresses A1, A2 and A3. In the connection setup process, two nodes exchange their address table, so each node holds the address table of both sides. For data flow from node S to node A, the destination address of out going packets are first marked. Here Weighted Round Robin algorithm is used on destination address table of node A. This marking insures that the arrival traffic load at A will be balanced. Each network interface at A will receive 1/3 of the data flow. The middle switches will route packets evenly to three network interfaces on node A, and the traffic load is automatically balanced by ordinary switches. Then the second stage scheduler evenly distributes data flow to the

network interface on S. The source address of the packets is also marked with Weighted Round Robin algorithm, and the data flow will be sent out from six NIC interfaces on sender S evenly.

On the receiver node A, the in-coming data flow will be evenly received. No matter what routing algorithms are used in the packet switching networks, each NIC interface will receive same number of packets, because the destination address of the packets is predetermined by the sender node S. The major jobs on the receiver are to convert these packets into original data flow. Furthermore, since the packets go through different network path, the out-of-order is unavoidable. The receiver must reorder the packets to restore the data flow.

From the description above we show that the sender node S controls wholly the traffic load. The receiver only needs to restore the data flow. At both sender and receiver sides, the traffic load is evenly going out and going in no matter what middle network is used. Additionally, the load is balanced when the NIC interfaces on sender and receiver are asymmetrical.

There is one issue required to be mentioned that even we settle the load balancing problem, the data transmission rate is still determined by certain conditions. Because the bandwidth of the sender and receiver are different; the maximum transmission performance between two nodes is determined by the lower bandwidth side (suppose the packet switching networks provide enough bandwidth). This transmission rate is also constrained by the capability of the packet switching network.

2.2.2 Case Two: Multiple Data Flows on Asymmetrical Network

In this section a case of multiple data flows and multiple nodes is considered. Figure 2.3 shows a data flow structure on the sender node S. Three receiver nodes A, B, C build connections with node S. Where node A has 3 addresses A1, A2 and A3,

node B has 2 addresses B1 and B2, node C has 1 address C1 and sender node S has 6 addresses S1 to S6. All the schedulers use Weighted Round Robin algorithm.

In Section 2.2.1, the destination address of a data flow must be marked by scheduler to make sure the load is balanced at the receiver node. In the case of multiple data flows, each data flow has its own first stage scheduler. This scheduler will mark the destination addresses of each data flow. Here Weighted Round Robin algorithm is employed on A, B, C's address table. This insures that three data flows will evenly arrive at NIC interfaces at receiver nodes A, B, C respectively. For the multiple data flows to the same receiver node, each data flow is also assigned a first stage marker. After this first stage scheduling, we can confirm that no matter how many data flows and receiver node exist, the NIC interfaces on the same receiver will receive balanced traffic load.

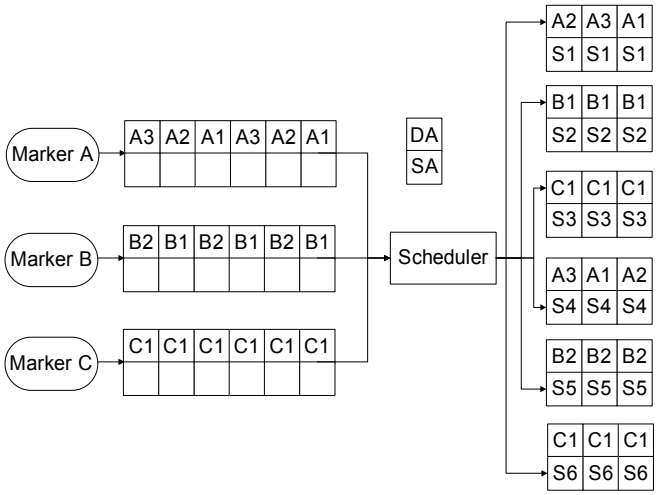

Figure 2.3 Load Balancing of Multiple Data Flows and Multiple Nodes

From first stage to second stage, three data flows will be merged into one queue and scheduled by the second stage scheduler. The mergence will provide a virtual data channel for all data flows. This design not only balances the load with simple algorithm, but also makes data flows independent of physical NIC interfaces. Thus, the data flows can migrate among the NIC interfaces freely. Whenever there is an error on

NIC interfaces, the scheduler can reschedule the data flows without affecting address marking and load balancing. Different merging algorithms can be used to provide service priorities for data flows of multiple nodes. In this case, three data flows are supposed to have the same priority and simply use a Round Robin algorithm.

After the data flows been merged into one queue, the second global scheduler uses Weighted Round Robin algorithm to distribute packets to the NIC interfaces on sender node S. Thus the packets from different data flows are evenly sent out to the network. The fact that all data flows share one global second stage scheduler insures that the out going traffic load is finely balanced.

On the receiver nodes A, B and C, the traffic load evenly arrives at specified NIC interfaces. The receiver only needs to distinguish different data flows and reorder them.

So far two cases are discussed to make clear that the traffic can be balanced at both sender and receiver side. With multiple data flows and multiple nodes with multiple NIC interfaces, the load is balanced on an end-to-end asymmetrical architecture. Furthermore, the design is very flexible that NIC interfaces on both sender and receiver can be added and removed without affecting other nodes. Multiple nodes with different bandwidth (or network interface numbers) can work together and the load balancing is maintained without any hardware support in either device or packet switching networks. In next section, we will discuss scheduling algorithms used on sender and receiver in detail.

2.3 Load Balancing Algorithms for Two Stage Schedulers

In previous section it has been shown that load balancing is controlled by two kinds of schedulers on the sender node. Each data flow has one first stage scheduler, and there is only one global second stage scheduler for all data flows. The first stage

scheduler (the marker) is used to balance the traffic load at multiple network interfaces of receiver node. The second stage scheduler is used to balance the out going traffic at the sender node. In the two demonstration cases, all schedulers use Weighted Round Robin algorithm to balance the load. Here a serial load balancing algorithms will be studied systematically.

Load balancing algorithm is used to balance traffic load to multiple channels in packet switching networks. While fair queuing algorithms are designed to fairly serve the packets form multiple queues. The load balancing algorithm could be acquired from a reverse of the fair queuing algorithm. A theorem of transformation from fair queuing to load balancing algorithm [22] has been proven. Consider a backlogged execution of a fair queuing algorithm. First a function $f(s)$ is applied to select a queue. The packet p at the head of the selected queue is transmitted and then the state is updated using a function $g(s, p)$. We can obtain a fair load sharing algorithm by using the same function f to pick a channel to transmit the next packet on, and update the state using the same function g.

Here, we intend to distribute variable sized packets to multiple channels. Each channel is backlogged and no channel is used as backup channel. Four reverse fair queuing algorithms are discussed and compared, namely Reverse Generalized Processor Sharing (RGPS), Reverse Weighted Round Robin (RWRR), Reverse Weighted Fair Queuing (RWFQ) and Reverse Deficit Round Robin (RDRR). The purpose is to determine the most suitable load balancing algorithms for the schedulers. The last three RWRR, RDRR and RWFQ can be implemented on two stage schedulers for source and destination load balancing. All these algorithms have small computation overhead.

2.3.1 Reverse Generalized Processor Sharing (RGPS)

The Generalized Processor Sharing (GPS) algorithm is used to provide a fair scheduling for best-effort connections. The GPS visits each non-empty queue in turn and serves infinitesimally small amount from each. Connections can have service weights and will receive service in proportion to these weights when they have data in the queue. If there is no data at a queue, scheduler skips to the next non-empty queue. The description of GPS is as such:

The GPS is work conserving and operates at a fixed rate r. If the N connections being served by GPS server have positive real weights $\phi(1), \phi(2)$, …, $\phi(N)$, then the server serves $S(i,\tau,t)$ amount of data from the i th connection in the interval $[\tau,t]$, so that for any connection i backlogged (A connection is backlogged whenever it has data in its queue.) in $[\tau,t]$, and for any other connection j, we have $\frac{S(i,\tau,t)}{S(j,\tau,t)} \geq \frac{\phi(i)}{\phi(j)}$. Summing all connection j, $S(i,\tau,t)\sum_j \phi_j \geq (t-\tau) r\phi_i$, and connection i is guaranteed a rate of $g_i = \frac{\phi_i}{\sum_j \phi_j} r$. GPS ensures that backlogged connections share the remaining bandwidth in proportion to their weights. GPS has many advantages. A connection can be guaranteed a throughput independent of the demands of other connections. Furthermore, by varying the ϕ_i, we have the flexibility of treating each connection in different manners.

The Reverse Generalized Processor Sharing (RGPS) algorithm is a reversal algorithm of GPS in a special condition. The RGPS is used to schedule one queue to multiple connections with service weights. The RGPS send infinitesimally small amount to each connection. In a finite time interval it will visit every connection once. In RGPS, every connection is backlogged, because we intend to use all connection

resources. For the same GPS definition, we have $\frac{S(i,\tau,t)}{S(j,\tau,t)} = \frac{\phi(i)}{\phi(j)}$, which means all connections strictly share the service in proportion to their weights. Each connection receives rate $g_i = \frac{\phi_i}{\sum_j \phi_j} r$ data flow.

RGPS is a perfect load balancing algorithm in theory, but it is unimplementable when the packets, not infinitesimal are scheduled. In the packet switching network, schedulers must approximate RGPS by scheduling packets with variable sizes. So, the concept of RGPS can help to evaluate other algorithms. Next, three simple algorithms are discussed, which are implementable and approximate to RGPS.

2.3.2 Reverse Weighted Round Robin (RWRR)

The Weighted Round Robin (WRR) algorithm [23, 24, 25, 26, 27] is widely used to serve a packet instead of infinitesimal from each backlogged queue in turn. The packet sizes should be equivalent, otherwise the schedule algorithm is unfair. To demonstrate this FQ algorithm, first, consider fixed packet size and different weights. The computation is simple that the scheduler serves more than one packet per round after normalizing weights to integers. Secondly, consider variable sized packets and different weight. The scheduler should normalize weights by mean packet size to acquire fair queuing. Since the mean packet size is usually unavailable, the WRR for variable packet size is hard to implement. Many modifications [28, 29, 30] are proposed to improve WRR algorithm.

The WRR works well for a network with fixed packet size, because the work complexity of WRR is $o(1)$ [31]. It is widely used in schedulers in ATM [32, 33, 34, 35] and IP switches and routers[36].

The Reverse Weighted Round Robin (RWRR) algorithm is the time reversal of the WRR on the condition that each connection of the WRR is backlogged. The RWRR sends packets to connections in turn. For fixed packet size and different weights, the number of served packets in one round is computed by normalized integer weights. When the RWRR is implemented on variable packet size, the fairness of RWRR is defined by $FairIndex = (Max / Min)$, where Max is the maximum packet size and Min is the minimum packet size.

The RWRR algorithm is a practical solution because of its less computation overhead. The traffic load is balanced when the packet size is fixed. In SAN, when mass data flows are transmitted, most of the data packet will use fixed maximum protocol data unit (MPDU) for efficiency. The proportion of the similar sized packets is high. Thus we can use RWRR algorithm to approximate a perfect load balancing algorithm. In Chapter 4 we will show that RWRR works well for remote disk reading in a normal packet switching environment.

2.3.3 Reverse Weighted Fair Queuing (RWFQ)

The Weighted Fair Queuing (WFQ) algorithm, also known as Packet-by-packet Generalized Processor Sharing (PGPS) [37, 38, 39], is designed for variable sized packets and weights. The WFQ algorithm serves packets in order of their finish number (or finish time), where the finish number of each packet is computed by a GPS algorithm. The finish number is given by

$$F(i,k,t) = \max\{F(i,k-1,t), R(t)\} + T(i,k,t), \qquad \textbf{(2.1)}$$

where, $$T(i,k,t) = L(i,k,t)/r \ , \qquad \textbf{(2.2)}$$

where $R(t)$ is the round number computed by GPS algorithm, $T(i,k,t)$ is the service time of k th packet on connection i, $L(i,k,t)$ is the length of k th packet that arrives on connection i at time t, and r is the link service rate.

The fairness of WFQ algorithm is $FairIndex = 1$, which means the traffic load is perfectly balanced. Based on WFQ, research on implementation and performance [40, 41, 42, 43] has been conducted widely. Many network applications currently use modified WFQ algorithms in switches and routers [44, 45, 46, 47], voice on IP [48, 49], and wireless LAN [50].

However, since the work complexity of WFQ is $o(\log n)$, research efforts are conducted to simplify WFQ [51, 52] and improve its adaptability [53]. The best solution not only balances the load, but also keeps the work complexity at $o(1)$. We will discuss a solution of work complexity $o(1)$ in the next section.

The Reverse Weighted Fair Queuing (RWFQ) algorithm is a reverse of WFQ algorithm without empty channel. The finish number is assigned to each channel and computed by RGPS algorithm. The channel with the minimum finish number is selected to send the next packet in queue. The finish number is given by

$$F(i,k,t) = F(i,k-1,t) + T(i,k,t), \qquad \textbf{(2.3)}$$

with

$$T(i,k,t) = L(i,k,t)/r \ , \qquad \textbf{(2.4)}$$

where $T(i,k,t)$ is the service time of k -th packet on channel i. $L(i,k,t)$ is the length of k th packet that arrives on connection i at time t, r is the link service rate. Unlike WFQ, since no channel is empty in the procedure of data transmission, round number are not needed when computing the $F(i,k,t)$. The weighted equation of RWFQ is

$$F(i,k,t) = F(i,k-1,t) + T(i,k,t) \div \phi(i), \qquad \textbf{(2.5)}$$

where $\phi(i)$ is the weight of the connection i.

The RWFQ algorithm is better than RWRR when packet sizes are not fixed. However, the computation overhead of RWFQ is greater than the RWRR algorithm. For each packet, all the channels need to be checked in turn and the growing finish number $F(i,k,t)$ also needs iterated deletion. If the channel numbers are huge, the computation overhead will degrade the performance. So, more efficient algorithms are required to handle the work complexity problem.

2.3.4 Reverse Deficit Round Robin (RDRR)

The Deficit Round Robin (DRR) [54] algorithm is a modification of Weighted Round Robin. To serve multiple queues, the scheduler assigns each queue a service quantum. If a queue was not able to send a packet in the previous schedule round because its packet size was too large, the remainder from the previous quantum is added to the quantum for the next round. The weight of each connection can be converted to quantum.

DDR is a low complexity version of Fair Queueing. It can be implemented at work complexity of $o(1)$ and has $FairIndex = 1$. Several modified DRR [55, 56, 57 , 58] have been proposed to achieve both simplicity and fairness. They are widely used in Internet software and hardware applications. For instance, DRR is used to provide bandwidth guarantee to competing TCP flows [59]. When combined with Random Early Detection [60], it can improve the fairness for different bandwidth capacity. In [61], DRR is used to schedule the traffic by links not by data flows, which can decrease the schedule cost. For the hardware applications, many switches and routers also use DRR algorithms [62, 63, 64].

We have more interest on a modified DRR called Surplus Round Robin (SRR) [65], which provide link striping for variable sized packets. The Deficit Counter (DC) of SRR is initialized to 0, and incremented by the quantum for the served queue. As

long as the DC is positive, packets are sent form that queue, and the DC is decremented by the size of the transmitted packet. If the DC is non-positive, the current queue will be skipped.

The Reverse Deficit Round Robin (RDRR) algorithm is a reverse DRR or SRR algorithm without empty queue. A detailed proof of transforming from FQ to load balancing algorithm can be found in [65]. The RDRR also use a Deficit Counter (DC) and quantum of service. Quantum is assigned by measuring the weight of the channels. The DC is initialized as 0 and served in turn. Each time a channel is selected, the DC is incremented by the quantum for that channel. Packets are sent to the channel and its DC is decremented by the packet size, till the DC becomes non-positive, then the next channel is selected.

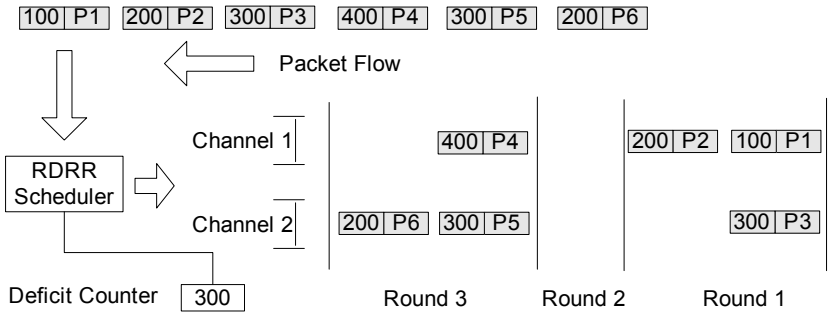

Figure 2.4 Reverse Deficit Round Robin Scheduler

In Figure 2.4, a RDRR scheduler with two channels is demonstrated. Packets of different sizes are scheduled by checking the Deficit Counter of each channel. If the packet size is less than DC, it will be sent to this channel and the DC is decreased by the packet size. If the packet size is bigger than DC, the scheduler will check next channel. If the packet size is bigger than all channel's DC, the scheduler will wait next round until the DCs are incremented. Suppose that the maximum quantum among all the channels is Quantum and the maximum packet size is Max. After K round, the difference between the bytes that should have been sent to channel i ,

i.e., $Max + 2 \times Quantum$, and the bytes actually sent to channel i is bounded by $K \times Quantum_i$.

The computation overhead of the RDRR is slightly greater than the RWRR because of the quantum service. The actual value of the quantum should be cautiously selected. If the quantum is much less than the packet size, the scheduler will wait many rounds to accumulate the quantum to send out one packet. If the quantum is much greater than packet size, many packets will be sent to one channel in one round, which may increase out-of-order rate.

The RWRR algorithm is simple and efficient for scheduling large number of channels with fixed packet size. The RWFQ algorithm conducts a perfect load balancing and data flow ordering, it can be used when storage system has excellent computation power. The RDRR algorithm is a better solution when many channels with various packet sizes are used and computation resource is constrained. For different network environment and data transmission models, proper algorithm can be selected for specific requirements.

2.4 Data Flow Priority

In this section we will explain why one data flow (or one conversation) is scheduled on all the addresses and how to assign priorities to multiple data flows on one device. Between first and second scheduler, we build one virtual data channel between nodes instead of multiple smaller data channels and use service disciplines algorithm to provide different priorities and rate-control for each data flow.

2.4.1 One Virtual Channel vs. Multiple Sub Channels

We use a structure of two stage schedulers to build one virtual data channel between nodes in stead of multiple smaller data channels, which means that each data

flow is scheduled on all workable sub channels for the efficiency and fault tolerance. Other protocols, for instance Link Aggregation Control Protocol (LACP), use a different method. It transfers one conversation on one physical network interface to avoid packets disorder. That will increase the delay, and can not handle load balancing problem for asymmetrical architecture.

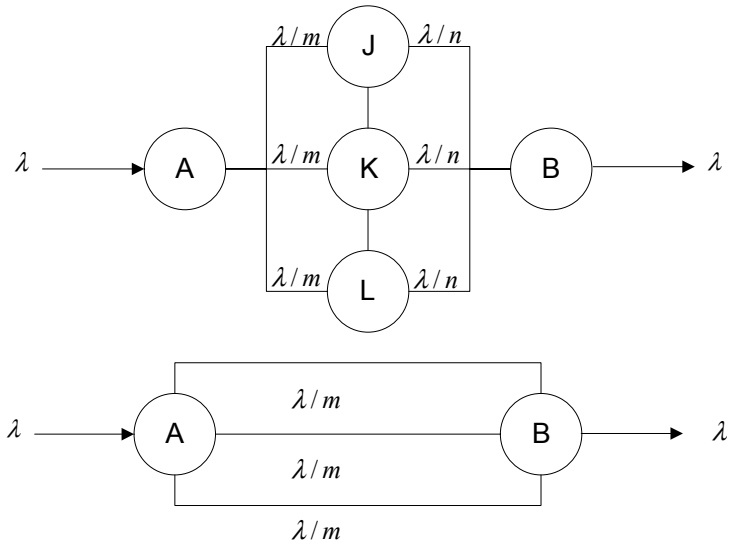

Figure 2.5 Single Virtual Channel and Multiple Channels

To compare the performance when multiple physical channels are used as one big data channel or multiple smaller data channels, consider a communication link serving m independent Poisson traffic streams with overall rate λ [66]. Suppose that the link is divided into m separate channels with one channel assigned to each traffic stream. However, if a traffic stream has no packet awaiting transmission, its corresponding channel is used to transmit a packet of another traffic stream. The transmission times of packet on each of the channels are exponentially distributed with mean $1/\mu$. The system can be modeled by the same Markov chain as the M/M/m queue. In the case of multiple sub channels, the average delay per packet is given by the M/M/m average delay expression

$$T = \frac{1}{\mu} + \frac{P_Q}{m\mu - \lambda}. \qquad \textbf{(2.6)}$$

While in the case of one virtual channel, the model is an M/M/1 system with the same arrival rate λ and service rate $m\mu$. The average delay per packet is

$$\hat{T} = \frac{1}{m\mu} + \frac{\hat{P}_Q}{m\mu - \lambda}, \qquad \textbf{(2.7)}$$

where P_Q and $\hat{P}_Q$ denote the queuing probability in each case ($P_Q = \frac{p0(m\rho)^m}{m!(1-\rho)}$), λ denotes the arrival rate (the inverse of average inter-arrival time), μ denotes the service rate (the inverse of average service time). When $\rho << 1$ (lightly loaded system) we have $P_Q \cong 0$, $\hat{P}_Q \cong 0$ and $\frac{T}{\hat{T}} \cong m$. When ρ is only slightly less than 1, we have $P_Q \cong 1, \hat{P}_Q \cong 1, 1/\mu << 1/(m\mu - \lambda)$ and $\frac{T}{\hat{T}} \cong 1$.

Therefore, for a light load, multiple sub channels produce a delay almost *m* times greater than the delay of one virtual channel. For a heavy load, the ratio of the two delays is close to 1. On average, keeping multiple conversations on one virtual channel is more efficient for packet transmission. This gives evidence that when implementing multiple address parallel transmission protocol, one conversation should view multiple network interfaces as one virtual channel. This also implies that packets of the conversation can migrate among the network interface without applications' awareness.

In our design, data flows are not constrained to specific channel. Packets of the data flows can be marked with any workable source and destination address pairs. Scheduled by the two stage scheduler, one conversation between two nodes takes multiple paths as one virtual channel.

2.4.2 Multiple Data Flows' Priorities with Service Discipline

The two stage schedulers can handle the end-to-end load balancing with multiple data flows. However the end-to-end performance guarantee is still need to considered, which requires various QoS service discipline algorithms.

One of the targets is to categorize the data flows. Since multiple data flows can be transmitted simultaneously in the architecture, service disciplines must be provided to determine the priorities of the data flows. For instance, the real-time data flows should have higher priority than non-real-time data flow.

The other target is to control the rate of each data flow to avoid the congestion. Since the best-effort data flow might cause unnecessary congestion and retransmission in the asymmetrical parallel architecture, a rate-controlled data flow can have better performance. Traditional congestion control methods [67, 68, 69] did not consider about the parallel architecture, thus an end-to-end congestion control method for parallel path is urgently needed.

In current packet switching network, the service disciplines which can provide per-connection end-to-end performance guarantee are still unavailable in most of the routers and switches. So the data flow rate and congestion can not be fully controlled before all the routers and switches along the path support QoS services. However, we can simply utilize service discipline algorithms on the source node to achieve the priority target. The target of rate control and congestion avoidance can be fulfilled when the whole packet switching network can support QoS service.

In recent years, QoS service disciplines have been widely studied. Two categories of service disciplines, the work-conserving disciplines and the non-work-conserving disciplines, can be found in [70, 71]. With the work-conserving disciplines such as, virtual clock, delay-EDD, WFQ, WF^2Q, and SCFQ, a sorted priority queue

mechanism can be used to provide priority service for the data flows. With non-work-conserving disciplines such as, jitter-EDD, Stop-and-Go, HRR and RCSP, a rate-controlled service can be employed to reduce the possibility of the congestion.

The best place for in this service discipline is between two stage schedulers. The choice of this priority algorithm highly depends on the applications. In our implementation, all data flows are assigned the same priority; and the transmission rate is controlled by the minimum bandwidth side of an asymmetrical parallel architecture. Other algorithms can also be adopted to meet the QoS requirement of different applications.

2.5 Data Flow Restore --Hashing Address Table and Reordering

In the receiver node, data flows are restored and reordered when packets arrive at different network interfaces. Here, a chaining hashing algorithm is used to search senders' address table and an out-of-order estimation algorithm is used to setup retransmission threshold and allocate reordering buffer. Since the mechanism of the out-of-order differs from a normal network, several closely related parameters are studied in detail.

2.5.1 Hash Chaining Structure for Address Restoring

In the parallel transmission architecture, the packets of a data flow arrive at different network interfaces. The receiver must reorganize these packets into their original data flows quickly. Indexed by their source addresses, packets are differentiated by using a hash chaining searching on senders' addresses table.

The search time of normal hashing table is $o(1)$, but this may waste too much memory resource when the address range is large. Here a hash chaining structure is used to balance the memory space and searching time. The searching time of a

chaining with separate lists is $o(n/l)$, where n is the number of addresses and l is the number of lists. The addresses with the same hash key are organized in one list to avoid collision. By adjusting the list number l, the search time can be controlled in a definite range.

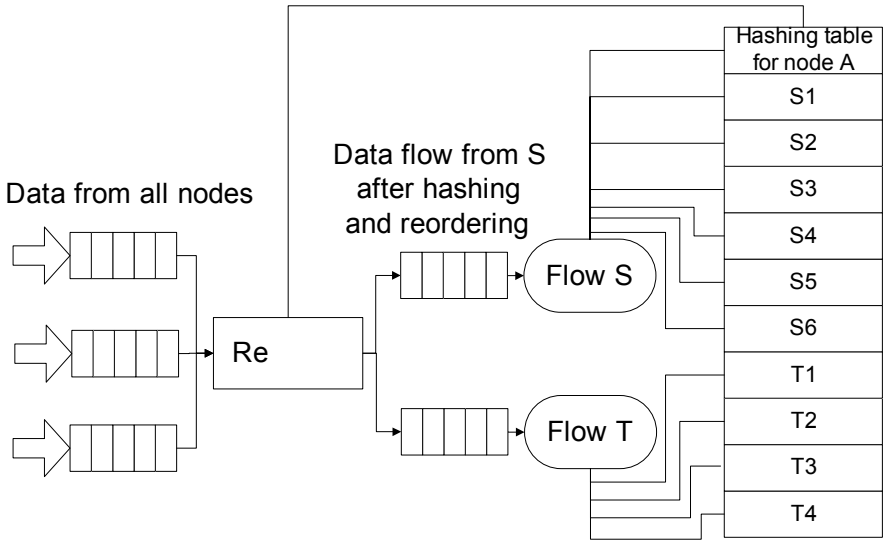

Figure 2.6 Hashing Algorithm for Data Flow Restoring

After the hash chaining searching, data flows from the same sender node are differentiated. Since every packet contains the unique ID of data flow on the same sender, the receiver will further classify different data flow by data flow ID. This procedure is quite simple and straightforward, thus, the major burden is the address table searching. To restore the data flow by searching sender's address table, the hash chaining address table structure is an efficient algorithm.

2.5.2 Out-of-order Analysis

In this section, we will discuss the reason of the out-of-order in parallel architecture. The out-of-order problem is quite prevalent in parallel transmission structure, the most reliable method is to mark sequence number on each packet. In our design, each packet in data flow is numbered and reordered with this sequence number.

The receiver reorders data flows by using a TCP-like algorithm. However, different parameters are used for the retransmission algorithm when out-of-order occurs. These parameters will be discussed in next section.

In normal network, out-of-order rate is not significant and hard to predict. The out-of-order is mainly caused by route changing. It rarely has a big impact on performance of a TCP similar protocol, because the scale of the out-of-order delivery is just a few packets.

In parallel transmission architecture, out-of-order is hard to avoid but can be predicted. In a theoretical parallel path transmission model, different channel bandwidths and packet sizes might cause out-of-order delivery. In a practical environment, the schedule delay and other system structure also influence the probability of out-of-order delivery.

First suppose that two nodes communicate through three sub channels with the same bandwidth (Figure 2.7). Three packets are sent to each channel at the time sequences t1, t2, t3. Since the sizes of the three packets are different, the packet 2 and 3 will arrive before the packet 1 (the arrival time of the last bit of a packet determines the arrival time of this whole packet). This will cause the receiver node to reorder the packets. While in a single channel case, the second packet will not be sent until the last bit of the first packet are sent out.

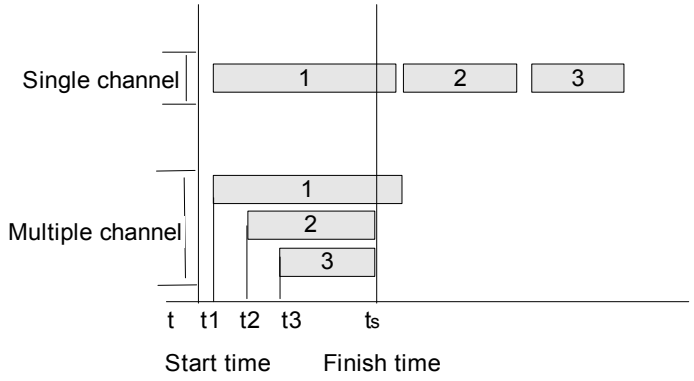

Figure 2.7 Out-of-order Caused by Different Packet Sizes

Different channel bandwidth can also cause out-of-order delivery with the similar mechanism. Consider packets with same size are sent into channels with different bandwidth. The arrival time sequence of each packet will not follow the sending sequence. In a channel with larger bandwidth packets can arrive early. To decrease the out-of-order delivery, the packet sizes should be constrained to fixed value and channels should be allocated with the same bandwidth.

It is worth mentioning that other system parameters may also cause out-of-order. Small schedule delay (Figure 2.8) can cause out-of-order delivery even when packet size and channel bandwidth are fixed. The schedule delay is the schedule time interval between two packets in the same schedule round. Since packets are transmitted at different network paths, if the schedule delay between two packets is small, the possibility that these two packets reach the receiver at same time is high, which denotes a high out-of-order rate.

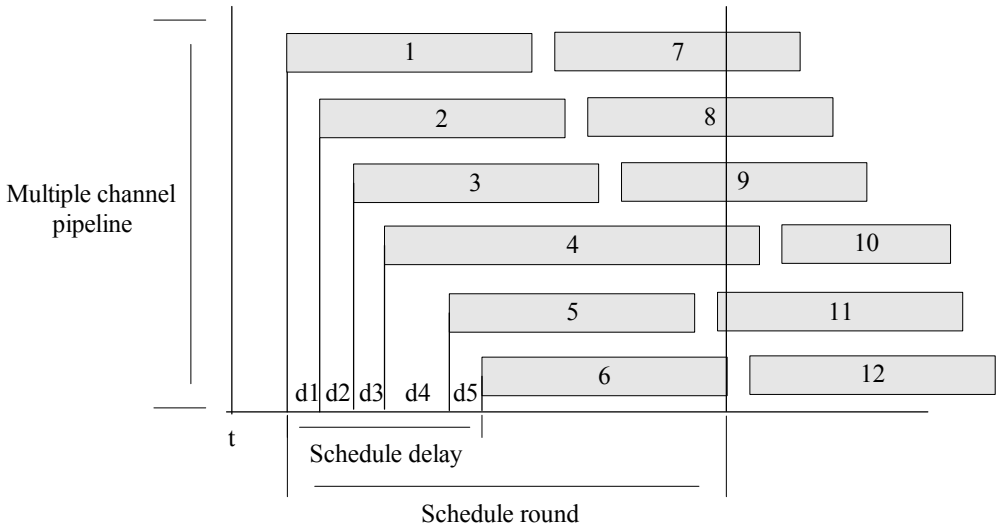

Figure 2.8 Schedule Delay in Each Schedule Round

The schedule delay can be affected by channel numbers, packet sizes. With more channels added, more packets need to be scheduled in one round. Thus the

schedule delay extends (Figure 2.8) and decreases the possibility of the out-of-order delivery. Similar result can also be observed when the packet size is increased, schedule delay also increases. With larger packet size, the out-of-order rate drops. Different system workload may also change the schedule delay. Thus, a stable system environment is required to test the out-of-order delivery caused by schedule delay.

The out-of-order delivery can be a function of the schedule delay, the variance of packet size and the variance of channel bandwidth. Define the out-of-order rate γ_o as the frequency that the system observes out-of-order delivery. We can have:

$$\gamma_0 = f[Var(packet\_size), Var(bandwidth), 1/T(schedule\_delay)], \quad \textbf{(2.8)}$$

$$T(schedule\_delay) = f[channel\_No., packet\_size], \quad \textbf{(2.9)}$$

The out-of-order rate γ_o will increase when the variance of packet size or channel bandwidth increases. The γ_o decreases when schedule delay increases, where the schedule delay can be increased by channel numbers and packet size.

Nevertheless, network storage device may use a multiple processor operating system. In such a system, different packet processing times on multiple processors can also cause out-of-order delivery. These problems should also be considered in the design when such operating systems are used.

The out-of-order delivery in parallel transmission architecture may affect many other network characteristics, such as, retransmission, window size, drop rate estimation, and buffer size. It also has an impact on the queuing delay analysis, since most of the queuing models are first-in-first-out (FIFO).

2.5.3 Retransmission Threshold and Reorder Buffer

In this section, we will deduce the retransmission threshold and reorder buffer by estimating out-of-order rate in parallel transmission architecture. The transmission performance and out-of-order distance are also discussed.

The relation between out-of-order and the retransmission is a key architectural consideration in protocol design [72]. Previous out-of-order and retransmission research focuses on Internet and TCP performance [73, 74, 75]. In such network environment, the scale of out-of-order is a few packets and the packet drop rate is relatively large (mainly caused by network congestion or data error). When there is either out-of-order or packet drop in a packet flow, the receiver will observe sequence broken and this may cause a retransmission. Thus, it is important for a fast retransmit mechanism to disambiguate packet drop from out-of-order delivery. The TCP uses a duplicate ACK threshold to infer that a packet requires retransmission. This threshold is selected by estimating how long a receiver needs to wait to disambiguate out-of-order from packet drop [76]. Normally the threshold is been given a small value to improve the fast retransmit mechanism. This suggests that a short wait time is preferred, since most of the sequence broken is caused by packet drop.

In SAN with parallel transmission structure, the selection of duplicate ACK threshold depends more on out-of-order rate estimation to avoid unnecessary retransmission. Since SAN is more stable than Internet, the packet drop is relatively rare, while out-of-order delivery is more prevalent [77].

The out-of-order delivery can be estimated and used to select a proper retransmitting waiting time and reorder buffer threshold. The waiting time has an upper bound τ, where τ can be deduced from the maximum Round Trip Time (RTT) of the data packet. Since the bandwidth and packet size can be variable, the maximum

RTT is the time that a maximum sized packet used to pass through the minimum bandwidth channel. The buffer bound for reordering is the integration of maximum RTT multiplying the individual channel bandwidth.

Consider that the receiver observes a missing packet at sequence number *N* by receiving packet *N*-1 and *N*+ 1. Suppose that the missing packet is caused by out-of-order and all sub channels are using a FIFO queue, the *N*-th packet must have been sent out but still not arrive. The worst case is that the *N*-th packet has maximum size and is passing through the minimum bandwidth channel. So we have $\tau = Max(RTT)/2$, if the waiting time exceeds τ, the possibility of packet drop increases. Thus the retransmission threshold must be larger than τ. The minimum buffer used to reorder one out-of-order packet can also be deduced by

$$W = \sum_{i=1}^{N} \tau * \varpi_i = \sum_{i=1}^{N} Max(RTT) * \varpi_i / 2, \qquad \textbf{(2.10)}$$

where *W* is the buffer size, *N* is the numbers of the channel, ϖ_i is bandwidth of each channel. This equation shows that in a worst case out-of-order, the data waiting in a reordering buffer equals all the received traffic within this waiting period.

Let out-of-order distance D_o be the number of packets that have arrived between the period of time in which that the system observes a missing packet and this missing packet arrives. Since each packet will arrive within time τ, the D_o or the number of packets arrived before the out-of-order one can be estimated by:

$$D_o = \frac{W}{Average\_Pakcet\_Size} = \frac{\sum_{i=1}^{M} Max(RTT) * w_i / 2}{Average\_Pakcet\_Size} \qquad \textbf{(2.11)}$$

This shows that the out-of-order distance can be controlled in finite range. Theoretically, the small out-of-order distance will not affect the transmission performance, it only increases the delay bound by waiting and reordering packets. The

estimation of the retransmission waiting time and reordering buffer produces a slightly loose bound for the queuing delay model while the transmission performance is unchanged.

The estimations are based on maximum Round Trip Time, which can be easily acquired and computed. Estimation algorithms based on out-of-order rate requires more knowledge about network topology and scheduling algorithm. For different applications, the out-of-order rate might have different values. Our estimation also provides a general delay bound and all the other parameters can be efficiently computed.

2.6 Fault Detection and Cost Estimation

The multiple addresses parallel transmission uses redundant channels to provide built-in fault tolerance feature. Whenever a fault channel is detected, data flows will be scheduled to other available channels.

To achieve fault tolerance, the protocol must be able to detect the error. It must recover from that error and eventually reaches a legal state and resumes its normal execution [78,79]. Several detection algorithms can be found in [80]. Most fault tolerance algorithms use spare resources to provide fault tolerance feature [81, 82, 83], which may be resource waste [84]. In our architecture, all the available resources are utilized to improve performance and achieve fault tolerance at the same time.

2.6.1 Three Fault Detection algorithms

Since NIC failure, cable failure and switch failure can all contribute to channel failure, we propose three fault detection methods to diagnose different levels of failures. The three fault detection algorithms can be hardware detection, software local

loop back and remote acknowledge detection. All these algorithms can be implemented jointly or independently.

The hardware fault detection depends on the link failure register of the network interface cards. Currently most of the network interface cards support a link failure register. By checking the register periodically, the status of the first hop link can be collected. The computation overhead of checking a register is very small and the detection frequency could be a few seconds. The limitation of the hardware fault detection is that node can only check the first hop link in the packet-switching network.

The second algorithm is local loop back fault detection. We can build loop back channels by using the multiple addresses on the same node (the channel number should be greater than or equal to 2). The merit of the local loop back detection is that if the hardware detection is unavailable, a node can send detection packets to itself to detect the status of the first hop link and the nearest switch. The draw back of this algorithm is that the working addresses must be greater than or equal to 2. The procedure of the loop back detection can be described by:

(A.) Select first network interface card (NIC) to be sender, send local detect packet to other NICs on the same node

(B.) If at least one of these NICs received a local detect packet, it is suggests that the sender NIC and the nearest switch is working well. Those who have not received local detect packet can be regarded is faulty. The fault might be on the cable or the card. The detection process can be finished.

(C.) If none of the other NICs received the local detect packet, we mark the sender NIC with “unknown”, because we cannot assure if the sender NIC has fault. So we choose next NIC as sender, send local detect packet to remaining NICs and continue the B, C process.

(D.) If no channel can be built on the local node, there are only two possible results. Either only one NIC is working or no NIC is working. To identify the only one working port (if existed), we must use remote detection algorithm.

The third algorithm is remote fault detection. The mechanism is to build channels with the neighboring nodes to detect fault. A node can ask another node to reply back detection packets to all of its NICs. If any NIC interface cannot receive the detection, a fault in the data path is found. The advantage of this algorithm is that the full data path is checked. The drawback is that there must be another neighboring node to send back detection packets.

Using hardware and local loop back detection could eliminate unnecessary load in the LAN. It can also make the detection fast. The remote fault detection algorithm is relatively reliable, however it will cause more traffic load and time.

2.6.2 Cost Estimation for Local and Remote Fault Detection

The cost estimation of hardware detection is quite straightforward. In one detection round, each network interface card is checked once. The cost estimation of the local loop back and remote acknowledge detections are more complicated. Suppose a node has N network interface cards, and W of the interface cards are working. The detection cost unit is the cost of sending and receiving a packet. The fault detection cost function will be:

$$\begin{cases} F_l(n) = (n-1)\dfrac{W}{n} + [(n-1) + F_l(n-1)]\dfrac{n-W}{n}, F_l(W) = W-1 \quad n = 2....N; W \geq 2; W \leq N \\ F_r(n) = (n-1) + (n-2) + ... + 1 + n \qquad n = 1....N; W = 0,1; W < N \end{cases}$$

(2.12)

$F_l(n)$ is the cost of local loop back detection. $W \geq 2$ suggests that working NICs must be more than 2. By building a channel between two NICs, the fault can be

detected locally. The first part of $F_l(n)$ shows if we select one working NIC with probability $\frac{W}{n}$, the detection can be finished with cost (n-1). The second part of $F_l(n)$ shows if we select a failed NIC, the detection must be repeated again until all the remaining ports are working NICs, where $F_l(W) = W - 1$ is the boundary of the formula.

$F_r(n)$ is the cost of local loop back plus remote acknowledge detection when there is at most one NIC still working. Since there is only one NIC or no NIC working, local loop back detection will go through all the ports, each time it sends (n-1) packets to remaining NICs. At the last step, the local loop back detection can not find this fault. So, every NIC will send out a remote acknowledge packet which is denoted by cost (n). The simplification of the cost estimation formula is:

$$\begin{cases} F_l(n) = (n-1) + F_l(n-1)\dfrac{n-W}{n}, F_l(W) = W-1 \quad n = 2....N; W \geq 2; W \leq N \\ F_r(n) = \dfrac{n(n+1)}{2} \qquad n = 1....N; W = 0,1; W < N \end{cases}$$

(2.13)

Denoted by *N, W*, Cost, we can have a cost estimation chart (Figure 2.9). This chart shows that when the number of working NICs is close to *N*, the detection cost is low.

Suppose in the fault detection period, the fault probability of each NIC is p. Then the probability that there are W working NICs can be denoted by

$$P = (1-p)^W * p^{N-W} \qquad N > 1; W \leq N \qquad \textbf{(2.14)}$$

In Figure 2.10, the probability value of the working NICs is quite large when W close to N. It suggests that the probability that most of the NICs is working is very high. In this chart, we assume p=0.1 which is quite a large fault probability to

demonstrate the trend. Normally, the *p* value in LAN is very small, which will result in the *W* more close to *N*.

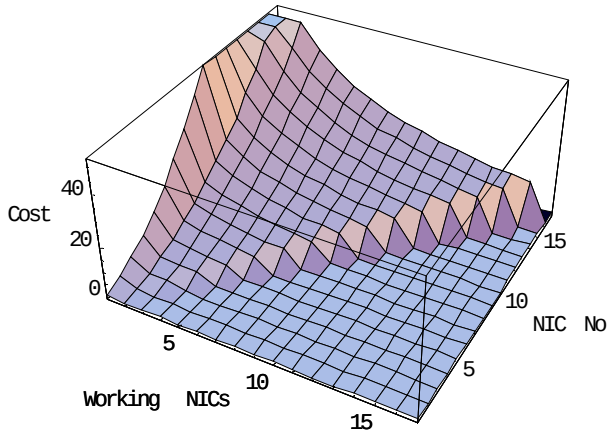

Figure 2.9 Local Loop Back and Remote Acknowledge Detection Cost Estimation

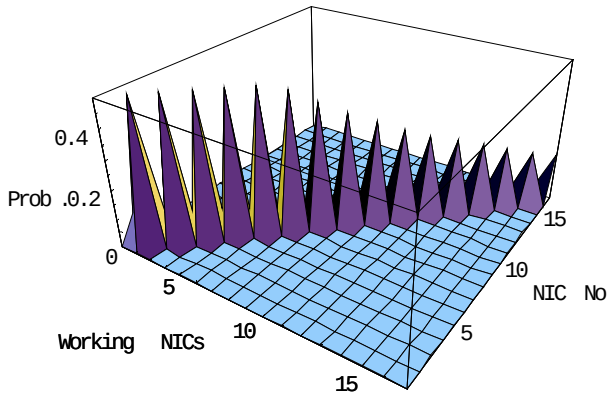

Figure 2.10 Probability Distribution of the Working NICs

By comparing Figure 2.9 and Figure 2.10 we can find that under normal condition, where *W* is close to *N*, the detection cost is small. The cost will increase only when many interfaces have fault (*W* is much less than *N*). However, the probability of many faults is so small that the detection cost normally remains to (*n*-1). This means that it is worthy maintaining a periodical fault detection mechanism.

In this chapter the theories of the network topology and parallel data transmission model was studied. An end-to-end parallel data transmission architecture was presented based on packet switching network that has variable bandwidth and packet size. The model of this transmission architecture is composed of two stage load balancing algorithm, data flow priority algorithm, data flow restoring, reordering and fault tolerance algorithm.

On sender devices, four theoretical reverse FQ algorithms, RGPS, RWRR, RDRR and RWFQ, were proposed for scheduling with small computation overhead. Then we proved that one virtual channel is better than multiple sub channels and used service disciplines algorithm to provide different priorities and rate-control for each data flow. On receiver devices, a chaining hashing algorithm was presented to search senders' address table with bounded delay. An efficient out-of-order estimation algorithm was proposed to setup retransmission threshold and allocate reordering buffer. The mechanism of the out-of-order in parallel architecture was analyzed by schedule delay and out-of-order distance. Fault detection algorithms of hardware based, software local loop back and remote acknowledge detection methods were proposed. SAN using this end-to-end parallel transmission architecture can have good scalability and fault tolerance without much additional complexity.

Chapter Three

3 Implementation

In Chapter 2, the theory of end-to-end parallel transmission architecture was proposed to provide scalability and fault tolerance for SAN. Based on this architecture, a transmission protocol will be implemented in this chapter. Firstly, the individual elements of the end-to-end parallel transmission architecture will be mapped to the major modules and sub functions of the protocol. Secondly, the logical relationship between these modules and functions will be demonstrated by software flow charts.

3.1 Multiple Address Parallel Transmission Protocol

The multiple address parallel transmission (Multi-Channel) protocol can be programmed above the Link layer or Network layer. The load balancing scheduler can use either Ethernet MAC address or IP address. To demonstrate the Multi-Channel performance in SAN, HyperSCSI [8] network storage protocol is employed as the Multi-Channel upper layer. The Multi-Channel protocol is implemented in the Linux kernel [85, 86], where a virtual network device driver was created to control the data flows and the data link layer. The general protocol structure is shown in Figure 3.1.

The Multi-Channel protocol includes three major function modules in Figure 3.1: data module, control module and a global state machine module. The data module is responsible for sending and receiving data flow packets through multiple network channels. The control module is used to manage multiple channels and select load balancing algorithm. The fault tolerance is also involved in the control module. The global state machine inspects global parameters and synchronizes the control and data

transmission states transition. In the following sections, we will specify the functions of each module.

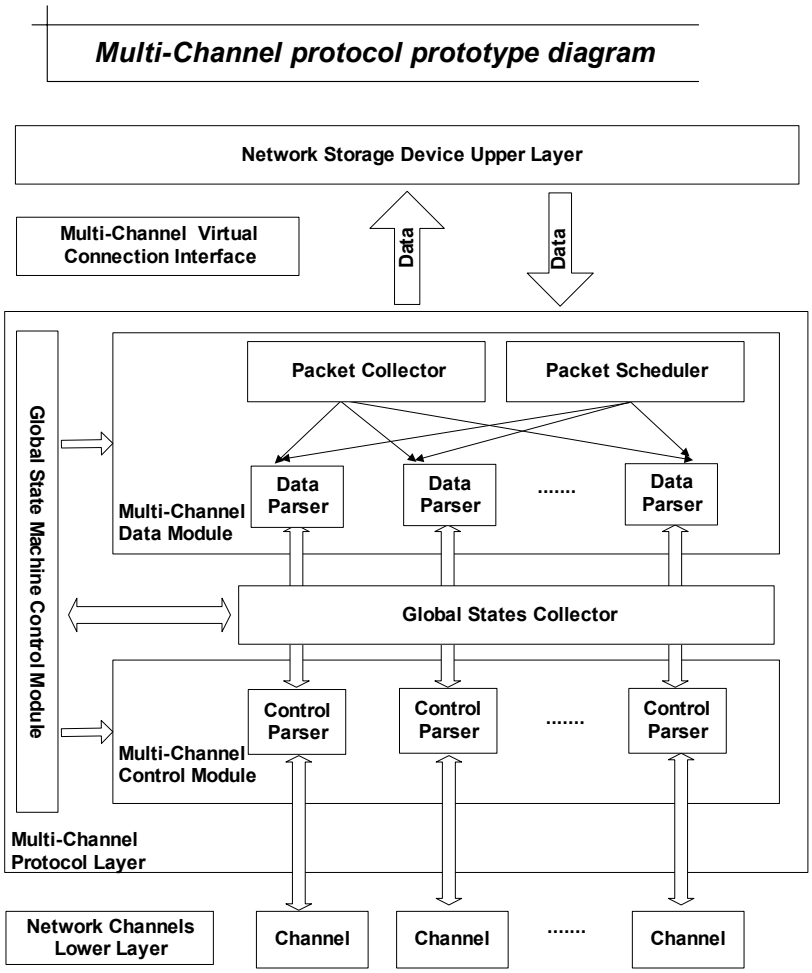

Figure 3.1 Multi-Channel Protocol Layer Diagram

3.1.1 Multi-Channel Data Module

Multi-channel data module is in charge of scheduling the data flow packets through multiple network channels.

When sending packets, the data module first checks the address table of the destination node and selects a load balancing algorithm and marks the destination address of this data flow. Thus the destination node will receive a balanced traffic load without the interference of the switches in between. In the second stage, the data

module checks the address table of sender node and selects a load balancing algorithm and marks the source address of all data flows. Since the second stage scheduler is a global scheduler, different data flows can share all the network channels. A balanced traffic load will be sent into the network.

When receiving packets, Multi-Channel data module collects packets from all the network channels. Data flows are distinguished by checking the address tables of the nodes. Using a proper hash chaining structure, the computation overhead of address searching is not significant. After the data flow is distinguished, the receiver will handle retransmission, window size and buffer size if there is out-of-order delivery. Then the packets will be reordered and sent to the upper layer applications.

3.1.2 Multi-Channel Control Module

The control module is responsible for managing multiple Ethernet or IP channels and selecting load balancing algorithm. The major control processes are listed below.

Multi-Channel startup process is used to initialize one virtual network driver by using multiple sub-channels. Data flow structures are connected to the address tables and address searching tables, which are used to send and receive data packets.

Address-discover process sends address-discover packets and retrieves address table from the address-discover ACK packets. In Ethernet network, the address-discover packets can be broadcasted to the LAN, while in IP network the address-discover packets are sent by point-to-point fashion.

Channel management process is used to dynamically append or remove sub-channels in virtual connection. For data link layer, operations will work on network device drivers. For IP layer, socket will be used. Whenever there is new channel

configuration, the process will call address service processes to update local and remote address tables.

Changing local address process is one of the address service processes. It is in charge of updating the local address table and sending this new address table to other active partners who have built connections with it.

Checking remote address process is also one of the address service processes. It enquires remote partners' address table and reschedules the data flow whenever there is address changing.

Fault detect processes include hardware detect, local loop back fault detect and remote acknowledge fault detect process. They check the channel status periodically. If there are channel faults, the channel management process will be informed and hence the channels will be rescheduled.

Channel services process sends and receives control packets which exchange channel information. The load balancing algorithms on two stage schedulers can also be selected by the control packets in this process.

3.1.3 Global State Machine Module

Global state machine manages all control and data transmission states transition. It inspects global parameters, such as, address table lists, fault detection algorithm, load balancing algorithm, control configuration, etc. So the data transmission and control flow can be synchronized. All the operation commands will refer to this module to avoid operating disorder.

Each connection can have a major state and a minor state at one time. The major states describe the major operation groups. The state of the connection must follow the state transition diagram in Figure 3.2. In each major state, there are several

detail steps exists. The minor states describe these steps that the operation should follow. They may be dissimilar in different applications.

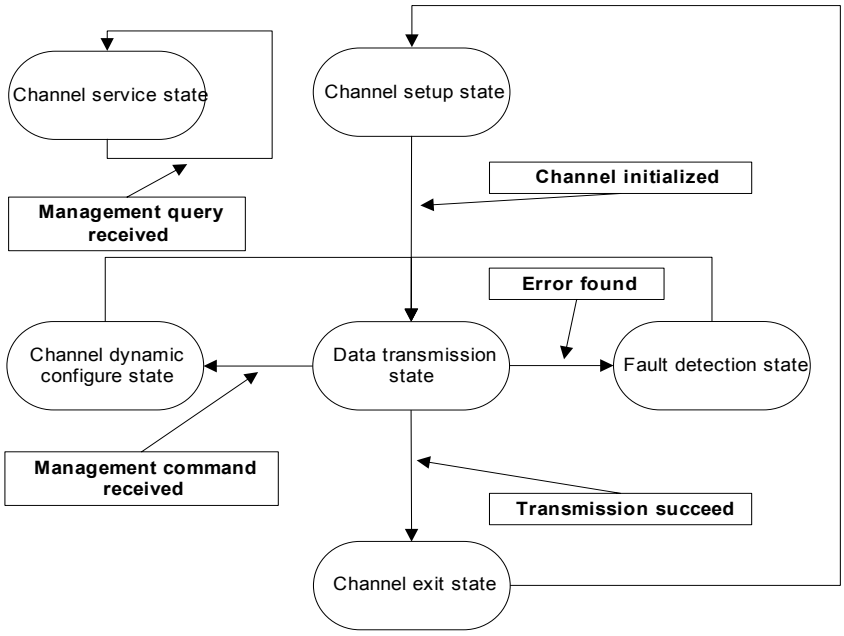

Figure 3.2 Multi-Channel Data Transmission State Transition Diagram

Here we list the major and minor states:

- Channel setup state: address discover, address table exchange, load balancing algorithm select, data flow setup.
- Data transmission state: data packets sending and receiving in multiple channels, hashing address search, out-of-order delivery control.
- Channel dynamic configure: new address table exchange and load balancing algorithm change.
- Fault detection state: hardware detect, local loop back detect, remote acknowledge detect. If fault detected, update address table and inform both side.
- Channel services state: channel information query and control command decode.
- Channel exit state: channel disconnect, channel resource release.

3.2 Multi-Channel Protocol Processes Software Flow Charts

In previous sections we discussed the MAP-TP protocol and functions that the protocol supports. In following sections, we will present the flow charts of the software protocol implementation.

3.2.1 Major Flow Charts, Key Data Structures and Events List

The software flow chart of MAP-TP protocol is composed of two major process flow groups, which conduct the functions of data module and control module as shown in Figure 3.3.

One flow group is data processes, which include channel address scheduling process and address restore process. The data processes start operating after the Multi-channel startup process and address discover process and stop operating before the channel exit process. These data processes can operate with other control processes simultaneously. Dynamical channel changing and fault detection should not stop the data transmission, unless there is not available channel at all.

The other flow group is Multi-channel management processes which employ: Multi-channel startup process, address discover process, channel manage process, fault detect process, change local address process, check remote address process, channel services process and channel exit process. Excluding the channel startup and channel exit processes, all other management processes can operate independently. The channel manage process is a public interface which accepts all kinds of commands and sends events to other processes. It is also in charge of the state synchronization.

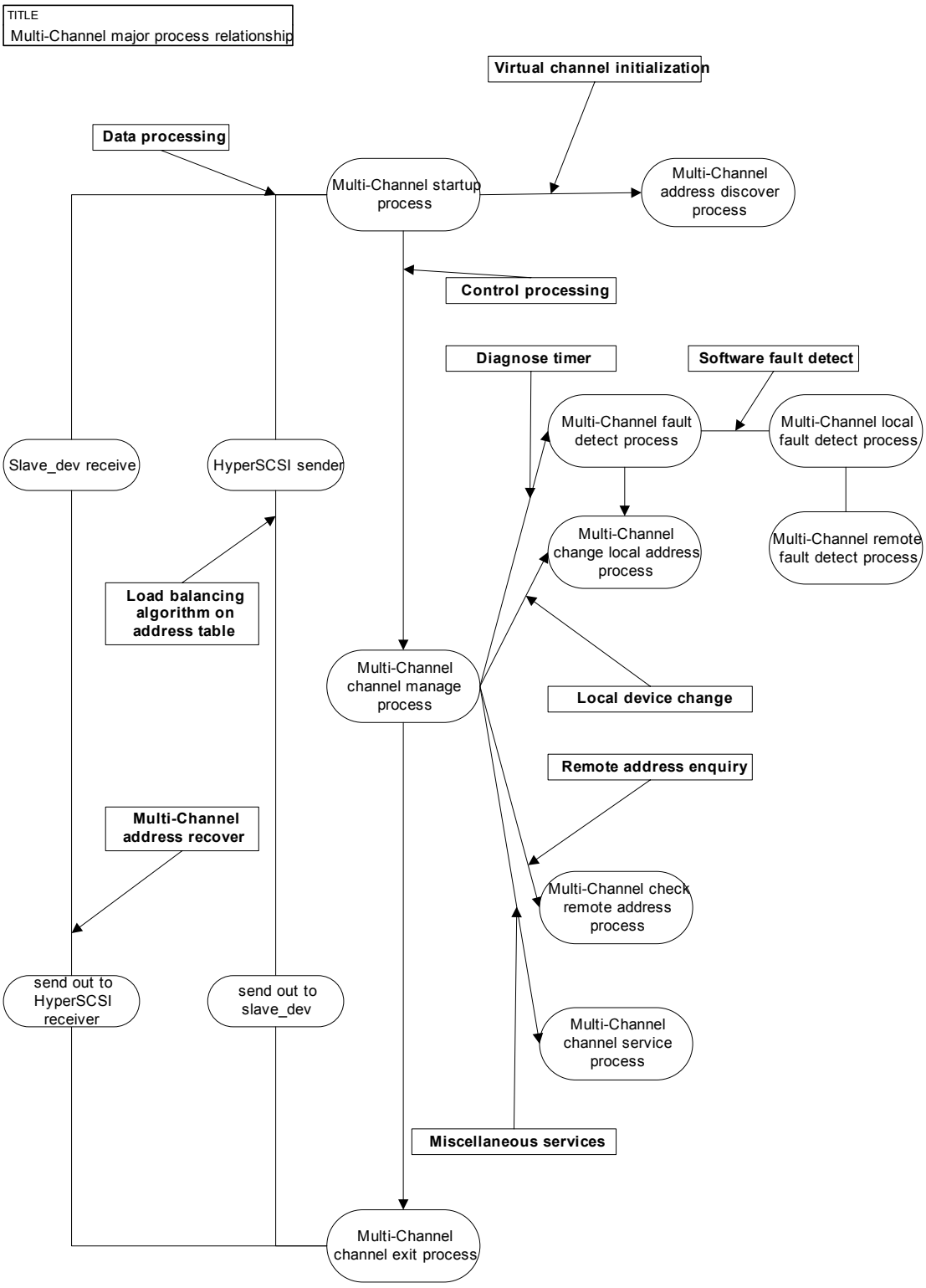

Figure 3.3 Data and Control Processes' Flow Chart

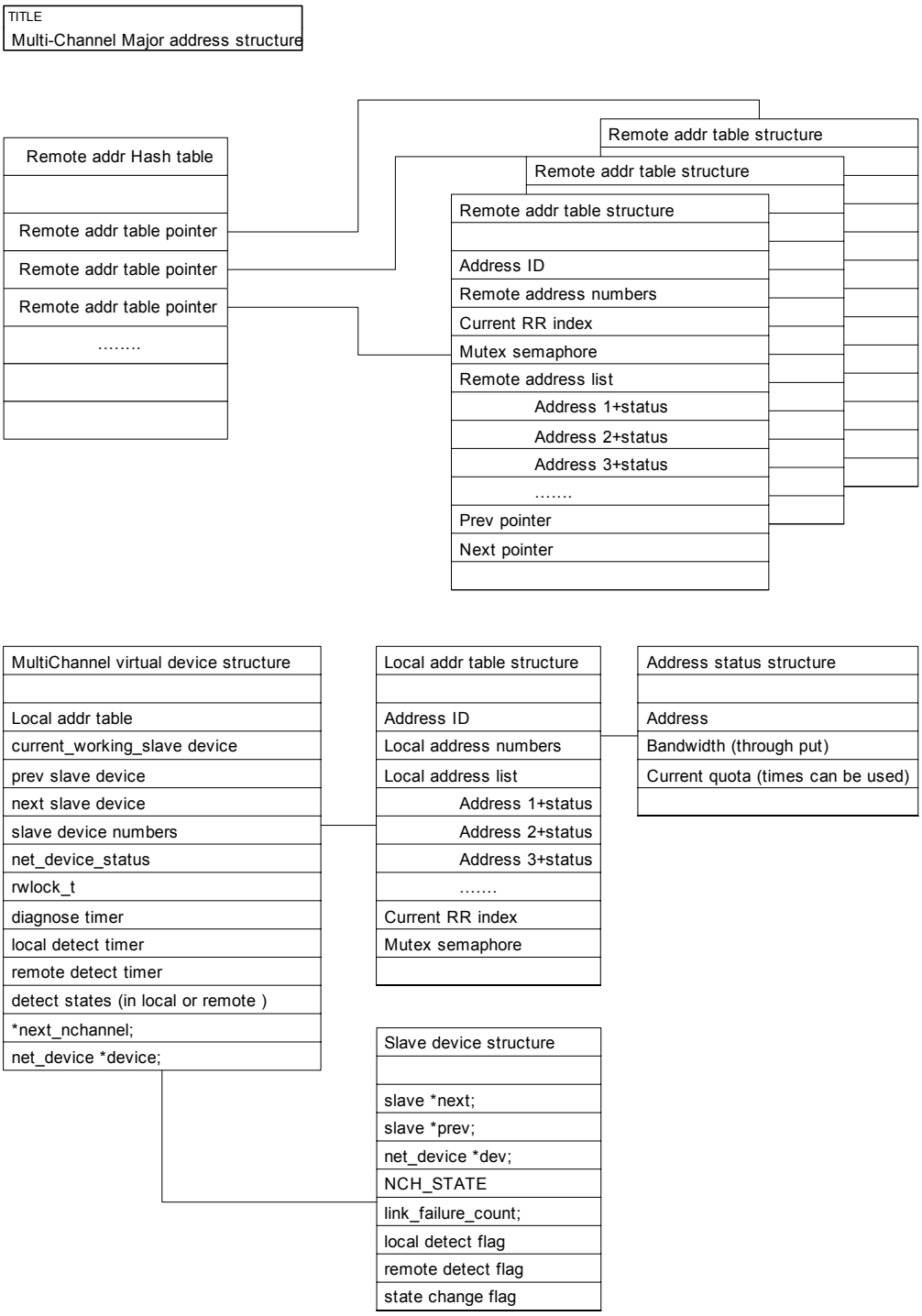

Figure 3.4 Multi-Channel Virtual Device and Hashing Address Table

The key data structure of the MAP-TP protocol is the Multi-channel virtual device and two address tables (Figure 3.4). The virtual device acts like a real network interface device and is in charge of multiple real slave NIC devices. With each slave

device, one local address entry is generated. The remote address table structure is built up in the procedure of address discovery where the bandwidth information is fed back.

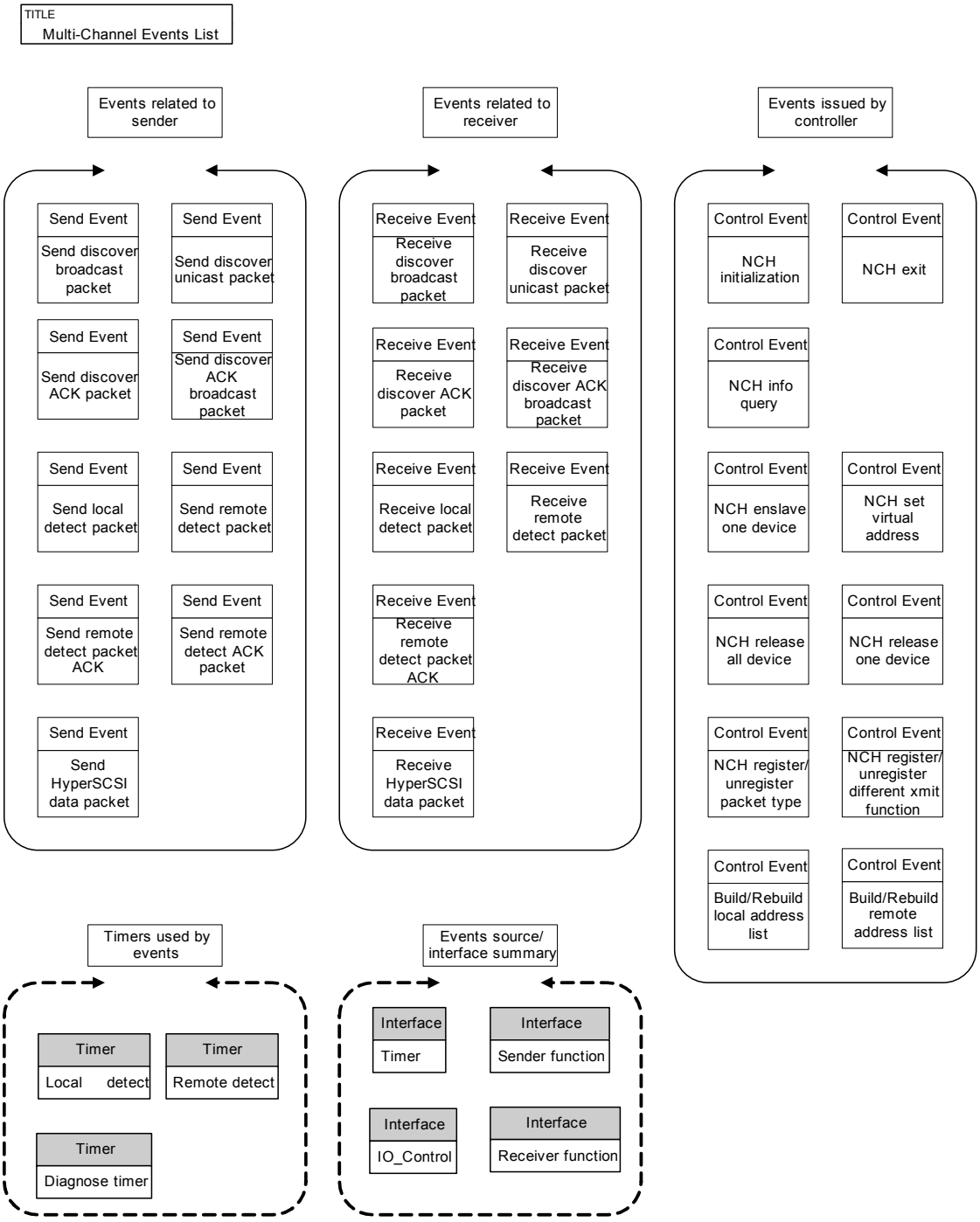

Figure 3.5 Events List for Sender, Receiver and Controller

The MAP-TP protocol data and control flow are driven by events. These events can be issued by system control module, in coming control packets or several timers. Here we list three group events and event sources.

3.2.2 Multi-Channel Packets Flow and Packet Types

There are two categories of packets in the MAP-TP protocol, data packets and control packets. As mentioned in previous sections, several types of control packets are used to exchange information and issue commands between devices.

The control packet types include: discover broadcast packet, discover unicast packet, discover ACK packet, discover ACK broadcast packet (don't need to reply), local detect packet, remote detect packet and remote detect ACK packet. Each type of packet will be issued by a sender event and this packet will cause a receiver event at the receiver device.

The data packets and control packets will be filtered and sent to different processes. The packet flow is illustrated in Figure 3.6.

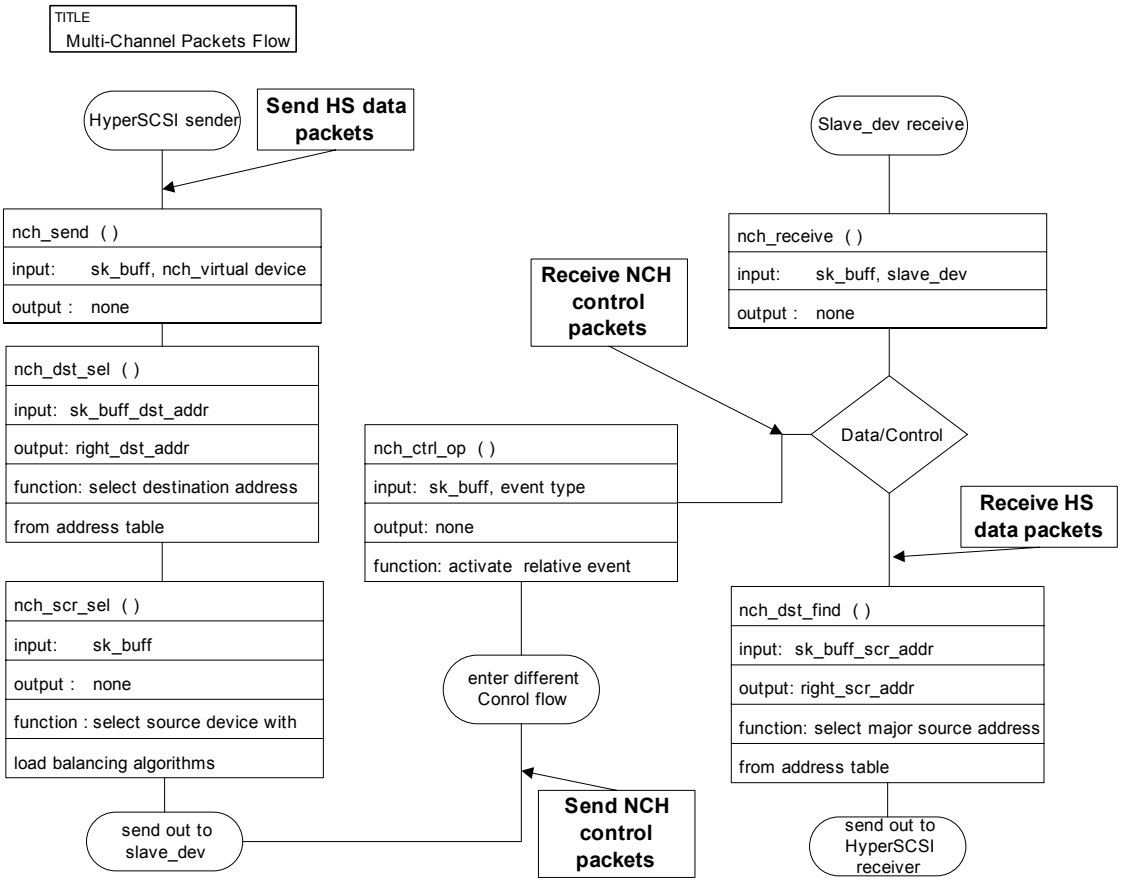

Figure 3.6 Multi-Channel Packet Flow

3.2.3 Multi-Channel Address Discover

After the initialization of the Multi-channel virtual device, the local address table is built up. Then, the local device broadcasts this address table to its neighboring hosts. When other devices receive this broadcast packet, they refresh their remote address table. After that, they send back an acknowledgment packet which includes their own local address table. Thus, all the devices will know each other's address tables (Figure 3.7).

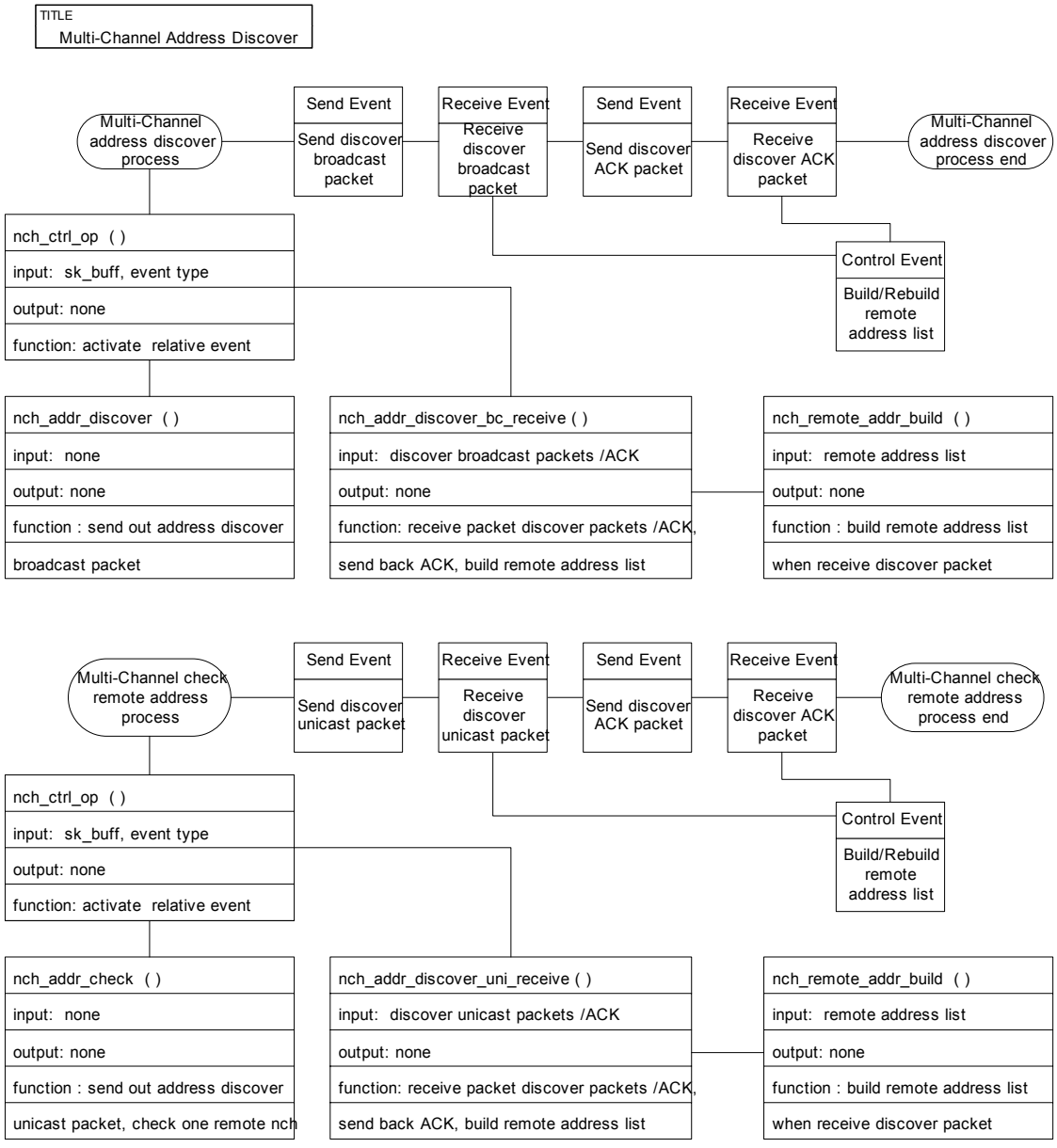

Figure 3.7 Multi-Channel Address Discover

The address discover packet can also be a unicast packet. This address-discover packet is used to enquire address table of a particular device (Figure 3.7). This method can be used where broadcast is inapplicable such as IP WAN.

3.2.4 Multi-Channel Address Table Change and Rebuild

From time to time, the address table can be changed under several conditions. For instance, one channel is added or deleted, channel fault is detected or channel fault is recovered. When the local address table is modified, the new local address table should be sent to its partner devices, and then these devices will update their remote address table (Figure 3.8).

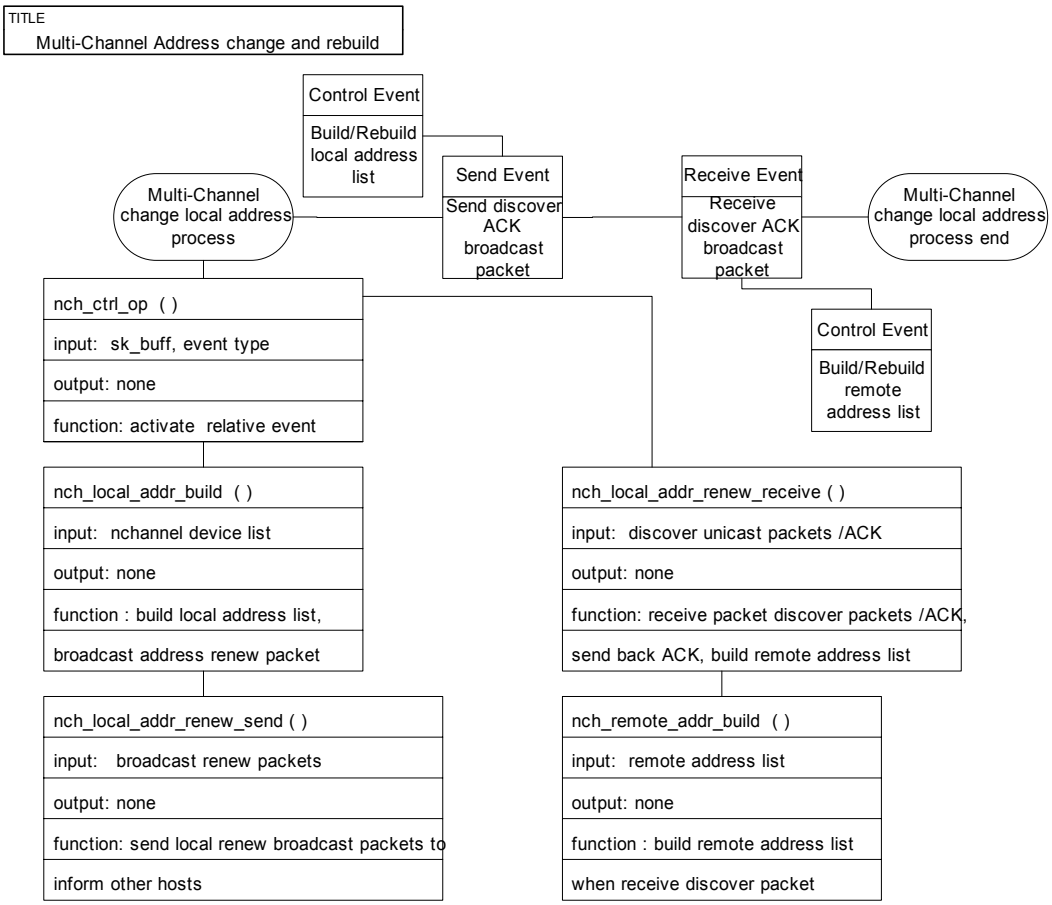

Figure 3.8 Multi-Channel Address Change and Address Table Rebuild

3.2.5 Multi-Channel Fault Detection Flow Chart

Fault tolerance is always a complicated part for network protocol design, because there are various of fault sources. A fault could be triggered by NIC interface fault, cable fault, switch fault or malfunctioned device driver. The Multi-channel fault detection flow is shown as Figure 3.9.

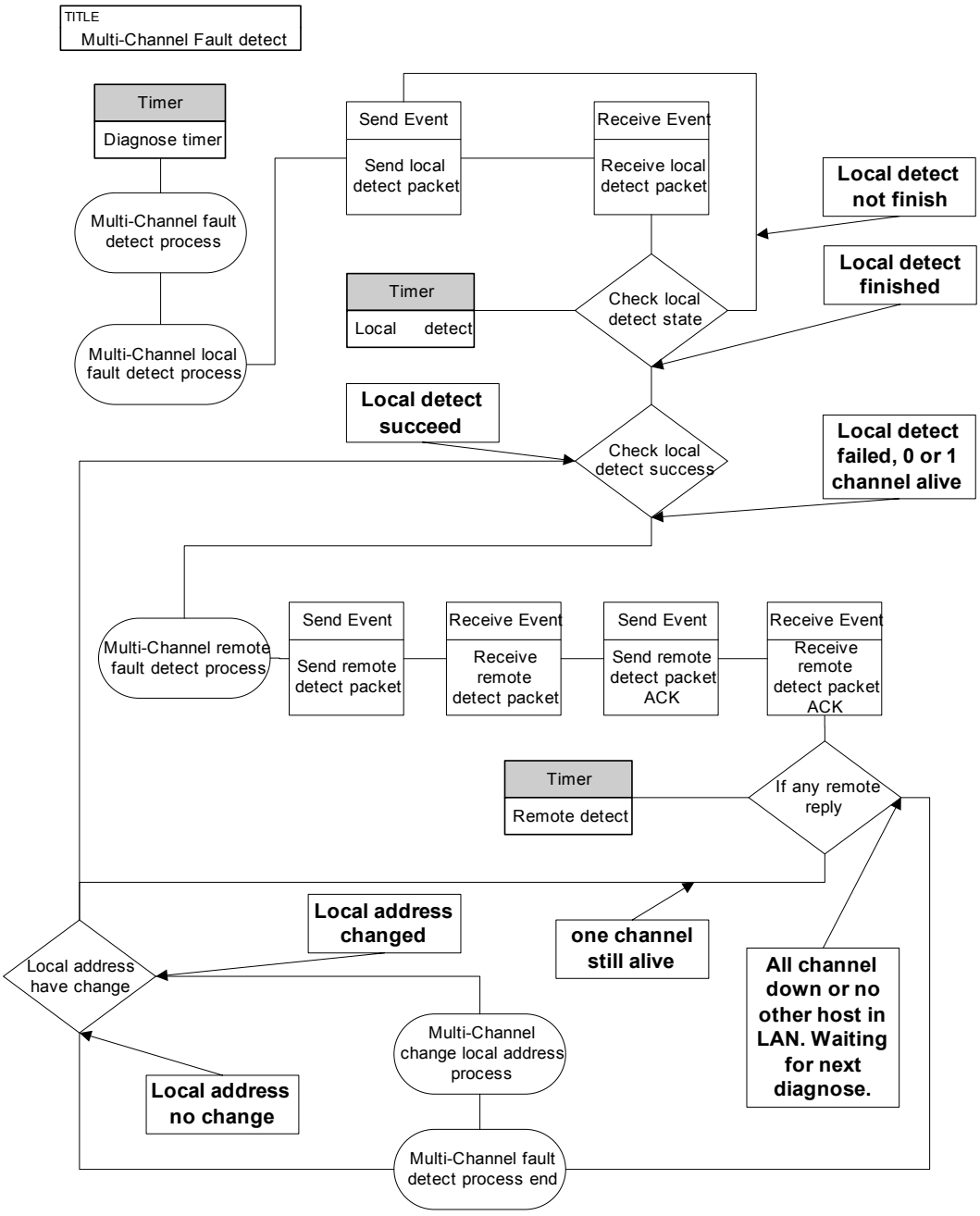

Figure 3.9 Flow Chart of Multi-Channel Fault Detect

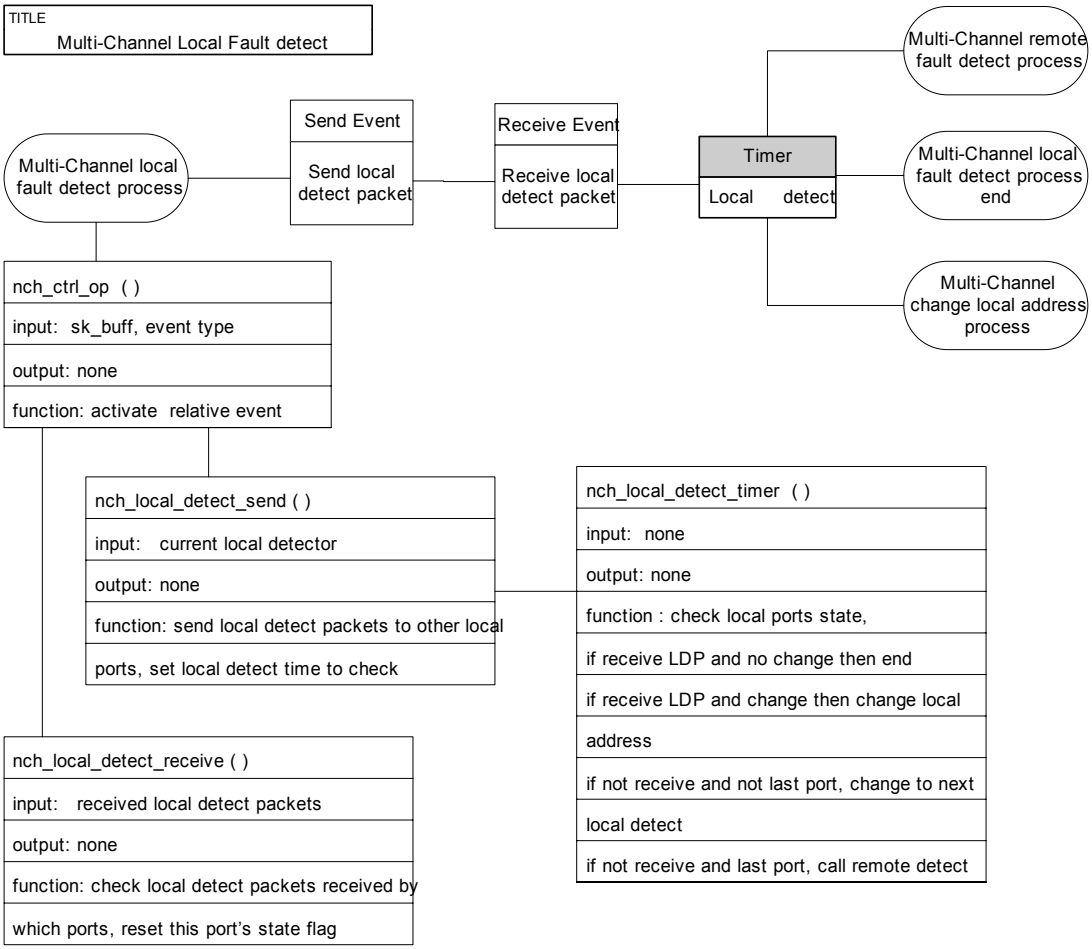

Figure 3.10 Multi-Channel Local Loop Back Detect

After checking the NIC hardware register, a local loop back fault detection solution is exploited as shown in Figure 3.10. The local loop back channels can be built by using NIC pairs on the same device. This can check all the cables, switches or other devices between two NICs. If any fault is found, the local address table will be updated and sent to other connected devices. If the fault can not be located by local loop back fault detection, a remote acknowledge fault detection might be issued.

Using local loop back detection can eliminate unnecessary detection load in the LAN, and it can also speed up the detection. Furthermore, since all loop back NICs are located on the same device, no acknowledge packets are required.

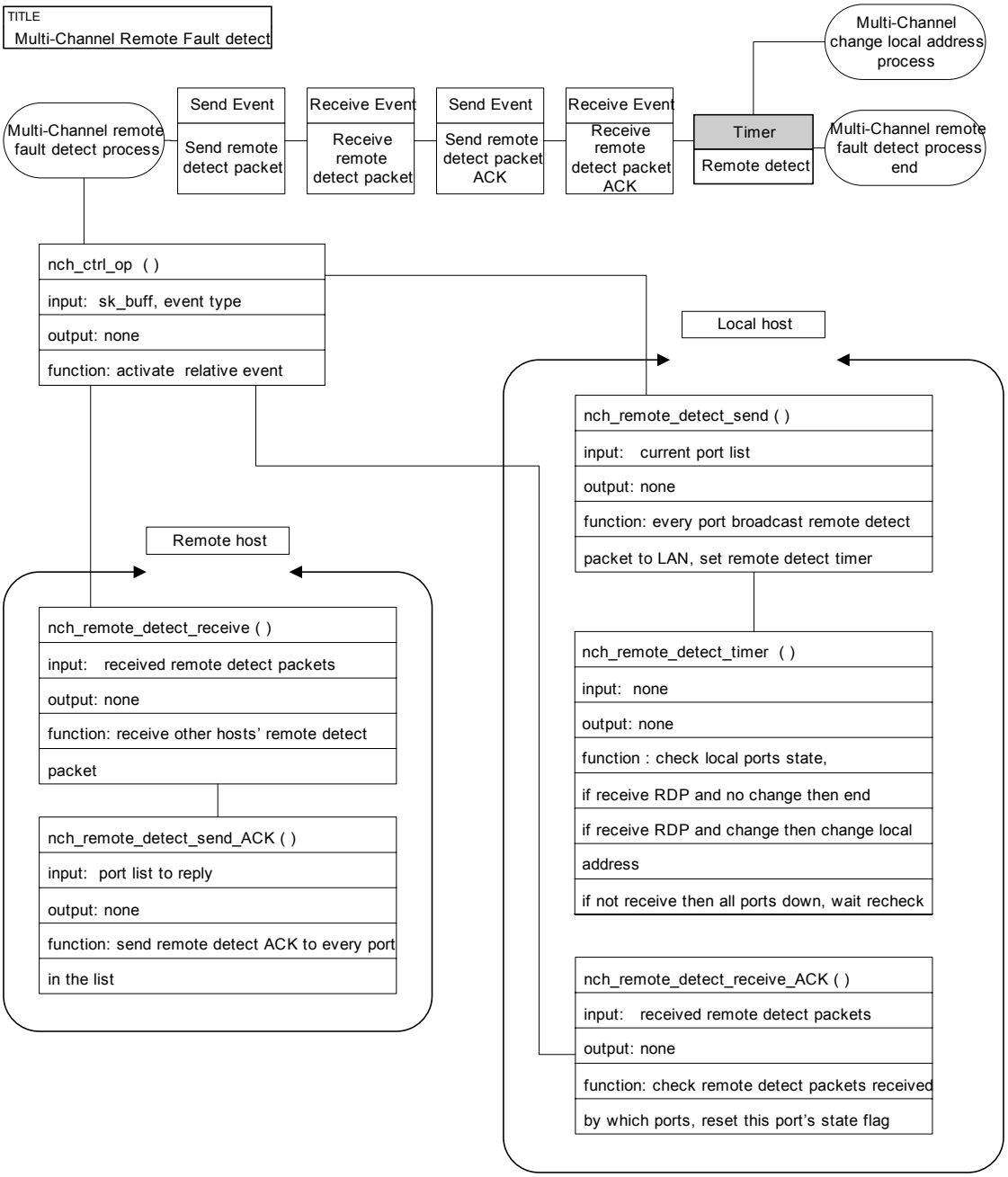

Figure 3.11 Multi-Channel Remote Acknowledge Fault Detect

The remote acknowledge detection is quite similar to the local loop back detection. The difference is that the neighboring devices are used to send back detection acknowledge packets as shown in Figure 3.11. This may consume more time and increase traffic load. However, the full data path is checked and a fault can be located even if there is only one NIC can communicate with other devices.

3.2.6 Virtual Device Management Flow Chart

The Multi-channel virtual device management flow chart involves four processes: Multi-channel startup process, exit process, channel management process and channel service process.

The startup process will register HyperSCSI packet type and transmit function handle, set up address, register NICs to Multi-channel, begin first address discovery. The exit process will do the reverse procedure as shown in Figure 3.12.

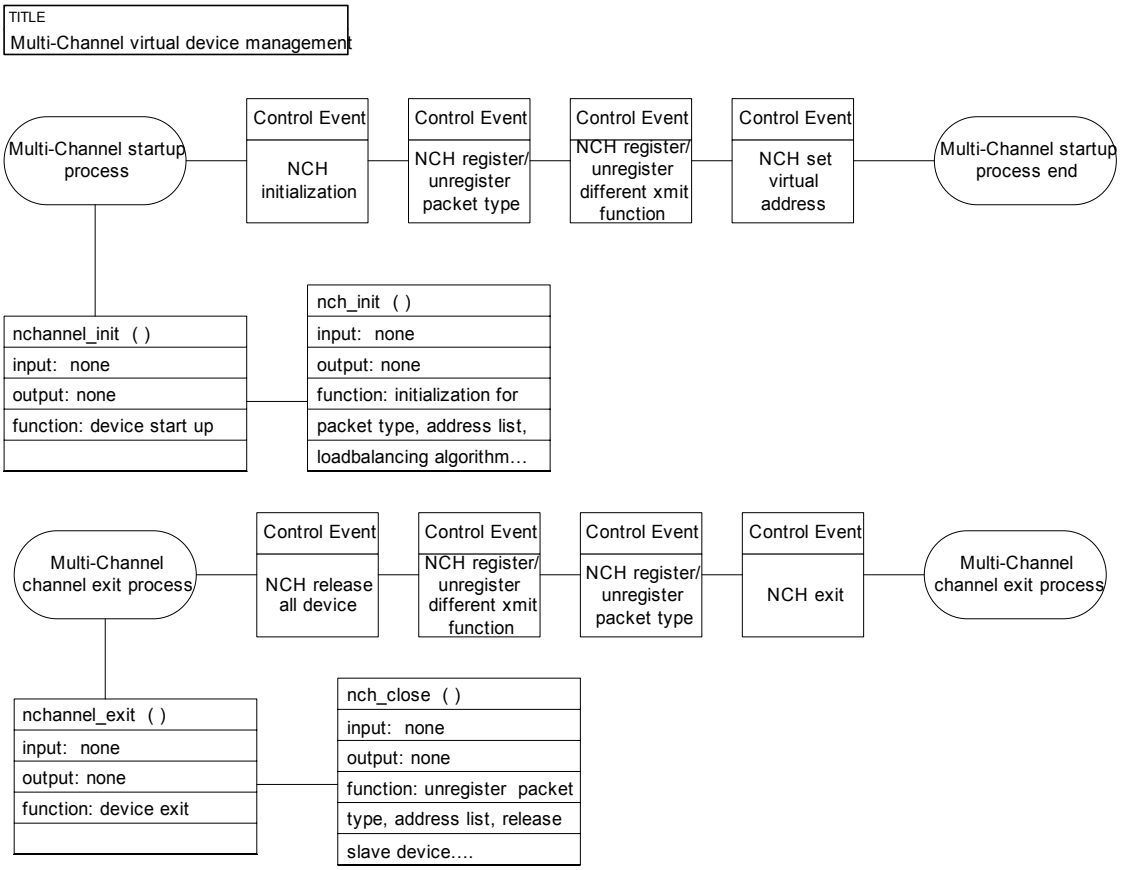

Figure 3.12 Multi-Channel Virtual Device Startup and Exit

The channel management process can dynamically add or remove slave devices to Multi-channel, update address table, and then synchronize neighboring devices' remote address table (Figure 3.13). The channel service process provides detailed

information for each NIC, manages the diagnose timer and monitors the status of Multi-channel system.

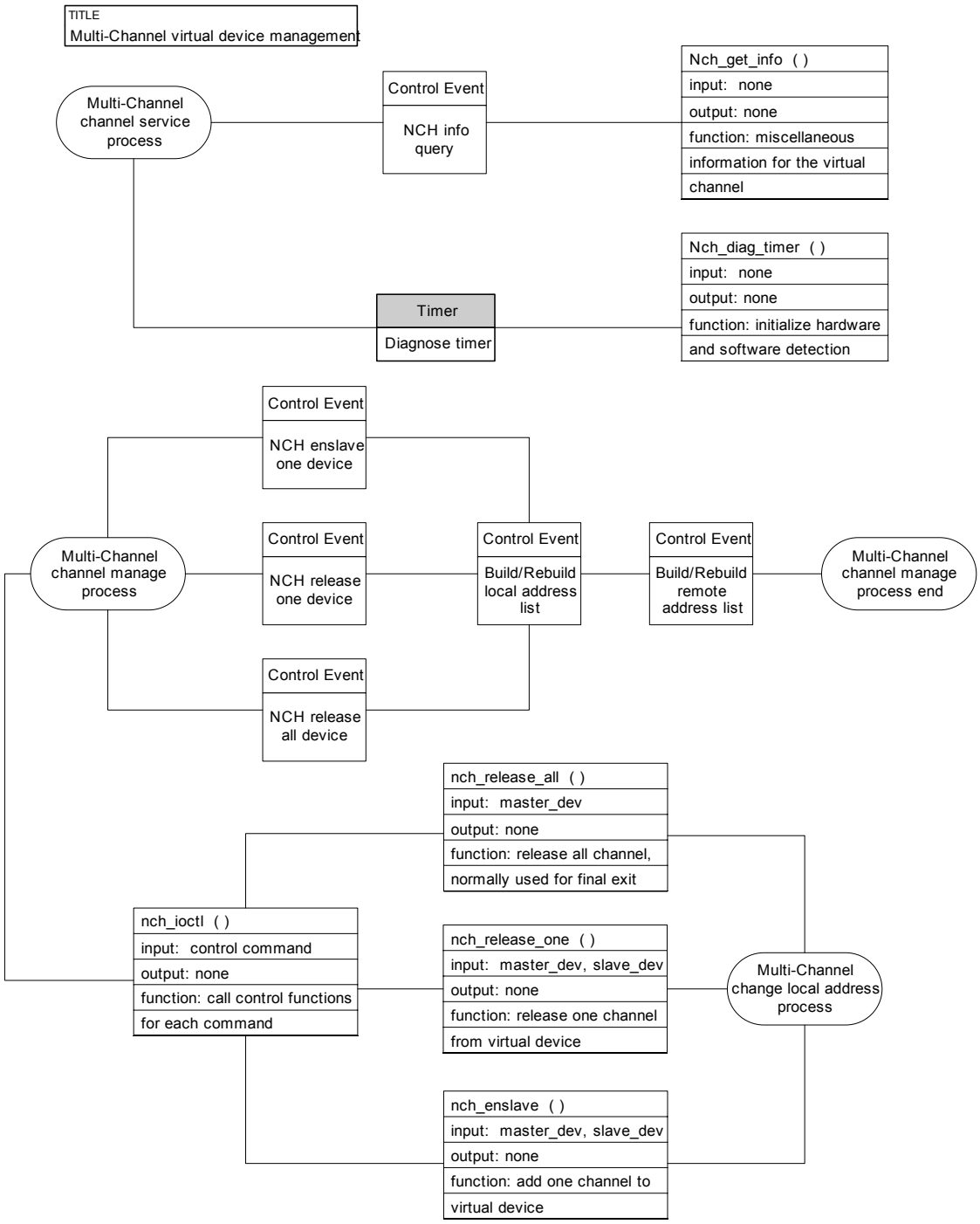

Figure 3.13 Channel Management and Channel Service

In this chapter, an implementation of multiple address parallel transmission protocol was proposed. The design of data module, control module and global state machine module was presented in detail. The protocol was programmed as a network layer module in Linux kernel. Furthermore, the major framework of the software was demonstrated by serial flow charts, which is a guideline for the implementations on other operating systems. After the MAP-TP protocol has been programmed, network storage applications can utilize it as a network layer interface, where the throughput and fault tolerance feature of the I/O can be measured and evaluated.

Chapter Four

4 Results and Discussion

As seen in Chapter 1, the HyperSCSI protocol provides a virtual SCSI interface service by packing SCSI protocol into network packets. We applied Multi-Channel on the network layer of the HyperSCSI storage device servers and clients. The packet switching network test platform was composed of switched Fast Ethernet (FE) and Gigabit Ethernet (GE) links. The computers were equipped with Intel Pentium III 1GHz, 256MB 133MHz SDRAM, two SCSI channels with 8 Seagate ST318406 LC Cheetah SCSI disks. The OS was RedHat Linux 7.21 (glibc RPM 2.2.4-24) with Linux Extended File System 2 (Ext2).

The performance was measured by testing the disk read speed of the virtual SCSI device. The throughput was bounded by the capability of the speed of the system bus and RAID bandwidth. Three popular benchmark tools, HDPARM, DD and IOZONE, were used to read 5GB data in the experiments.

HDPARM is a Linux shell utility for viewing and manipulating various IDE drive and driver parameters. The –t option means to report stats on the disk in question, reading data not in the cache. The command used is:

hdparm -tT <device>

DD copies a file with a user selectable block size, while optionally performing conversions on it. In each round, 1GB data is transmitted, so the result will reflect real disk read performance without the influence from cache size. The command used is:

time dd if=<device> of=/dev/null bs=4096 count=250000

IOZONE is a file system benchmark tool. The benchmark generates and measures a variety of file operations. A large file size of 500MB is also used to eliminate the influence of cache size. The command used is:

iozone -R -I 0 -I 1 -s 500M

In the following sections, the results of Multi-Channel disk read performance and out-of-order delivery will be discussed. Since many parameters map affect the result, experiments will be based on different channel numbers, channel bandwidths, packet sizes and system resources.

4.1 Disk Read Performance

4.1.1 Multi-Channel with Symmetrical Parallel Network

First a symmetrical parallel architecture of multiple FE NIC pairs is studied. Up to 7 pairs of FE cards are used to build a parallel transmission channel. To balance the computation load, the server provides separate virtual SCSI disks while the client utilizes these virtual disks to build software RAID.

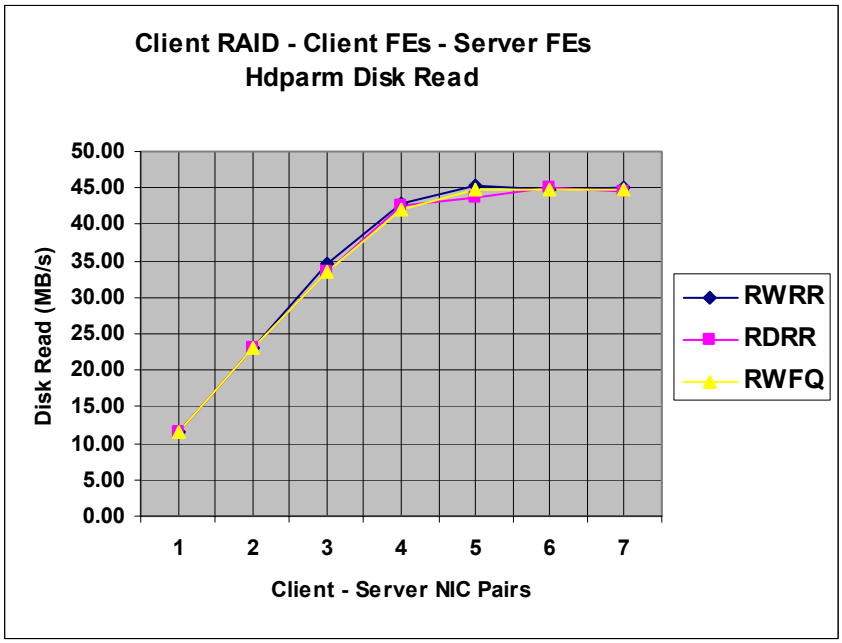

Figure 4.1 Client RAID Symmetrical FE Pairs HDPARM Read

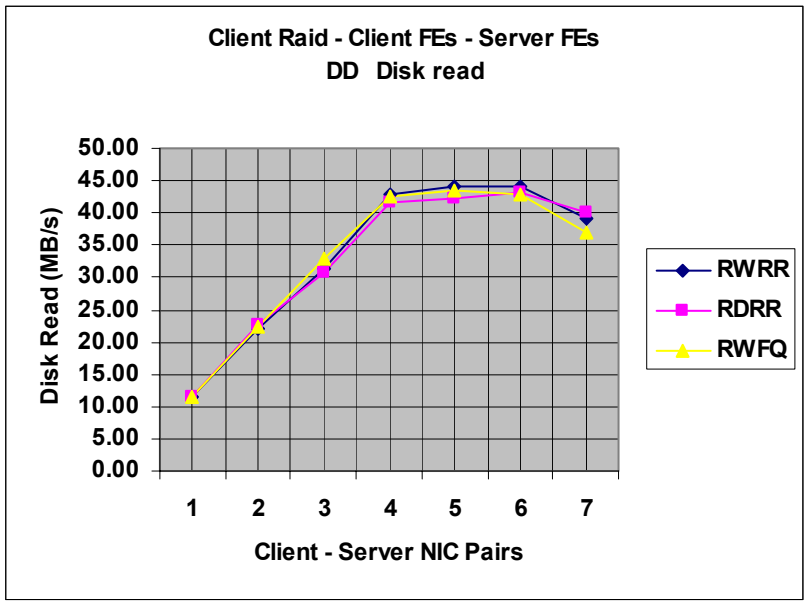

Figure 4.2 Client RAID Symmetrical FE Pairs DD Read

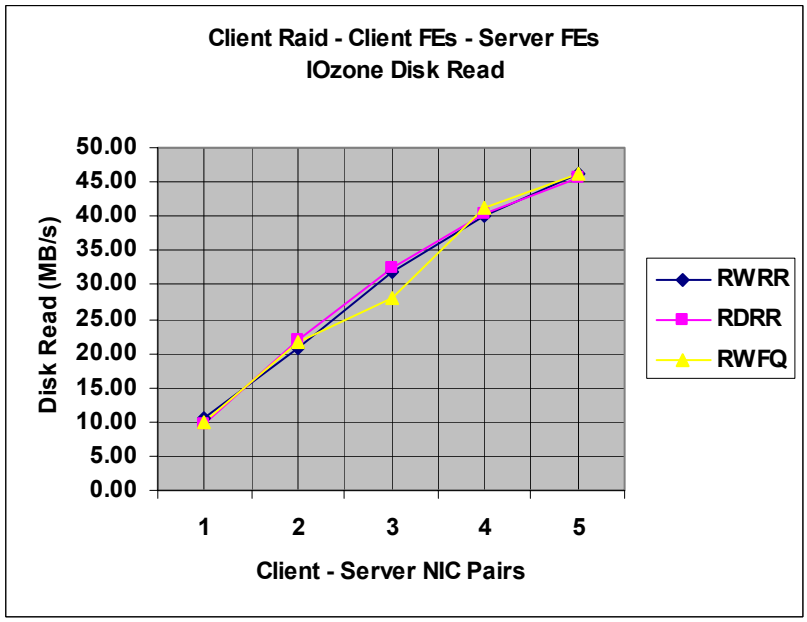

Figure 4.3 Client RAID Symmetrical FE Pairs IOZONE Read

From Figure 4.1, Figure 4.2 and Figure 4.3, with more sub channels added, the symmetrical disk read performance of the HyperSCSI device increases nearly linearly until the limit of the system performance is reached.

With all three scheduling algorithm RWRR, RDRR and RWFQ, HyperSCSI bandwidth can be increased above 380% when 4 links are used. Thus the network

utilization rate of the multiple channels HyperSCSI is 95%. The overhead caused by MAP-TP is roughly 5% per channel, which is acceptable when compared to the bandwidth increment.

Since most packets in the experiment have similar size, all three algorithms can achieve similar performance result. In real environment, if packets have various sizes, the RWFQ and RDRR can have better channel utilization. All these algorithms can be used for load balancing with little computation overhead.

4.1.2 Multi-Channel with Asymmetrical Parallel Network

To demonstrate the flexibility of the MAP-TP protocol, an asymmetrical parallel network was built for the Multi-Channel experiment. The server used seven FE cards while the client used only one GE network card. The RAID algorithm was also running on the client.

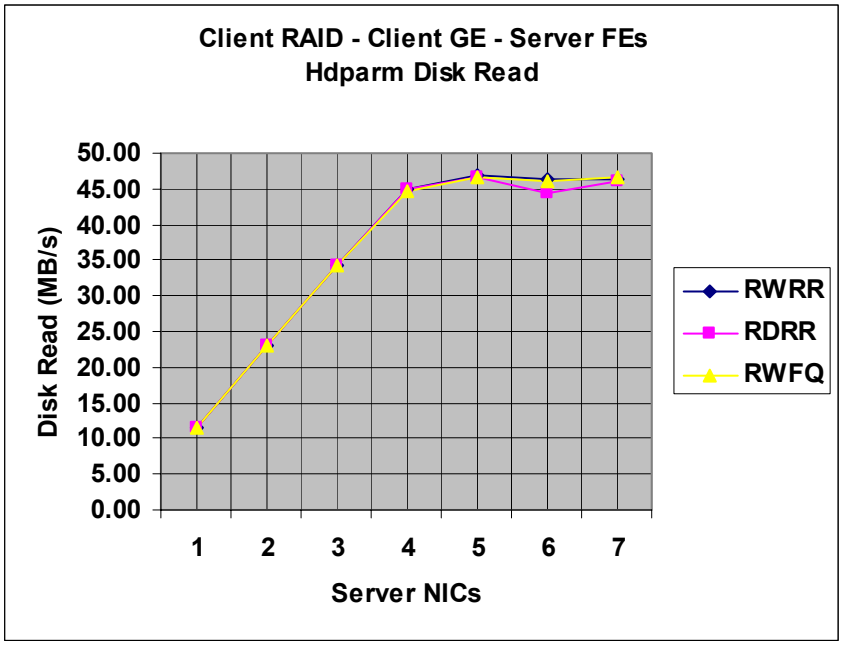

Figure 4.4 Client RAID Asymmetrical GE-FE Pairs HDPARM Read

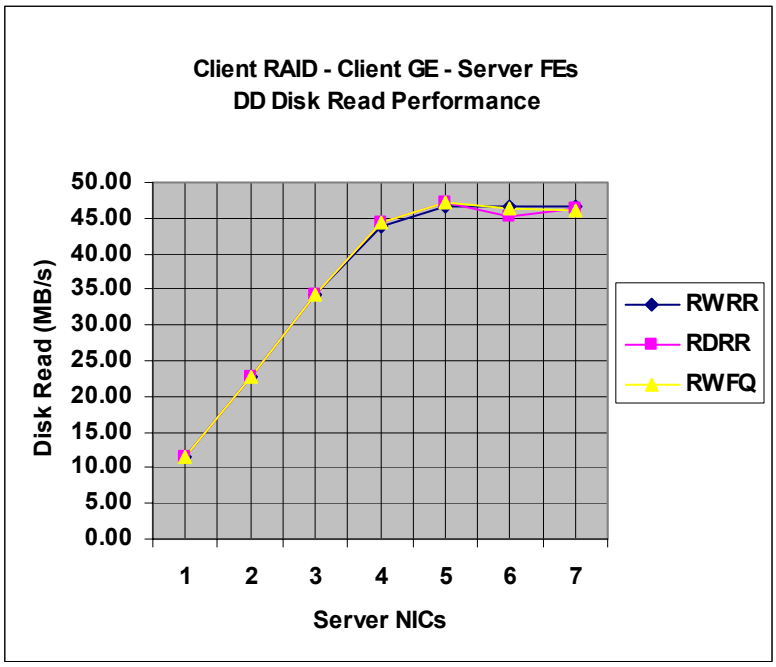

Figure 4.5 Client RAID Asymmetrical GE-FE Pairs DD Read

From Figure 4.4 and Figure 4.5, the MAP-TP protocol can achieve the same linear increment. Since this experiment uses high speed GE network interface cards, the asymmetrical transmission result is slightly better. The schedule load is the same with the symmetrical situation, while the GE network interface card does provide larger bandwidth.

4.1.3 Influence of System Resources on Disk Read Performance

In the experiments, the disk read performance approaches a limit when the network bandwidth keeps increasing. The reason of this limit is that the performance is constrained by the system resources. Besides the data transmission, the CPU rounds and cache memory are also consumed [87] by scheduling algorithms, SCSI command block packaging, RAID algorithms, network interface interruption handling, fault-handling, end-to-end flow control, and reliable transmission, etc. The distribution of these algorithms will influence the maximum disk read performance, however the increasing rate will keep constant.

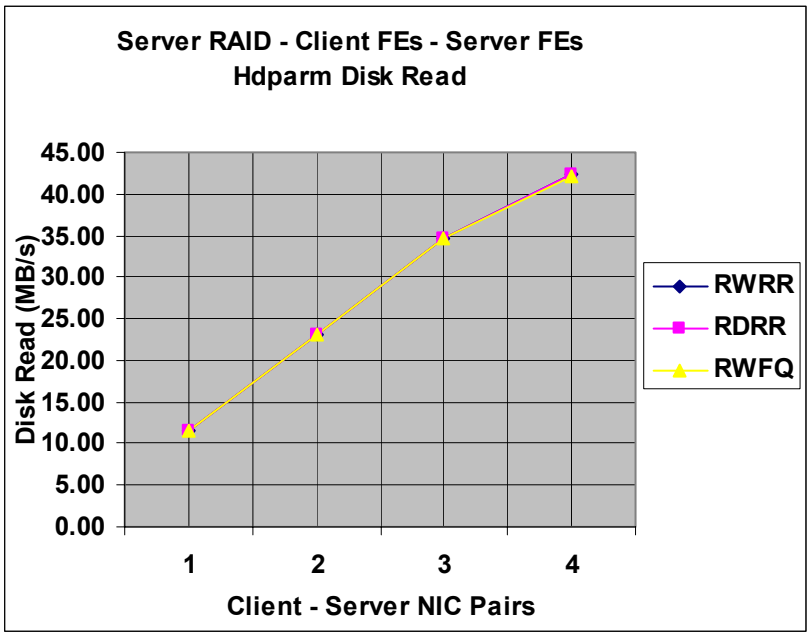

Figure 4.6 Server RAID Symmetrical FE Pairs HDPARM Read

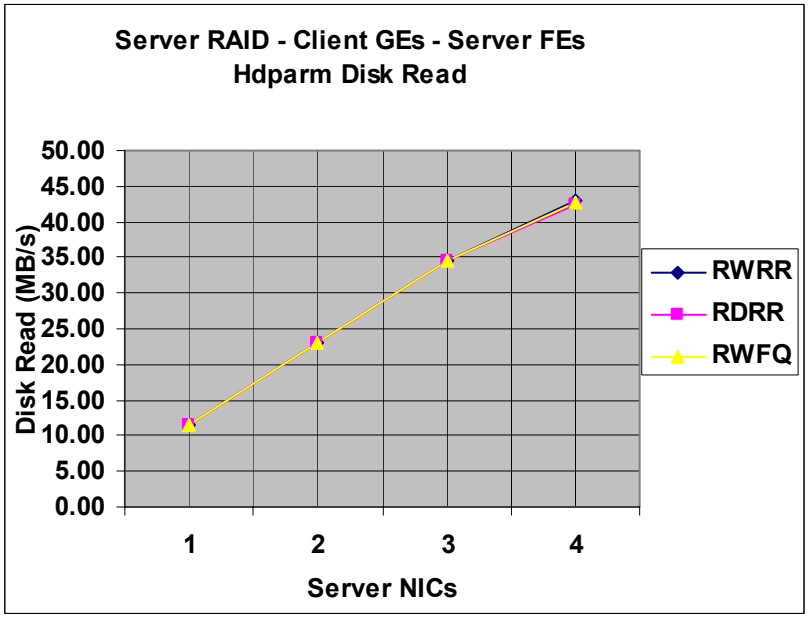

Figure 4.7 Server RAID Asymmetrical GE-FE Pairs HDPARM Read

When few FE NICs are used, the network bandwidth is the bottleneck of whole storage system. System resources can meet the requirement of network traffic load. The total computation cost is low and the performance can increase linearly when more bandwidth is provided. However, with the increasing bandwidth, the system

resources are gradually used up. The disk read performance will keep constant under heavy traffic load.

In previous sections, the RAID algorithm was applied on the client device. In this experiment, we apply RAID algorithms on the server device with various parallel architectures (Figure 4.6, Figure 4.7, Figure 4.8). For all these experiments, the increasing rate of disk read performances are linear when traffic load is low. However, the read performance ends up with different values when reaching the limit of system resource. These results show that the MAP-TP has good scalability. The performance is only bounded by total available system resource. By reallocating the system resources, maximum disk read performance can be improved without compromising the scalability.

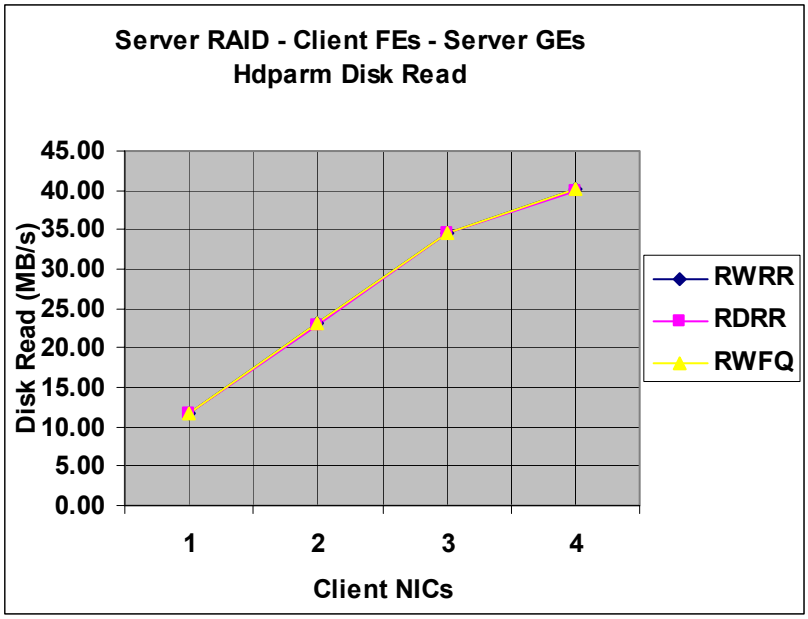

Figure 4.8 Server RAID Asymmetrical FE-GE Pairs HDPARM Read

4.2 Out-of-order Rate Analysis

In this section, we will discuss the architecture influence on the out-of-order rate and the relationship between the out-of-order rate and disk read performance. First a symmetrical FE pair parallel architecture was studied. In Figure 4.9 and Figure 4.10, the out-of-order rate decreases when more FE pairs are used. Although the decreasing curves differ for various benchmark tools, we can draw a conclusion that more identical channels lower the probability of out-of-order delivery.

The reason of the decrease is that in each schedule round, the schedule delay between sequential packets increases with more channels added. As proved in theoretical section, larger schedule delay between packets results in the small probability of out-of-order delivery.

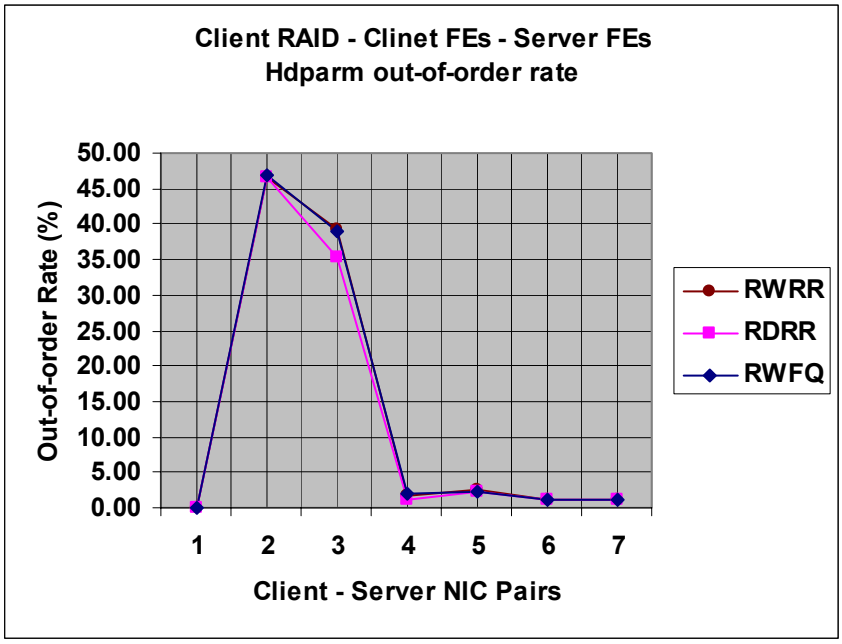

Figure 4.9 Client RAID Symmetrical FE Pairs HDPARM Out-of-order Rate

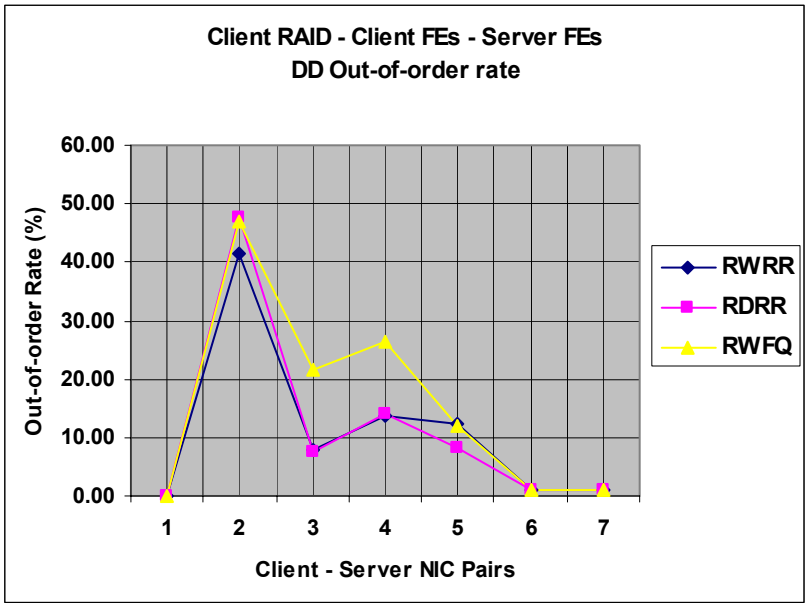

Figure 4.10 Client RAID Symmetrical FE Pairs DD Out-of-order Rate

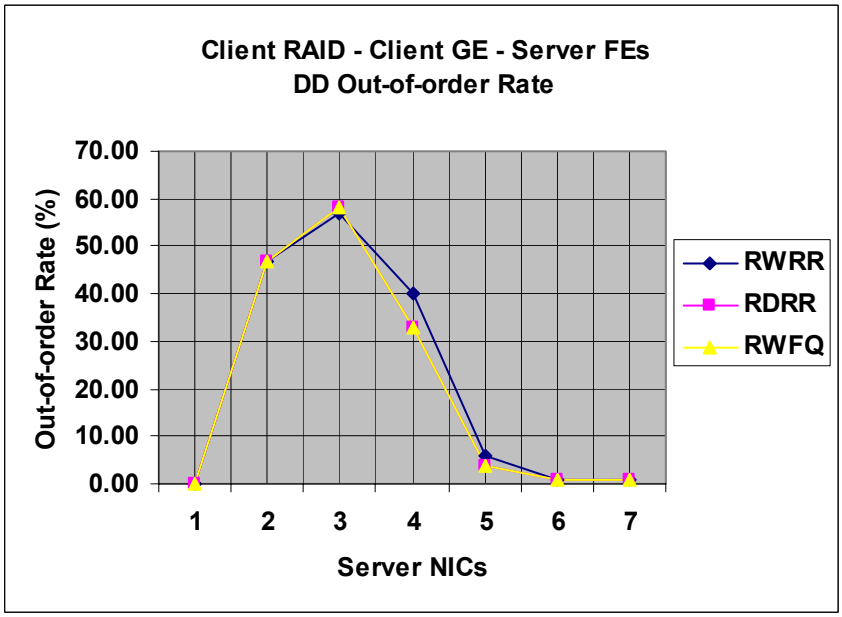

Figure 4.11 Asymmetrical GE-FE Pairs DD Out-of-order Rate

Second, an asymmetrical architecture was examined. The client device used one GE NIC and the server device used 7 identical FE NICs (Figure 4.11). The out-of-order rate curve provides similar result as symmetrical architecture. Since the client

device has wider bandwidth, the schedule delay is smaller than that of a symmetrical architecture. This makes the out-of-order rate curve decreasing slower.

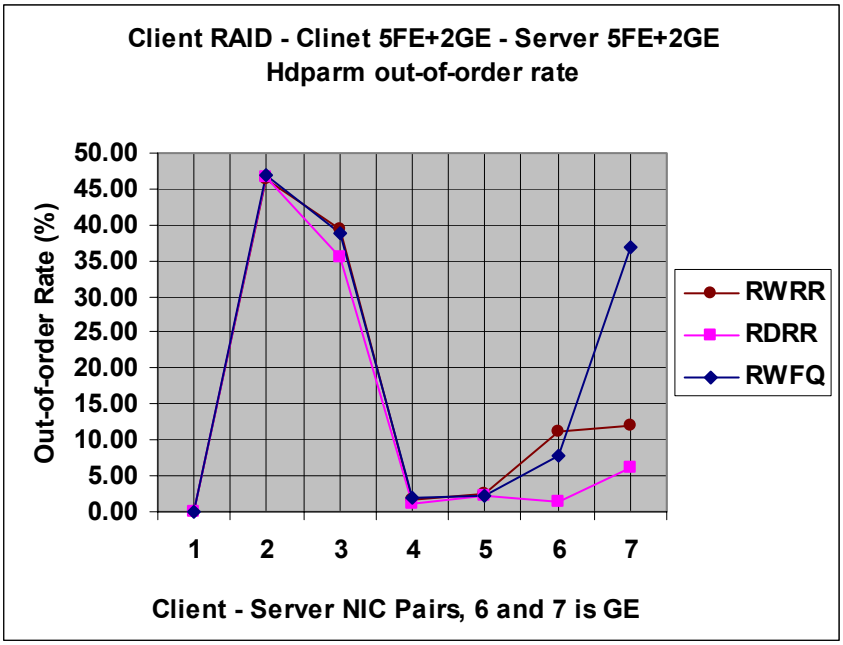

Figure 4.12 Asymmetrical GE-FE Mixed Pairs, HDPARM Out-of-order Rate

Finally, an asymmetrical architecture was built with mixed bandwidths. On each client and server device, 5 FE and 2 GE NICs were used (Figure 4.12). From the result, it is observed that the out-of-order rate increases with different bandwidth channels mixed together. This increment can also be explained by the changing of schedule delay. Since the GE channel is ten times faster than FE channel, packets on GE channels have shorter transmission time which means a shorter schedule delay between the packets.

From the discussion above, a conclusion can be drawn that the architecture of the parallel network has great impact on the out-of-order delivery. To decrease the out-of-order rate, we should use identical channels and increase the number of identical channels.

The relationship between performance and out-of-order rate can be found by comparing the figures of performance with that of the out-of-order.

Before the experiments, it is supposed that larger out-of-order rate might cause larger performance decrease. However, the result shows that the out-of-order rate does not evidently affect the disk read performance. Although the out-of-order rate is very large with a few channels, the performance keep increasing linearly and the channels are fully utilized.

This phenomenon can be explained by the out-of-order distance. Although the out-of-order rate is large, the out-of-order distance is only a few packets. So the receiver can reorder the data flow without affecting the disk read performance. In the theoretical part the out-of-order waiting buffer was already provided. Actually, the out-of-order rate only consumes certain system resource and causes a bounded delay. Whenever the system resource is adequate, the transmission performance will not be affected too much by the out-of-order delivery.

4.3 Packet Size's Affect on Out-of-order and Performance

In this section, the affect of packet size on out-of-order rate and transmission performance will be discussed. The experiment was designed working on maximum performance to demonstrate the influence of the packet size. The parallel channels employed two GE NICs. 1500 Maximum Transport Unit (MTU) and 9000 MTU packet size were studied respectively.

The influence of packet size on out-of-order rate is first checked. Figure 4.13 and Figure 4.14 show the maximum throughput of HDPARM and DD on 1500 MTU. The out-of-order rate results are similar with different scheduling algorithms and benchmark tools. For this two GE architecture, the 1500 MTU causes 43% out-of-order rate. While in Figure 4.15 and Figure 4.16, 9000 MTU causes only 15% out-of-

order rate. A conclusion can be drawn that larger packet size will decrease the out-of-order delivery. Again, this decrement can be explained by larger schedule delay which is caused by longer packet.

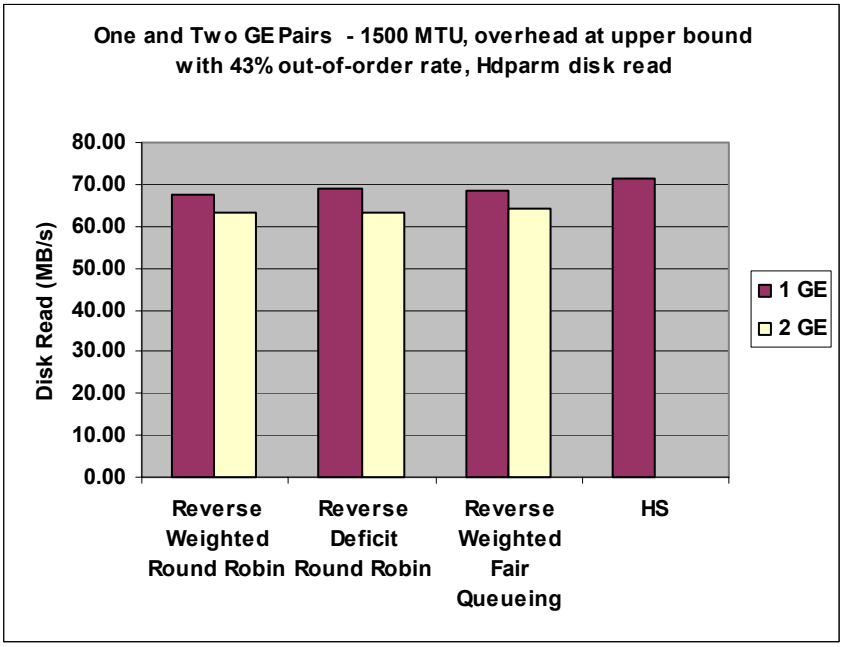

Figure 4.13 GE Pairs HDPARM Read, MTU = 1500, Out-of-order Rate = 43%

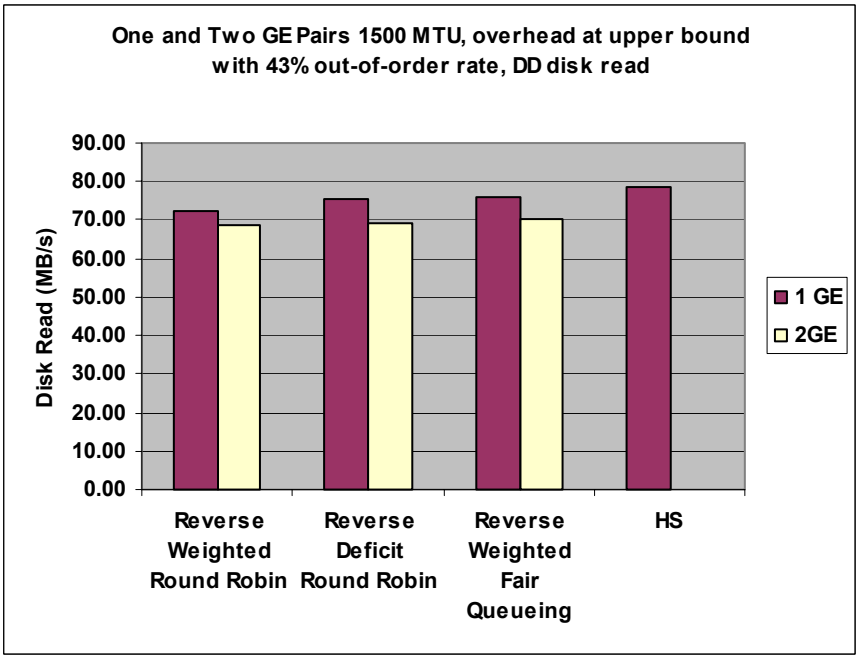

Figure 4.14 GE Pairs DD Read, MTU = 1500, Out-of-order Rate = 43%

Next the performance was examined. Compared with 1500 MTU, the 9000 MTU has better maximum performance. One reason is that the packet packaging computation is more efficient for larger packet size. The other reason is that the reordering overhead also decreases with a small out-of-order rate.

The computation overhead of the out-of-order delivery is illustrated in this experiment. Since the single channel maximum performance was provided, the difference between the single GE and double GE can only be caused by scheduling algorithms and reordering algorithms. It is found that the difference is larger when 1500 MTU is used. This can be explained that larger out-of-order rate needs more computation power.

As a conclusion, we suggest using larger packet size whenever the network supports.

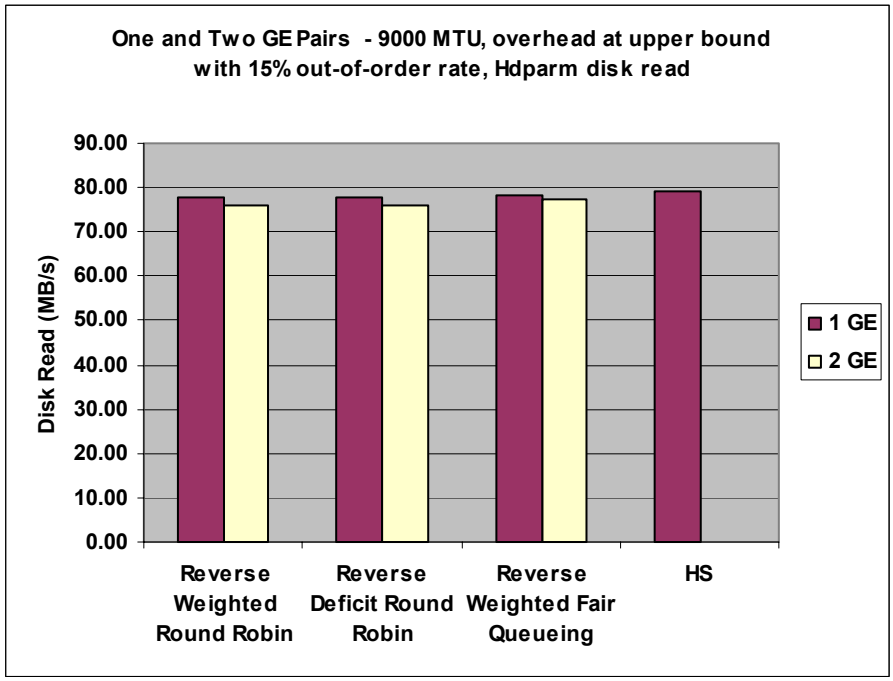

Figure 4.15 GE Pairs HDPARM Read, MTU = 9000, Out-of-order Rate = 15%

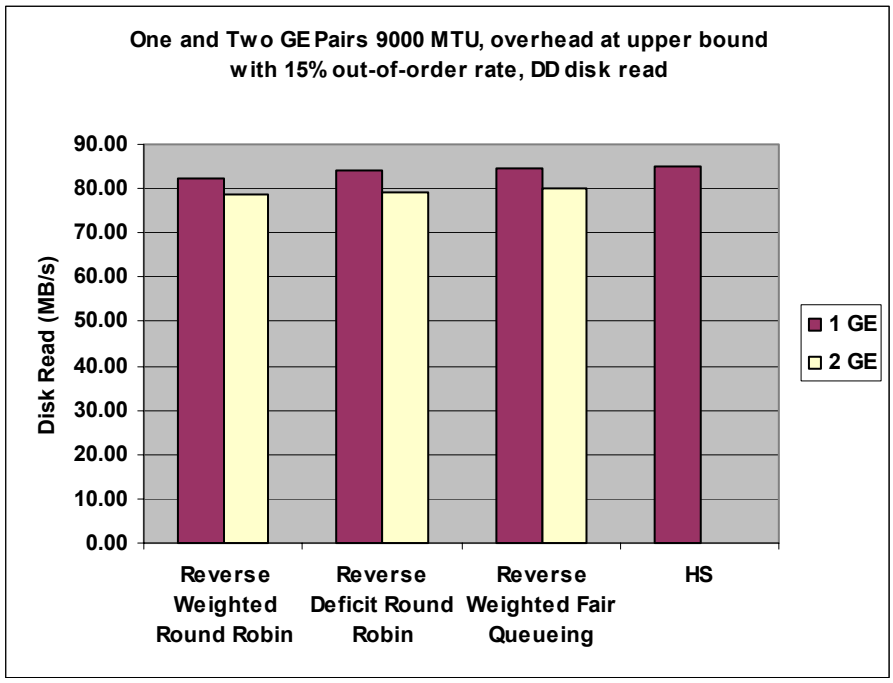

Figure 4.16 GE Pairs DD Read, MTU = 9000, Out-of-order Rate = 15%

4.4 Multiple Clients - Multiple Channels Load Balancing

In the previous sections, the experiments are based on multiple channels on one pair of client and sever. In this section, the multiple clients and multiple channels load balancing will be studied, where one server with GE and two clients with multiple FEs were employed.

Figure 4.17, Figure 4.18 and Figure 4.19 show that with all three scheduling algorithms, the traffic load is evenly balanced on both clients and channels. Since RAID and multiple devices management work load were distributed on two clients, the system resource could be fully utilized. The sum of the disk read performance could approach the maximum throughput when 6 FEs were used.

In this experiment, not only does the network topology have influence on the performance, the distribution of the system resource is also critical for achieving better throughput.

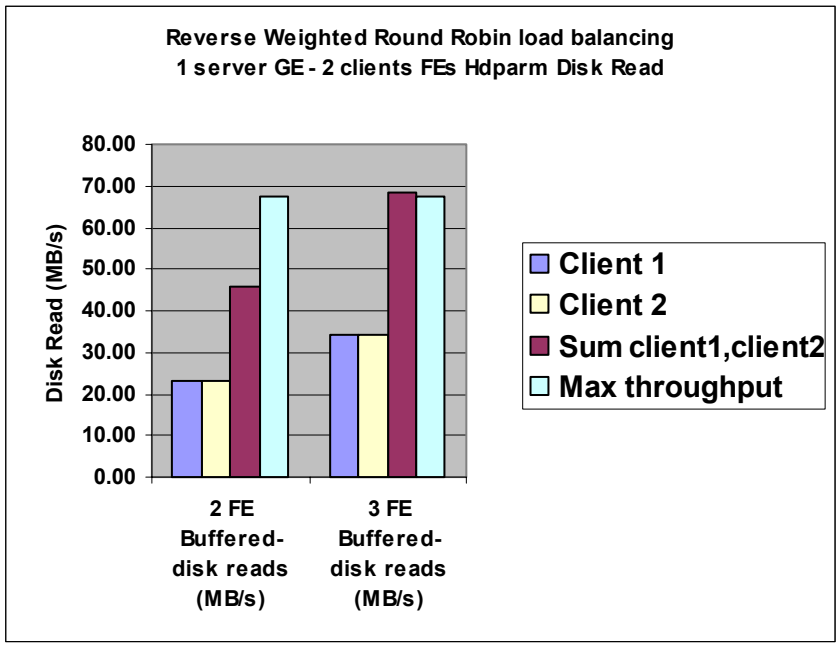

Figure 4.17 RWRR Load Balancing on Two Clients with GE-FE Pairs

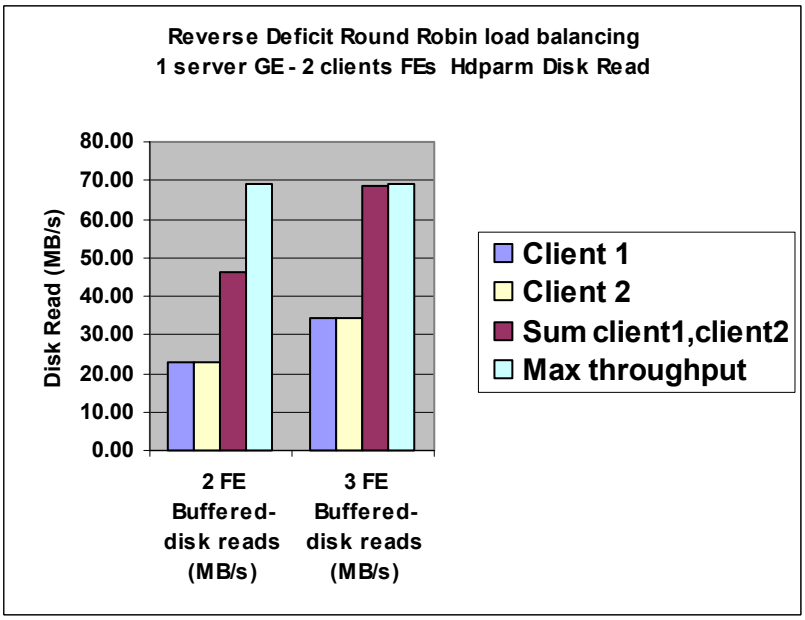

Figure 4.18 RDRR Load Balancing on Two Clients with GE-FE Pairs

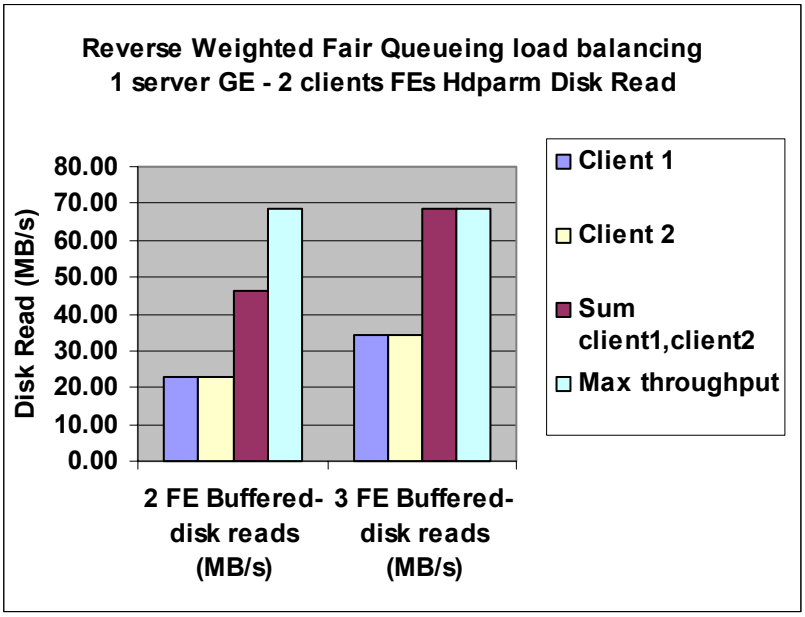

Figure 4.19 RWFQ Load Balancing on Two Clients with GE-FE Pairs

In this chapter, the throughput of the MAP-TP was measured by testing the disk read of the HyperSCSI network storage device. Parallel symmetrical and asymmetrical architectures were studied under three scheduling algorithms, RWRR, RDRR and RWFQ. The result showed that HyperSCSI bandwidth could be increased above 380% when 4 channels were employed. By analyzing the system resource distribution and out-of-order rate, it was found that MAP-TP has good scalability continuously within the range of maximum performance. Furthermore, the packet size was studied to improve the throughput and reduce the out-of-order rate. The last experiment demonstrated that the traffic load can be balanced on multiple nodes and channels.

Chapter Five

5 Conclusions and Future Work

In this thesis, we provide a theoretical model and an implementation demonstrating the multiple address parallel transmission architecture. In this chapter, we first summarize the major results and then discuss some of the future work of parallel architecture.

In chapter 2, the theories of the network topology and parallel data transmission model were discussed. The SAN topology requires working in both LAN and WAN by supporting Ethernet MAC and IP packets. The storage devices in SAN may have variable bandwidth and packet sizes.

Based on packet switching network, an end-to-end parallel data transmission architecture was designed. SAN topology using this structure will have good scalability and fault tolerance but without much additional complexity. The model of this transmission architecture is composed of two stage load balancing algorithm, data flow priority algorithm, data flow restoring, reordering and fault tolerance algorithm.

On sender device, four theoretical load balancing algorithms, RGPS, RWRR, RDRR and RWFQ, were deduced from reverse Fair Queuing algorithm. For different network environment and data transmission models, we have shown that RWRR, RDRR and RWFQ can be utilized and implemented with small computation overhead. Then we proved that one virtual channel is better than multiple sub channels and used service discipline algorithm to provide different priorities and rate-control for each data flow.

On receiver device, data flows are restored and reordered. A chaining hashing algorithm was used to search senders' address table with bounded delay. An efficient

out-of-order estimation algorithm was used to setup retransmission threshold and allocate reordering buffer. The mechanism of the out-of-order in parallel architecture differs from a normal network. Different bandwidths and packet sizes are the major reason of this out-of-order delivery. The out-of-order rates were analyzed by schedule delay and out-of-order distance.

Since the end-to-end parallel transmission architecture uses redundant channels, the scheduler can provide fault tolerance by detecting channel fault and rescheduling data flows to usable channels. The fault detection algorithm can use either hardware based, software local loop back and/or remote acknowledge detection methods. With different computation complexity, the failure of the first hop link, the nearest switch and the full data path could be detected.

To demonstrate the multiple addresses parallel transmission protocol (MAP-TP) in SAN, we implemented the protocol as network layer in the Linux kernel module of the HyperSCSI in Chapter 3.

The MAP-TP protocol includes three major function modules: data module, control module and a global state machine module. The data module is responsible for sending and receiving data flow packets through multiple network channels. The control module is used to manage multiple channels and selecting load balancing algorithm. The fault tolerance is also provided in the control module. The global state machine inspects global parameters and synchronizes the control and data transmission states transition. Each major module is composed of group of processes and events, and they were illustrated in detail by software flow charts.

In Chapter 4, the parallel transmission performance was measured by testing the disk read speed of the virtual SCSI device with three popular benchmark tools, HDPARM, DD and IOZONE. Different parallel symmetrical and asymmetrical

architectures were studied under all three scheduling algorithms, RWRR, RDRR and RWFQ.

The result shows that HyperSCSI bandwidth can be increased above 380% when 4 links are used. Thus the network utilization rate of the multiple channels HyperSCSI is 95%, which is acceptable for high speed transmission. Interestingly, by analyzing results of the out-of-order rate, it is found that out-of-order rate did not affect the disk read performance very much, which could be explained by the theory of schedule delay and out-of-order distance.

To improve transmission performance and decrease out-of-order rate, the network architecture and system resource distribution must be designed wisely. We suggest using identical channels on one device, increasing the number of identical channels and utilizing larger packet size whenever possible.

As a conclusion, this multiple address parallel transmission architecture supports flexible performance scaling in a packet switching network. As such, SAN topology using this structure will have good scalability and fault tolerance but without much additional complexity. In addition, by supporting Ethernet MAC and IP packets, the network storage devices can work in both LAN and WAN environments. This architecture could be a key factor in deploying SANs, which can provide a highly scalable bandwidth with full redundancy over switched, parallel data paths.

5.1 Extensions

In this thesis, simple priority service discipline was employed on the scheduler. However, service disciplines which can provide per-connection end-to-end performance guarantee are still unavailable along the path. In future, parallel packet

switching network will have to support QoS services that allow clients to transport information with performance guarantee.

Thus, future research work may focus on QoS applications in parallel transmission packet switching SAN, where the delay model of parallel transmission is quite different from normal end-to-end transmission. Characteristics such as, delay, delay jitter, throughput and loss rate should be studied to control the rate of each data flow and avoid the congestion. Traditional congestion control methods did not consider about the parallel architecture where out-of-order delivery is quite prevalent.

In this thesis, features of out-of-order delivery has been studied which is expressed by characteristics such as, retransmission, window size, drop rate estimation, and buffer size. More study should be conducted on delay and delay jitter caused by out-of-order delivery. Both single and multiple channel end-to-end queuing models should be analyzed by factors of out-of-order delivery.

Additionally, parallel architecture should be studied on emerging Data Grids, which could be fundamental in future data-intensive applications such as geographically dispersed extraction of complex scientific information from very large collections of measured or computed data. The reliability and performance of the GridFTP might be enhanced by applying Multi-channel technology on striping and partial file access.

Last but not the least, high level data management interface should be studied for the implementation of the parallel SAN architecture. A new standard, the Storage Management Initiative Specification (SMI-S) by Storage Network Industry Association (SNIA), is rapidly emerging as the architecture for fully interoperable SANs. SMI-S is intended to be the unifying interface between storage objects that

must be managed and the management applications. It is necessary to trace this standard in future research works.

References

[1] M.A. Marsan and D. Roffinella, "Multichannel local area network protocol," IEEE, J. Sel. Areas in Commun., vol. SAC-1, pp. 885-897, Nov. 1983.

[2] Bill Ham, Digital Equipment, "Parallel SCSI Grows, Shrinks and Stays the Same," www.scsita.org, 1997.

[3] Paul H. Fredette, "The Past, Present, and Future of Inverse Multiplexing," IEEE Comm. Magazine, vol. 32, no. 4, pp. 42, 6 April 1994.

[4] James P. G. Sterbenz, Josehp D. Touch, "High-Speed Networking, a Systematic Approach to High-Bandwidth Low-Latency Communication," published by John Wiley & Sons, Inc, 2001.

[5] "Link Aggregation," IEEE Std 802.3, 2000 Edition.

[6] SUN Trunking Software.http://www.sun.com .

[7] Quanlong Ding, Soung C. Liew, "a Performance Analysis of a Parallel Communications Scheme for ATM Networks," Proc. of IEEE Globecom'95, pp. 898--902, 1995.

[8] Patrick Beng T. KHOO and Wilson Yong H. WANG, "Introducing A Flexible Data Transport Protocol for Network Storage Applications," 10th NASA Mass Storage Systems and Technologies Conference / 19th IEEE Symposium on Mass Storage Systems, Apr. 2002.

[9] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, S. Tuecke, "Data Management and Transfer in High Performance Computational Grid Environments," Parallel Computing Journal, vol. 28, no5, pp. 749-771, May 2002.

[10] Brian L. Tierney, Dan Gunter, Jason Lee, Martin Stoufer, ”Enabling Network-Aware Applications,” Tenth IEEE International Symposium on High Performance Distributed Computing, pp. 281, Aug. 2001.

[11] Hari Adiseshu, Guru Parulkar and George Varghese, “A Reliable and Scalable Striping Protocol,” Proc. ACM SIGCOMM, pp. 131-141, Aug. 1996.

[12] Hari Adiseshu, George Varghese and Guru Parullkar, “An Architecture for Packet-Striping Protocols,” ACM Transactions on Computer Systems, vol. 17, no. 4, pp. 249-287, Nov. 1999.

[13] Josep M.Blanquer and Banu Ozden, “Fair Queuing for Aggregated Multiple Links,” Proc. Sigcomm, pp. 189-197, 2001.

[14] Jorge A.Cobb, “An In-Depth Look at Flow Aggregation for Efficient Quality of Service,” IEEE International Conference on Network Protocols, pp. 127, Nov. 1999.

[15] Jorge A.Cobb, “Preserving Quality of Service Guarantees in Spite of Flow Aggregation,” IEEE/ACM Transactions on Networking, vol.10, no. 1, pp. 43-53, 2002.

[16] Jorge A. Cobb and Miaohua Lin, “End-to-End Delay Guarantees for Multiple-Channel Schedulers,” IEEE International Workshop on Quality of Service (IWQoS), May 2002.

[17] C. Brendan S. Traw and Jonathan M. Smith, “Striping within the Network Subsystem,” IEEE Network, pp. 22--32, Jul./Aug. 1995.

[18] Seungjae Han, Kang G. Shin, “Efficient Spare-Resource Allocation for Fast Restoration of Real-Time Channels form Network Component Failure,” IEEE Real-Time Systems Symposium, 1997.

[19] Colin J. Parris and Domenico Ferrar., ”A Dynamic Connection Management Scheme for Guaranteed Performance Services in Packet-Switching Integrated Services Networks, ” Tenet Technical Report TR-93-005, Computer Science Division, University of California at Berkeley, 1993.

[20] Michael O. Rabin, “Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance,” Journal of ACM, vol. 36, no.2, pp. 335-348, Apr. 1989.

[21] Xiaohu Qie, Andy Bavier, Larry Peterson, and Scott Karlin, “Scheduling Computations on a Software-Based Router**,”** Proc. of the 2001 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, pp. 13-24, 2001.

[22] Hari Adiseshu, Guru Parulkar and George Varghese, “Reliable FIFO Load Balancing over Multiple FIFO Channels,” tech. rep., Washington University, St. Louis, May 1995.

[23] Haining Wang, Chia Chen, and Kang G. Shin, “Adaptive-Weighted Packet Scheduling for Premium Service,” Proceedings of IEEE International Conference on Communications, 2001.

[24] Bhumip Khasnabish and Majid Ahmadi, “In Search of a Fair Scheduling Policy for Serving Multi-Queue System,” IEEE, 0-7803-1443-3/93, 1993.

[25] Taeck-Geun Kwon, Sook-Hyang Lee and June-Kyung Rho, “Scheduling Algorithm for Real-Time Burst Traffic Using Dynamic Weighted Round Robin,” IEEE, 1998.

[26] Idris A. Rai and Murat Alanyali, “Uniform Weighted Round Robin Scheduling Algorithms for Input Queued Switches,” Communications, 2001. ICC 2001. IEEE International Conference, Vol. 7, pp. 2028-2032, 2001.

[27] Sungwon Ha, Kang-Won Lee and Vaduvur Bharghavan, “Performance Evaluation of Scheduling Algorithms in an Integrated Packet Services Network Environment,” Proc. of ISCC'98, Athens, Greece, Jul. 1998.

[28] Ian R. Philp and Jane W.S. Liu, “End-to-End Scheduling in Real-Time Packet-Switched Networks,” Proc. of IEEE International conference on Network Protocols (ICNP’96), pp. 23, Oct./Nov. 1996.

[29] Hemant M. Chaskar and U. Madhow, “Fair Scheduling with Tunable Latency: A Round Robin Approach,” Proc. of IEEE/ACM Transactions on Networking , vol. 11, no. 4, pp. 592-601, Aug. 2003.

[30] Jong-Seon Kim and Daniel C. Lee, “Weighted Round Robin Packet Scheduler Using Relative Service Share,” Proc. IEEE MILCOM'2001, McLean, VA, Oct. 2001.

[31] Dimitrios Stiliads and Anujan Varma, “Design and Analysis of Frame-based Fair Queueing: A New Traffic Scheduling Algorithm for Packet-Switched Networks,” Proceedings of the 1996 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, pp. 104-115, 1996.

[32] Yao-Tzung Wang, Tzung-Pao Lin and Kuo-Chung Gan, “An Improved Scheduling Algorithm for Weighted Round Robin Cell Multiplexing in an ATM Switch,” Technical Report, Industrial Technology Research Institute, 1994.

[33] Hideyuki Shimonishi, Makiko Yoshida, Ruixue Fan and Hiroshi Suzuki, “An Improvement of Weighted Round Robin Cell Scheduling in ATM Networks,” Proc. GLOBECOM'97, pp1119-1123, 1997.

[34] Norio Matsufuru and Reiji Aibara, “Efficient Fair Queueing for ATM Networks Using Uniform Round Robin,” Proc. IEEE INFOCOM'99, 1999.

[35] Amitava Raha, Nicholas Malcolm and Wei Zhao, “Hard Real-Time Communications with Weighted Round Robin Service in ATM Local Area Networks,” IEEE 0-8186-7123-8/95, 1995.

[36] Yoshihiro Ito, Shuji Tasaka and Yutaka Ishibashi, “Variably Weighted Round Robin Queueing for Core IP Routers,” Proc. IEEE IPCCC'02, pp. 159-166, Apr. 2002.

[37] A. Demers, S. Keshav and S. Shenker, “Analysis and Simulation of a Fair Queueing Algorithm,” Proc. of ACM SIGCOMM, pp. 1-12, Sep. 1989.

[38] Abhay K. Parekh and Robert G. Gallager, “A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case,” IEEE/ACM, Transactions on Networking, vol.1, no. 3, pp. 344-357, Jun. 1993.

[39] Abhay K. Parekh and Robert G. Gallager, “A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case,” Proc. of the INFOCOM'93, pp. 521-530, Mar. 1993.

[40] Jon C.R. Bennett and Hui Zhang, “WF^2Q: Worst-case Fair Weighted Fair Queueing,” Proc. of IEEE INFOCOM 96, pp. 120-128, Mar. 1996.

[41] S. Jamaloddin Golestani, “A Self-Clocked Fair Queueing Scheme for Broadband Applications,” Proc. of IEEE INFOCOM'94, pages 636--646, Jun. 1994.

[42] Dimitrios Stiliadis and Anujan Varma, “Rate-Proportional Servers: A Design Methodology for Fair Queueing Algorithms,” IEEE/ACM Transactions on Networking, Apr. 1998.

[43] Dimitrios Stiliadis and Anujan Varma, “Latency-Rate Servers: A General Model for Analysis of Traffic Scheduling Algorithms,” IEEE/ACM Transactions on Networking, Oct. 1998.

[44] Jaesun Cha, Jisoo Park, Changhwan Oh, and Kiseon Kim, “A Time-Based Weighted Fair Queueing Algorithm for Improving CDV and CLP in ATM Networks,” IEEE TENCON'99. Vol. II, pp. 840-843, Sep. 1999.

[45] Yuhua Chen and Jonathan S. Turner, “Design of a Weighted Fair Queueing Cell Scheduler for ATM Networks,” Proc. of IEEE GLOBECOM 98, Sydney, Australia, Nov. 1998.

[46] Yoshihiro Ohba, “QLWFQ: A Queue Length Based Weighted Fair Queueing Algorithm in ATM Networks,” Proc. IEEE IN- FOCOM'97, Kobe, Japan, pp. 567-576, Apr. 1997.

[47] Chin-Chang Li, Shiao-Li Tsao, Meng Cheng Chen, Yeali Sun and Yueh-Min Huang, “Proportional Delay Differentiation Service Based on Weighted Fair Queuing,” Proc. IEEE Int. Conf. Computer Communications and Networks (ICCCN), pp. 418-423, Oct. 2000.

[48] Andrea Francini and Fabio M. Chiussi, “A Weighted Fair Queueing Scheduler with Decoupled Bandwidth and Delay Guarantees for the Support of Voice Traffic,” Proc. of GlobeCom, vol. 3, 2001.

[49] Mong-Fong Horng, Wei-Tsong Lee, Kuan-Rong Lee and Yau-Hwang Kuo, “ An Adaptive Approach to Weighted Fair Queue with Qos Enhanced on IP Network,” Proc. IEEE TENCON 2001, Aug 2001.

[50] Albert Banchs and Xavier Perez, “Distributed Weighted Fair Queuing in 802.11 Wireless LAN,” Proc. of IEEE ICC2002, Apr./May. 2002.

[51] Dimitrios Stiliadis and Anujan Varma, “Efficient Fair Queueing Algorithms for Packet-Switched Network,” IEEE/ACM Transaction on Networking, vol. 6, no. 2, pp.175-185, Apr. 1998.

[52] Nam-Seok Ko and Hong-Shik Park, "Emulated Weighted Fair Queuing Algorithm for High-Speed Packet-Switched Network," Proc. of the 15th International Conference on Information Networking (ICOIN'01), pp. 52, Jan./Feb. 2001.

[53] Jose R. Gallardo and Dimitrios Makrakis, "Dynamic Predictive Weighted Fair Queueing for Differentiated Services," IEEE International Conference on Communications (ICC 2001), Helsinki, Finland, Jun. 2001.

[54] M. Shreedhar and George Varghese, "Efficient Fair Queuing using Deficit Round Robin," Proc. of the conference on Applications, technologies, architectures, and protocols for computer communication, vol. 25, no. 4, pp. 231-242, 1995.

[55] L. Lenzini, E. Mingozzi and G. Stea, "Aliquem: a Novel DRR Implementation to Achieve Better Latency and Fairness at *O*(1) Complexity," Proc. of the 10th International Workshop on Quality of Service (IWQoS 2002), pp. 77-86, Miami Beach, FL, May 15-17, 2002.

[56] M. H. MacGregor, W. Shi, "Deficit for Bursty Latency-critical Flows: DRR++," Proc. IEEE ICON 2000, Singapore, pp. 287-293, Sept. 2000.

[57] Salil S. Kanhere and Harish Sethu, "Fair, Efficient and Low-Latency Packet Scheduling Using Nested Deficit Round Robin," Proc. of the IEEE Workshop on High-Performance Switching and Routing (HPSR), Dallas, Texas, USA, May 2001.

[58] Salil S. Kanhere and Harish Sethu, "On the Latency Bound of Deficit Round Robin," Proc. of the IEEE International Conference on Computer Communications and Networks, Miami, Florida, USA, Oct. 2002.

[59] Onur Altintas, Yukio Atsumi and Teruaki Yoshida, "A Note on Fair Queueing and Best-Effort Service in the Internet," Proc. of 1999 Internet Workshop, pp. 145, Osaka University Convention Center Suita, Osaka, Japan, Feb. 1999.

[60] Go Hasegawa, Takahiro Matsuo, Masayuki Murata and Hideo Miyahara, “Comparisons of Packet Scheduling Algorithms for Fair Service among Connections on the Internet,” Journal of High Speed Networks, vol.12, no. 1,2, pp.1-28, 2002..

[61] Salil S. Kanhere and Harish Sethu, “Fair, Efficient and Scalable Scheduling Without Per-Flow State,” Proc. of the IEEE International Performance, Computing and Communications Conference, Phoenix, Arizona, USA, Apr. 2001.

[62] Kimihiro Yamakoshi, Eiji Oki and Naoaki Yamanaka, “Dynamic Deficit Round-Robin Scheduler for 5-Tb/s Switch Using Wavelength Routing,” Proc. of High Performance Switching and Routing, May 2002.

[63] Salil S. Kanhere, Harish Sethu and Alpa B. Parekh, “Fair and Efficient Packet Scheduling Using Elastic Round Robin,” IEEE Transactions on Parallel and Distributed Systems vol. 13, no. 3, pp. 324-336 March 2002.

[64] J. M. Arco, D. Meziat and B. Alarcos, “A Suitable Service Discipline for ATM-Ethernet Interconnection,” ECUUM´2000, Colmar (France), pp. 87-92, Oct. 2000.

[65] Hari Adiseshu, Guru Parulkar and George Varghese, “Reliable FIFO Load Balancing over Multiple FIFO Channels,” tech. rep., Washington University, St. Louis, May 1995.

[66] Dimitri Bertsekas and Robert Gallager, “Data Network,” Prentice-Hall, 1992.

[67] V. Jacobson, “Congestion Avoidance and Control,” Proc. of the SIGCOMM '88, pp. 314- 329, Aug. 1988.

[68] W. Stevens, “TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms,” Internet RFC, Jan. 1997.

[69] Sally Floyd and Kevin Fall, "Promoting the Use of End-to-End Congestion in the Internet," IEEE/ACM transactions on networking, vol. 7, no. 4, pp. 458-472, Aug. 1999.

[70] Hui Zhang, "Service Disciplines for Guaranteed Performance Service in Packet-Switching Networks," Proc. of the IEEE, 83(10), pp. 1374-1396, Oct. 1995.

[71] Pawan Goyal and Harrick M. Vin, "Generalized Guaranteed Rate Scheduling Algorithms: A Framwork," IEEE/ACM Transactions on Networking, vol. 5, no. 4, pp. 561-571, 1997.

[72] David D. Clark and David L. Tennenhouse, "Architectural Consideration for a New Generation of Protocols," In SIGCOMM Symposium on Communications Architectures and Protocols**,** ACM, pp. 200-208, Philadelphia, PA, Sep. 1990.

[73] Vern Paxson, "End-to-End Internet Packet Dynamics," IEEE/ACM Transactions on Networking, vol. 7, no. 3, pp. 277-292, Jun. 1999.

[74] Feng Wang and Yongguang Zhang, "Improving TCP Performance over Mobile Ad-Hoc Networks with Out-of-Order Detection and Response**,"** Proc. of the 3rd ACM international symposium on Mobile ad hoc networking & computing, pp. 217-225**,** 2002.

[75] Dmitri Loguinov and Hayder Radha, "Large-Scale Experimental Study of Internet Performance Using Video Traffic," ACM SIGCOMM Computer Communication Review (CCR), vol. 32, no. 1, pp. 7-19, Jan. 2002.

[76] Sally Floyd, "Re: TCP and Out-of-Order Delivery," PILC Mailing list, pilc.lerc.nasa.gov, 1999.

[77] Jon C. R. Bennett, Craig Partridge and Nicholas Shectman, "Packet Reordering is Not Pathological Network Behavior," IEEE/ACM Transactions on Networking, vol. 7, no. 6, pp. 789-798, Dec. 1999.

[78] Anjali Agarwal and J. William Atwood, “A Unified Approach to Fault-Tolerance in Communication Protocols Based on Recovery Procedures,” IEEE/ACM Transactions on Networking, vol. 4, no. 5, pp.785-795, Oct. 1996.

[79] Flavin Cristian, “Understanding Fault-Tolerant Distributed Systems,” Communications of ACM, vol. 34, no.2, pp. 56-78, Feb. 1991.

[80] Seungjae Han and Kang G. Shin, “Experimental Evaluation of Failure-Detection Schemes in Real-time Communication Networks,” IEEE Symposium on Fault-Tolerant Computing, pp. 122-131, 1997.

[81] Seungjae Han and Kang G. Shin, “Fast Restoration of Real-time Communication Service from Component Failures in Multi-hop Networks,” ACM SIGCOMM, pp. 77-88, 1997.

[82] Qin Zheng and Kang G. Shin, “Fault-Tolerant Real-Time Communication in Distributed Computing System,” IEEE, Transaction on parallel and distributed systems, vol. 9, no. 5, pp. 470-480, May 1998.

[83] Constantinos Dovrolis and Parameswaran Ramanathan, “Resource Aggregation for Fault Tolerance in Integrated Services Networks,” ACM SIGCOMM Computer Communication Review, vol. 28, no. 2, pp. 39-53, Apr.1998.

[84] Anindo Banerjea, “Simulation Study of the Capacity Effects of Dispersity Routing for Fault Tolerant Realtime Channels,” ACM SIGCOMM '96, vol. 26, no. 4, pp. 194-205, Oct. 1996.

[85] Daniel P. Bovet and Marco Cesati, “Understanding the Linux Kernel,” O’Reilly, January 2001.

[86] Alessandro Rubini and Jonathan Corbet, “Linux Device Drivers Snd Edition”, O’Reilly, June 2001.

[87] Vijay Karamcheti, Andrew A. Chien, “Software Overhead in Message Layers: Where Does the Time Go?” Proc. of the Sixth Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VI), pp. 51-60, San Jose, California, Oct. 1994.

Appendices

Appendix A Fault Tolerance Cost Estimation

Below is the reliability and fault tolerance cost estimation functions and some numerical result:

```
(*function to build fault tolerance -- Mathematica*)
(*Probability of working ports*)
Clear[g, W, n, No, p];
 No = 17;
 p = 0.1;
 g = Table[0, {i, No}, {j, No}];

 For[n = 1, n < No, n++;
       For [W = 1, W ≤ n, W++,
           g[[n, W]] = (1 - p)^W * p^(n-W);
           ]
 ]
 MatrixForm[g]
 ListPlot[g[[n - 1]], PlotJoined → True]
 ListPlot3D[g, BoxRatios → {1, 1, 0.5}, AxesLabel → {"Working NICs", "NIC No", "Prob."}]
 Show[%, ViewPoint → {1, -1.5, 1}, PlotRange → {0, .5}]
 Show[%, ViewPoint → {-1.5, 0, 0}, PlotRange → {0, 0.01}]
 Show[%, ViewPoint → {0, -2, 0}, PlotRange → {0, 0.01}]
 Show[%, ViewPoint → {0, 0, 2}]
 ListDensityPlot[g]
```

0	0	0	0	0	0	0	0
0.09	0.81	0	0	0	0	0	0
0.009	0.081	0.729	0	0	0	0	0
0.0009	0.0081	0.0729	0.6561	0	0	0	0
0.00009	0.00081	0.00729	0.06561	0.59049	0	0	0
$9.\times10^{-6}$	0.000081	0.000729	0.006561	0.059049	0.531441	0	0
$9.\times10^{-7}$	8.1×10^{-6}	0.0000729	0.0006561	0.0059049	0.0531441	0.478297	0
$9.\times10^{-8}$	8.1×10^{-7}	7.29×10^{-6}	0.00006561	0.00059049	0.00531441	0.0478297	0.430467
$9.\times10^{-9}$	8.1×10^{-8}	7.29×10^{-7}	6.561×10^{-6}	0.000059049	0.000531441	0.00478297	0.0430467
$9.\times10^{-10}$	8.1×10^{-9}	7.29×10^{-8}	6.561×10^{-7}	5.9049×10^{-6}	0.0000531441	0.000478297	0.00430467
$9.\times10^{-11}$	8.1×10^{-10}	7.29×10^{-9}	6.561×10^{-8}	5.9049×10^{-7}	5.31441×10^{-6}	0.0000478297	0.000430467
$9.\times10^{-12}$	8.1×10^{-11}	7.29×10^{-10}	6.561×10^{-9}	5.9049×10^{-8}	5.31441×10^{-7}	4.78297×10^{-6}	0.0000430467
$9.\times10^{-13}$	8.1×10^{-12}	7.29×10^{-11}	6.561×10^{-10}	5.9049×10^{-9}	5.31441×10^{-8}	4.78297×10^{-7}	4.30467×10^{-6}
$9.\times10^{-14}$	8.1×10^{-13}	7.29×10^{-12}	6.561×10^{-11}	5.9049×10^{-10}	5.31441×10^{-9}	4.78297×10^{-8}	4.30467×10^{-7}
$9.\times10^{-15}$	8.1×10^{-14}	7.29×10^{-13}	6.561×10^{-12}	5.9049×10^{-11}	5.31441×10^{-10}	4.78297×10^{-9}	4.30467×10^{-8}
$9.\times10^{-16}$	8.1×10^{-15}	7.29×10^{-14}	6.561×10^{-13}	5.9049×10^{-12}	5.31441×10^{-11}	4.78297×10^{-10}	4.30467×10^{-9}
$9.\times10^{-17}$	8.1×10^{-16}	7.29×10^{-15}	6.561×10^{-14}	5.9049×10^{-13}	5.31441×10^{-12}	4.78297×10^{-11}	4.30467×10^{-10}

From the estimation data, we found that with the increment of the channels, the reliability ratio increases rapidly.

```
(*function to build fault tolerance -- Mathematica*)
Clear[cost, W, n, No];
 No = 17;
 cost = Table[0, {i, No}, {j, No}];

 For[W = 2, W < No, W++,
       cost[[W, W]] = W - 1;
      For [n = W + 1, n < No, n++,
           cost[[n, W]] = (n - 1) + cost[[n - 1, W]] (n - W)/n;
           ]
 ]
 For [n = 1 , n < No, n++,
          cost[[n, 1]] = n (n + 1) / 2;
          ]
 MatrixForm[cost]
 ListPlot[cost[[n - 1]], PlotJoined → True]
 ListPlot3D[cost, BoxRatios → {1, 1, 0.5}, AxesLabel → {"Working NICs", "NIC No", "Cost"}]
 Show[%, ViewPoint → {-1.5, 0, 0}];
 Show[%, ViewPoint → {0, -2, 0}]
 Show[%, ViewPoint → {0, 0, 2}]
 ListDensityPlot[cost]
```

$$\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
6 & \frac{7}{3} & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
10 & \frac{25}{6} & \frac{7}{2} & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
15 & \frac{13}{2} & \frac{27}{5} & \frac{23}{5} & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
21 & \frac{28}{3} & \frac{77}{10} & \frac{98}{15} & \frac{17}{3} & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
28 & \frac{38}{3} & \frac{52}{5} & \frac{44}{5} & \frac{160}{21} & \frac{47}{7} & 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
36 & \frac{33}{2} & \frac{27}{2} & \frac{57}{5} & \frac{69}{7} & \frac{243}{28} & \frac{31}{4} & 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
45 & \frac{125}{6} & 17 & \frac{43}{3} & \frac{260}{21} & \frac{205}{28} & \frac{175}{18} & \frac{79}{9} & 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
55 & \frac{77}{3} & \frac{209}{10} & \frac{88}{5} & \frac{319}{21} & \frac{187}{14} & \frac{143}{12} & \frac{484}{45} & \frac{49}{5} & 9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
66 & 31 & \frac{126}{5} & \frac{106}{5} & \frac{128}{7} & \frac{225}{14} & \frac{43}{3} & \frac{194}{15} & \frac{648}{55} & \frac{119}{11} & 10 & 0 & 0 & 0 & 0 & 0 & 0 \\
78 & \frac{221}{6} & \frac{299}{10} & \frac{377}{15} & \frac{65}{3} & \frac{533}{28} & \frac{611}{36} & \frac{689}{45} & \frac{767}{55} & \frac{845}{66} & \frac{71}{6} & 11 & 0 & 0 & 0 & 0 & 0 \\
91 & \frac{259}{6} & 35 & \frac{147}{5} & \frac{76}{3} & \frac{89}{4} & \frac{119}{6} & \frac{161}{9} & \frac{896}{55} & \frac{329}{22} & \frac{539}{39} & \frac{167}{13} & 12 & 0 & 0 & 0 & 0 \\
105 & 50 & \frac{81}{2} & 34 & \frac{205}{7} & \frac{180}{7} & \frac{275}{12} & \frac{62}{3} & \frac{207}{11} & \frac{190}{11} & \frac{415}{26} & \frac{1350}{91} & \frac{97}{7} & 13 & 0 & 0 & 0 \\
120 & \frac{172}{3} & \frac{232}{5} & \frac{584}{15} & \frac{704}{21} & \frac{206}{7} & \frac{236}{9} & \frac{1064}{45} & \frac{1184}{55} & \frac{652}{33} & \frac{712}{39} & \frac{1544}{91} & \frac{1664}{105} & \frac{223}{15} & 14 & 0 & 0 \\
136 & \frac{391}{6} & \frac{527}{10} & \frac{221}{5} & \frac{799}{21} & \frac{935}{28} & \frac{119}{4} & \frac{1207}{45} & \frac{1343}{55} & \frac{493}{22} & \frac{1615}{78} & \frac{1751}{91} & \frac{629}{35} & \frac{2023}{120} & \frac{127}{8} & 15 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

From the estimation result we found that fault detection cost remains low when most channels are working.