PROVISIONING TASK BASED SYMMETRIC QoS IN iSCSI SAN

A Thesis by

Padmanabhan Ramaswamy

Bachelor of Engineering, Anna University, India, 2005

Submitted to the Department of Electrical and Computer Engineering
and the faculty of the Graduate School of
Wichita State University
in partial fulfillment of
the requirements for the degree of
Master of Science

December 2008

PROVISIONING TASK BASED SYMMETRIC QoS IN iSCSI SAN

The following faculty has examined the final copy of this thesis for form and content, and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science with a major in Electrical Engineering.

_____

Ravindra Pendse, Committee Chair

_____

M. Edwin Sawan, Committee Member

_____

Krishna Krishnan, Committee Member

ACKNOWLEDGEMENTS

I would like to express my heartfelt and sincere thanks to my advisor Dr. Ravi Pendse for his insightful guidance and support during my master's degree at WSU. I am grateful to my committee members Dr. M. Edwin Sawan and Dr. Krishna Krishnan for their time to review the report.

I am thankful to Mr. Amarnath Jasti for his remarkable patience all through my research and valuable suggestions that helped to complete my thesis work. I would like to extent my thanks to my supervisors and colleagues in Technical Assistance Centre (TAC), in enhancing my skills to analyze and resolve computer networking problems.

I would also like to acknowledge stupendous efforts of open-source community for removing barriers to knowledge by making software that is at free of cost. I would like to mention special thanks to Ross S.W. Walker of iSCSI Enterprise Target project and Jan Engelhardt of Netfilter project for sharing their expertise.

I am earnestly grateful to my brother and his family for their encouragement and support. I am also thankful to all of my friends, who helped me and made my stay at WSU memorable.

Most important of all, I am forever indebted to my parents for making me the person who I am, and this work would have never been accomplished without their support.

ABSTRACT

This thesis examines Internet Small Computer System Interface (iSCSI) storage architecture and introduces a framework to furnish Quality of Service (QoS) for the qualified storage traffic. The framework incorporates IO scheduler that does priority scheduling for desired IO request and receives higher degree of service in its journey through the network by adopting Differentiated Services (DS) network architecture with each node exhibiting Expedited Forwarding (EF) behavior. In this thesis, to achieve end-to-end QoS for the application, a novel functional module is introduced in storage server to provision QoS for the requested data and the proposed framework is examined subsequently with an analytical model to estimate the maximum delay bound. The results derived in the analytical model are compared with the available experimental results in the literature. The distinct implication of the proposed framework is in achieving its objectives of guaranteed transmission of data with deterministic maximum delay, without modifying existing structural components and network protocol semantics.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

LIST OF FIGURES

LIST OF ACRONYMS

AF	Assured Forwarding
AHS	Additional Header Segment
API	Application Programming Interface
AQM	Active Queue Management
BFQ	Budget Fair Queuing
BHS	Basic Header Segment
CBQ	Class Based Queuing
CDB	Command Descriptor Block
CFQ	Complete Fair Queuing
CPU	Central Processing Unit
DAS	Direct Attached Storage
DS	Differentiated Services
DSCP	Differentiated Services Code point
EF	Expedited Forwarding
EUI	Enterprise Unique Identifier
ExpCmdSN	Expected Command Sequence Number
FIFO	First-in-First-Out
FQDN	Fully Qualified Domain Name
GPS	Generalized Processor Sharing
HBA	Host Bus Adapter
HSM	Hierarchical Storage Management
IDC	International Data Corporation

LIST OF ACRONYMS (Continued)

IETF	Internet Engineering Task Force
ILM	Information Lifecycle Management
IntServ	Integrated Services
IO	Input and Output
IOPS	Input-Output Per Second
IP	Internet Protocol
IQN	iSCSI qualified name
iSCSI	Internet Small Computer System Interface
ISID	Initiator Session Identification
ITT	Initiator Task Tag
LBA	Logical Block Address
LBAP	Linear Bounded Arrival Processes
MaxCmdSN	Maximum Command Sequence Number
MiB	Mebibyte
MTU	Maximum Transmission Unit
NAT	Network Address Translation
NIC	Network Interface Controller
PDU	Protocol Data Unit
PHB	Per-Hop Behavior
QoS	Quality of Service
RAID	Redundant Array of Inexpensive Disks
RED	Random Early Detection

LIST OF ACRONYMS (Continued)

RSpec	Resource Specifications
SAN	Storage Area Network
SATA	Serial Advanced Technology Attachment
SCSI	Small Computer System Interface
SLA	Service Level Agreement
StatSN	Status Sequence Number
TBF	Token Bucket Filter
TCB	Traffic Conditioner Block
TCP	Transmission Control Protocol
TPGT	Target Portal Group Tag
TSIH	Target Session Identification Handle
TSpec	Transmission Specifications
WFQ	Weighted Fair Queuing
ZBR	Zone Bit Recording

LIST OF SYMBOLS

λ	Lambda
∑	Summation
μ	Micro
ρ	Rho

CHAPTER 1

INTRODUCTION

The need to access and share data from different locations led to the advent of ARPANET to current complex network infrastructure known as the Internet. With this as platform, numerous applications are being developed that bring out a multitude of services to end users that are accompanied by exponential growth in digital data. As per the IDC forecast [3], this growth is going to sustain for the next decade. The storage industry has responded with increased data intensity in storage media to accommodate data and advances in data access mechanisms to improve performance and enhance interfacing with computing servers. Besides developing new storage architecture models like Storage Area Network (SAN) and technical standards, the need for managing huge volume of data has evolved new trends in management practices like Information Lifecycle Management (ILM), Hierarchical Storage Management (HSM), Tiered Storage and Storage Resource Management (SRM) that provide guidelines for organizing, storing and retrieving data and building storage system environments.

1.1 Storage Area Network

The above mentioned storage management policies have the task of managing distributed data across several servers and efficient usage of storage resources. These storage resources introduce a multitude of complex implementational issues such as scalability, administrative cost, capacity planning, data protection, backup and recovery operations. Many technical standards had evolved with new storage access techniques to provide independence of data storage devices from the computing platforms. The development of storage protocols like iSCSI, FCP helped to build new architecture for interconnecting computing systems and storage devices; and this new class of network is known as Storage Area Network (SAN). It is

characterized by design philosophy of reliable, high bandwidth and low latency network interconnects.

These stringent technical requirements were met by Fibre Channel Protocol (FCP) and were quickly adopted as de-facto standard for storage area networks [2]. Due to drawbacks of high infrastructure cost and need of a specialized operational skill set, research initiatives was undertaken to take advantage of omnipresent low cost IP networks and provide greater flexibility in infrastructure management by leveraging existing skill sets. This led to the emergence of Internet Protocol based SAN architectures. The principle design is to encapsulate storage block access protocols like SCSI, ATA and transport over IP network and Ethernet physical links, which led to the development of IP storage protocol mechanism, iSCSI that removed the barrier of limiting the SAN to data center networks and enabled remote storage operations.

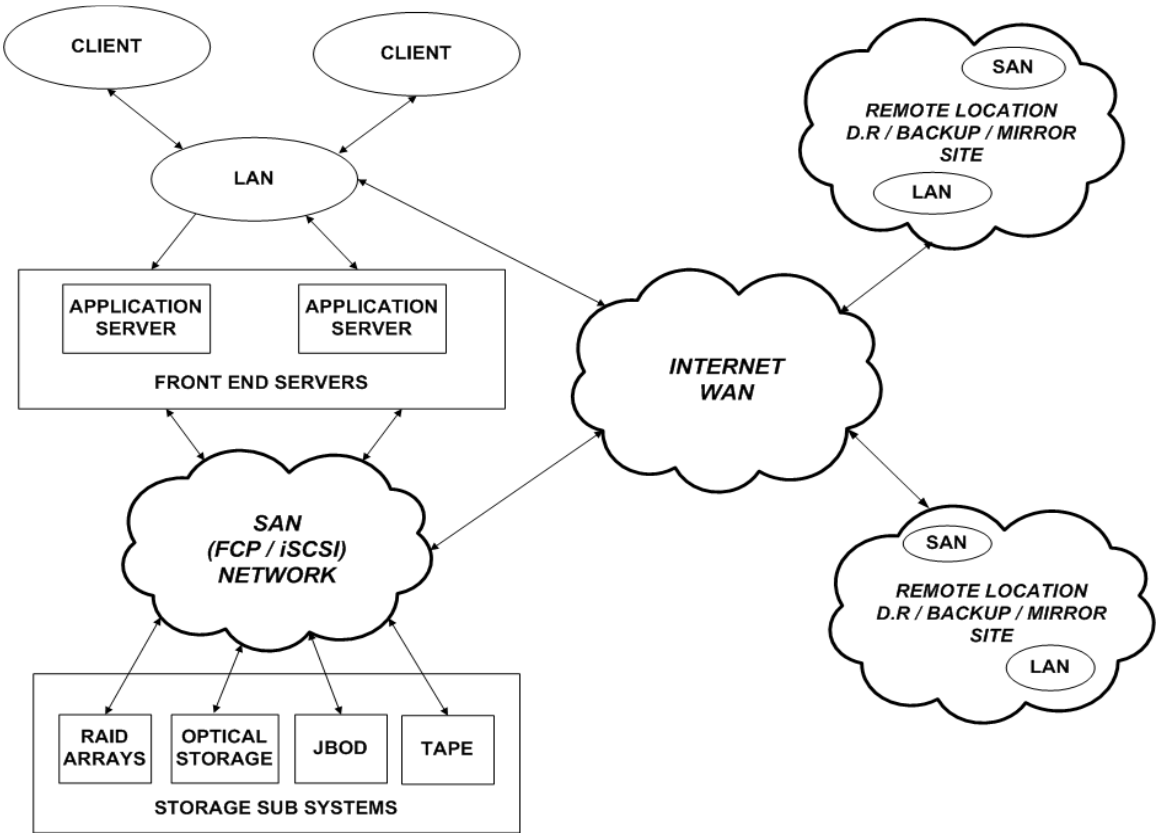

Figure 1 Conceptual architecture of Storage Area Network

Apart from reduced total cost of ownership, IP SAN inherits features of IP networks like quality of service, security mechanisms which can be applied to storage data traffic. As computing world tends towards the ubiquitous communication, the task at hand is to store information growing at the explosive rate and guaranteeing access to required information at any requested time. The challenge to accomplish this task has been in developing robust technical standards and management policy framework.

1.2 Problem Description

The deployment of wide area storage subsystems for business applications requires detailed performance modeling and analysis of design architecture involving evaluation of resources that could be a system bottle neck [2] [6]. In this thesis, the author considers an environment where servers are connected to storage devices through iSCSI based IP SAN network cloud and applications demand certain level of performance guarantee and resources from the system components. The application requirements are not satisfied by the Best Effort model adopted by packet switched networks and the iSCSI protocol is not equipped with QoS mechanisms. Through this research work, the author proposes a QoS design policy framework to guarantee packet delivery and incorporated mechanism provides a deterministic delay bound for the qualified storage traffic. It also introduces QoS provisioner functionality within iSCSI storage server and evaluates the design with analytical model to determine end-to-end delay bounds for the interesting application traffic flow.

1.3 Significance of this Thesis

This thesis has attempted in furnishing QoS in iSCSI SAN environment that extends through internet infrastructure. Only few research initiatives are undertaken in this dimension and, as per author's literature survey, previous work are available for discrete components that

constitute the proposed framework. It makes a novel contribution of QoS provisioning algorithm within iSCSI functionality sans making any modification to protocol semantics.

1.4 Thesis Organization

The remainder of the thesis is organized as follows.

Chapter 2 presents a brief overview on constituents of two domains namely- storage and network that influences processing of an IO request as it traverses between the application and storage device.

Chapter 3 reviews contemporary research work in the field of IO scheduler algorithms, Network QoS and iSCSI storage protocol.

Chapter 4 proposes QoS provisioning framework within iSCSI SAN architecture and subsequently an analytical model that bounds maximum end-to-end delay is presented.

Chapter 5 draws conclusion and offers suggestions for future work.

CHAPTER 2

TRAVERSING IO PATH IN iSCSI SAN

2.1 Introduction

2.1.1 Linux Kernel

In any operating system, the kernel is a software piece of code consisting of fundamental core functions to provide access and manage resources between different competing user and system processes. Even though the Linux kernel is primarily based on monolithic architecture with several logical subsystems integrated as a single program, it also provisions for modularity where compiled programs can be dynamically loaded and unloaded from the kernel at runtime. The kernel manages efficient utilization of resources such as CPU and memory among various subsystems.

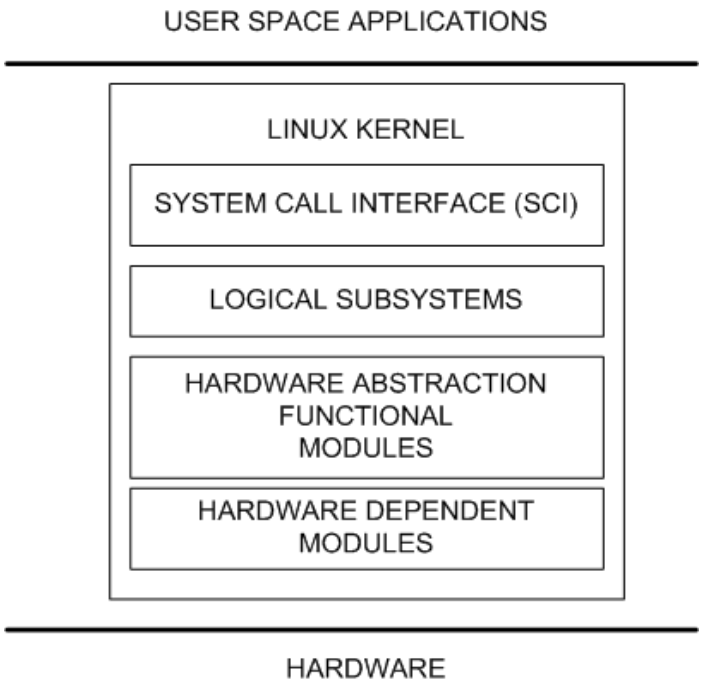

Figure 2 Logical view of Linux kernel

The subsystems are interdependent to each other, striving for same set of resources and they seldom operate in isolation. The applications running in user mode can utilize the hardware resources through direct kernel system calls or through interaction with a set of standard library functions that interface with the kernel.

2.1.2 System Calls Interface

It is an architecture dependent interface which enables user space applications to execute CPU instructions in kernel mode to access the hardware devices. Each time a system call is executed, the CPU has to switch context with current task [40]. The application function calls the wrapper routine in the library code which in turn invokes a specific system call identified by a numbered parameter. The system handler initiates appropriate assembly language instruction that will be executed by the CPU in kernel mode. System calls, thus provide security mechanism in kernel for end user applications to use hardware resources. It also enables portability of the programs across different kernels and hides the low-level complexity of interacting with devices.

2.2 Storage subsystem

This logical component of the kernel is responsible for handling all activities related to storing and retrieving data from the storage device. It delivers a generic set of functions to the application then onto low-level hardware dependent modules to manage the devices. It is comprised of functional blocks such as filesystem, volume management, IO scheduler and storage device interface modules like ATA, SCSI and hardware device drivers.

2.2.1 Filesystem Overview

The filesystem is a management framework that offers certain services for the data being written to storage media. The basic functions are logical abstractions of storage media; manage available storage space and remembering the physical location of the data. It also provides

protection to data access between different owners and various sets of standard interfaces for applications to store and retrieve the data. This group of descriptors associated with the user data is known as metadata. To support speedy data recovery most of the modern filesystems include journaling mechanisms that log metadata information before committing it to filesystem space. The combination of different services and framework policies has lead to evolution of different filesystems such as File Allocation Table (FAT), MS-DOS, New Technology Filesystem (NTFS), Network Filesystem (NFS), Journaling Filesystem (JFS), Extended filesystem (EXT2, EXT3), Global Filesystem (GFS) and Virtual Filesystem (VFS) etc. which fulfils different management requirements. Volume Management delivers logical abstraction of groups of disk devices as a single device upon which filesystem can operate on. It also provides features like mirroring, striping and concatenation of data. It can exist as a separate entity of the storage subsystem or merged with the filesystem block as lower layer functions. There are also several virtual filesystems like sysfs, procfs, tempfs etc. that enables the kernel to manage data or interact with a set of devices in an optimized manner.

The VFS is native to Linux, which is implemented as a software abstraction layer to support different real filesystems [40]. It exists as an object in kernel and greatly enhances the flexibility for the application to work with diverse underlying filesystems. The actual filesystem either manages the data residing on local disk or on a disk connected remotely through network or exists as special data structure to serve kernel management functions (not intended to manage disk space). The application request to read a file is tuned into a system call, which is then handled by VFS to take the appropriate filesystem function call to access the data. From the perspective of Linux, every object is treated as a file and is identified by inode number rather than filename. The inode uniquely points to the physical location of data on the storage device.

2.2.2 Storage Interface - SCSI Architecture Model

The storage medium has to adhere to set of specifications that are established to provide interoperability between different hardware vendors to interface with computer systems.

The widely adopted standards with variations are the Advanced Technology Attachment (ATA) and Small Computer Small Interface (SCSI). They constitute procedures right from electrical specifications to command sets that storage device has to comply while reading and writing data from the medium.

SCSI is layered architecture which has proven reliability and delivered high performance in the storage domain [47]. Its procedure includes support for other peripheral devices such CD-ROMs, scanner, printers, etc. The service interface between the layers and the objects are well defined, which makes it capable of using different transport protocols and physical interconnects to be used. The relationship between SCSI devices forms a client-server architecture model. The initiator makes a request which might be host computer's driver and peripheral device controller called as target functions as the server processing the request. The exchanged control data units are known as Command Descriptor Block (CDB) which has field that defines the task being executed. The number of devices it can address and maximum distance of the bus length varies by the SCSI standards. Two devices on the bus are identified by unique triplet I-T-L nexus; which stands for Initiator, Target and Logical Unit Number (LUN). LUN is defined as a numerical parameter to identify a SCSI entity on the storage bus. The number of logical units that can be attached to the SCSI bus is limited by the bus standards.

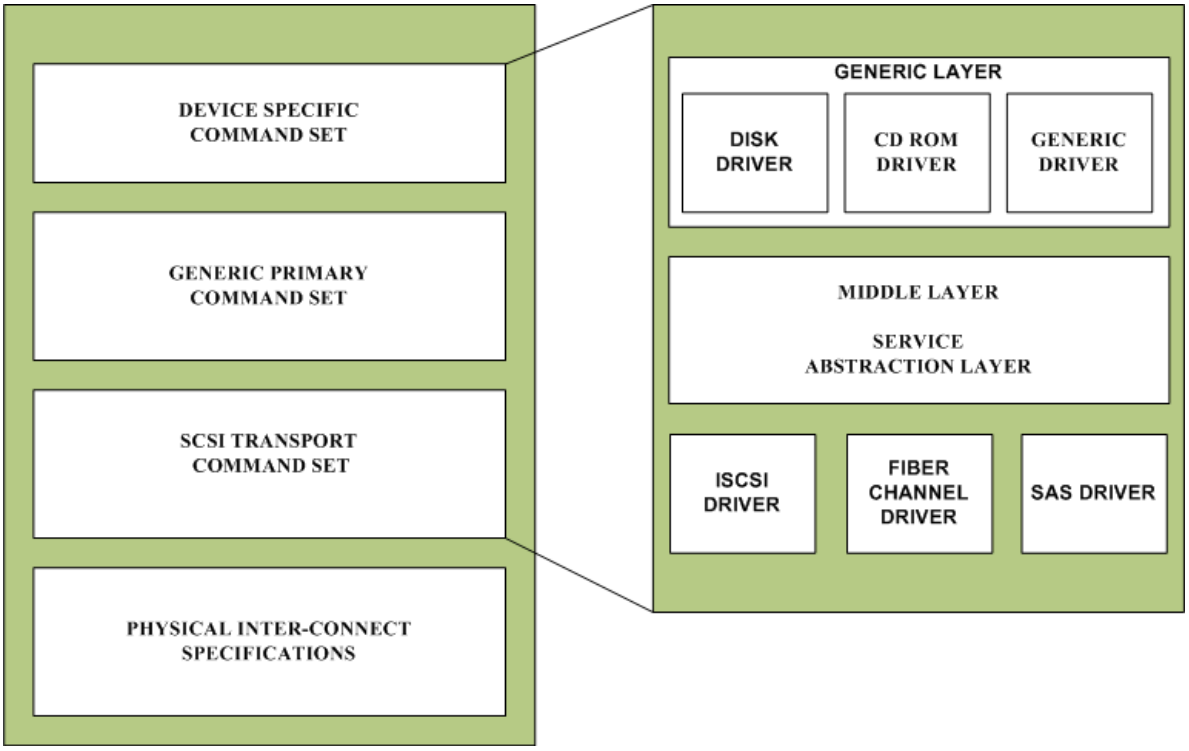

Figure 3 Logical view of Linux kernel

The top layer delivers the common set of functions for the class of devices such as tape, multimedia devices, disk drives etc. It describes the common behavior for a class of devices and collection of procedures to communicate with a device. As in Linux kernel implementation, it is organized based on the device types such as disk drive, DVD, storage tape units etc. The generic command layer prescribes commands common to all SCSI devices and forms a layer of abstraction between the actual physical low end drivers that interfaces with target. This enables to initiator to communicate between different vendor implementations of same type of physical interconnects or SCSI transport medium protocols.

2.2.3 IO Scheduler

As stated by Amdhal's law that the overall system speed is determined by the slowest component, the fetching, or writing data to and from the disk subsystem consumes more CPU time than any other kernel activity. Therefore, the tasks of minimizing disk seek latency and to

optimization of the disk IO performance is assigned to the IO scheduler as the primary job function [45]. It reduces seek time by sorting requests generated by the above layers based on spatial location of disk access and improves performance through merging multiple request to a single operation based on their disk activity. The initial disk scheduling algorithms like Shortest Seek Time First (SSTF), SCAN, LOOK, C-SCAN and C-LOOK were solely aimed to reduce head movement, thereby minimizing seek times. As research advanced in this field, the IO scheduler was added with functions to allocate disk bandwidth resources among different processes in fair and efficient manner.

It has lead to the development of four schedulers that are available in the Linux kernel that are optimized to different types of workload. They are Noop (no-operation), Anticipatory scheduler, Deadline and Complete Fair Queuing (CFQ) Scheduler.

The scheduler maintains different input queues for the incoming request, which are sorted and merged with pending requests which are implemented using red-black binary tree and FIFO lists, which are maintained for time-based searching operations [46]. Noop scheduler is suited for truly random workload and random-access devices where only merging action is performed. Deadline scheduler provides strict guarantees for dispatching a request where it maintains a timer for requests in the FIFO queue. It is well suited for disks with heavy workloads that avoid write-starving-read problems. Anticipatory scheduler anticipates that the next IO request will be near to the current seek location and stay over the current position for a configured time limit. The anticipation is made on the characteristics of dependent disk activity for a particular operation and is based on IO statistics that are monitored for individual processes.

CFQ supports time-sliced operations on IO request and maintains separate queues for each process. It has features to classify request into three scheduling classes and assigns eight levels of

priority. Request from queues are served in a round-robin fashion, achieving fairness among multiple processes' IO request along with overall throughput is still an active research area. As of this writing, the Budget Fair Queuing (BFQ) scheduler is an advancement of CFQ which allocates disk bandwidth based on the budgets (number of sectors) and guarantees high disk throughput of IO request from several processes [17].

2.2.4 iSCSI Concepts

The omnipresence of IP networks has compelled research initiatives to develop mechanisms to transport SCSI CDBs over TCP/IP layers. This has led to the emergence of Internet over SCSI (iSCSI) protocol which has given a new interface to attach storage over the network.

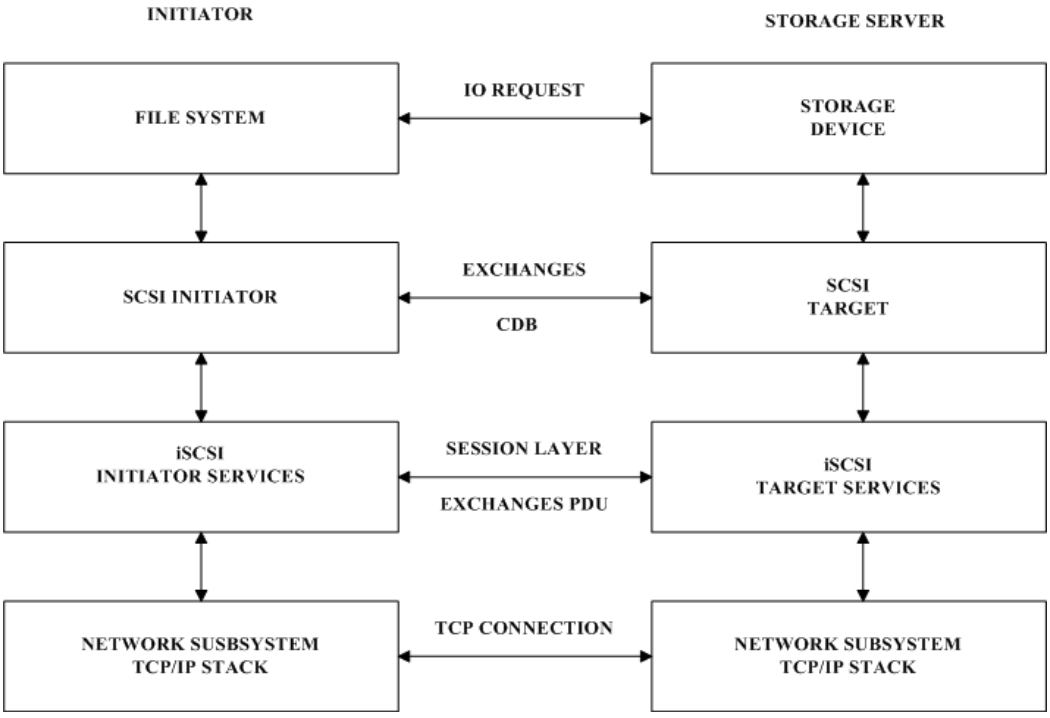

Figure 4 iSCSI layer interfaces storage subsystem with TCP/IP stack

The iSCSI layer interfaces between SCSI subsystem and TCP/IP layer, and encapsulates CDBs into iSCSI Protocol Data Units (PDU) which are handed over to Transmission control Protocol (TCP) for delivery to the storage device connected across the network.

The PDU consists of Basic Header Segment (BHS) and optional Additional Header Segment (AHS) that characterizes the request type exchanged between iSCSI nodes. The functional roles are much similar to SCSI where initiator generates request and target processes the request. However, the addressing convention requires that the name of the initiator and target nodes to be unique in the world [11]. There are two types of naming policies used with iSCSI. The iSCSI Qualified Names (iqn) uses a combination of characters “iqn” along with reversed Fully Qualified Domain Name (FQDN) and date of assignment. The second method Enterprise Unique Identifier (EUI) is based on IEEE registered 64 bit length represented in hexadecimal digits. Within an iSCSI node, many iSCSI portals can exist that interface with network ports to establish connection with another iSCSI portal group. The initiating portal groups are identified by an initiator node name and Initiator Session Identifier (ISID) which is created by vendor ID, enterprise number of vendor and a qualifier. The target portal uses iSCSI target node plus Target Portal Group Tag number as identification and unique session established with Initiator Portal Group. The network portal group consists of an IP address and port number which can be associated with only a particular iSCSI node group.

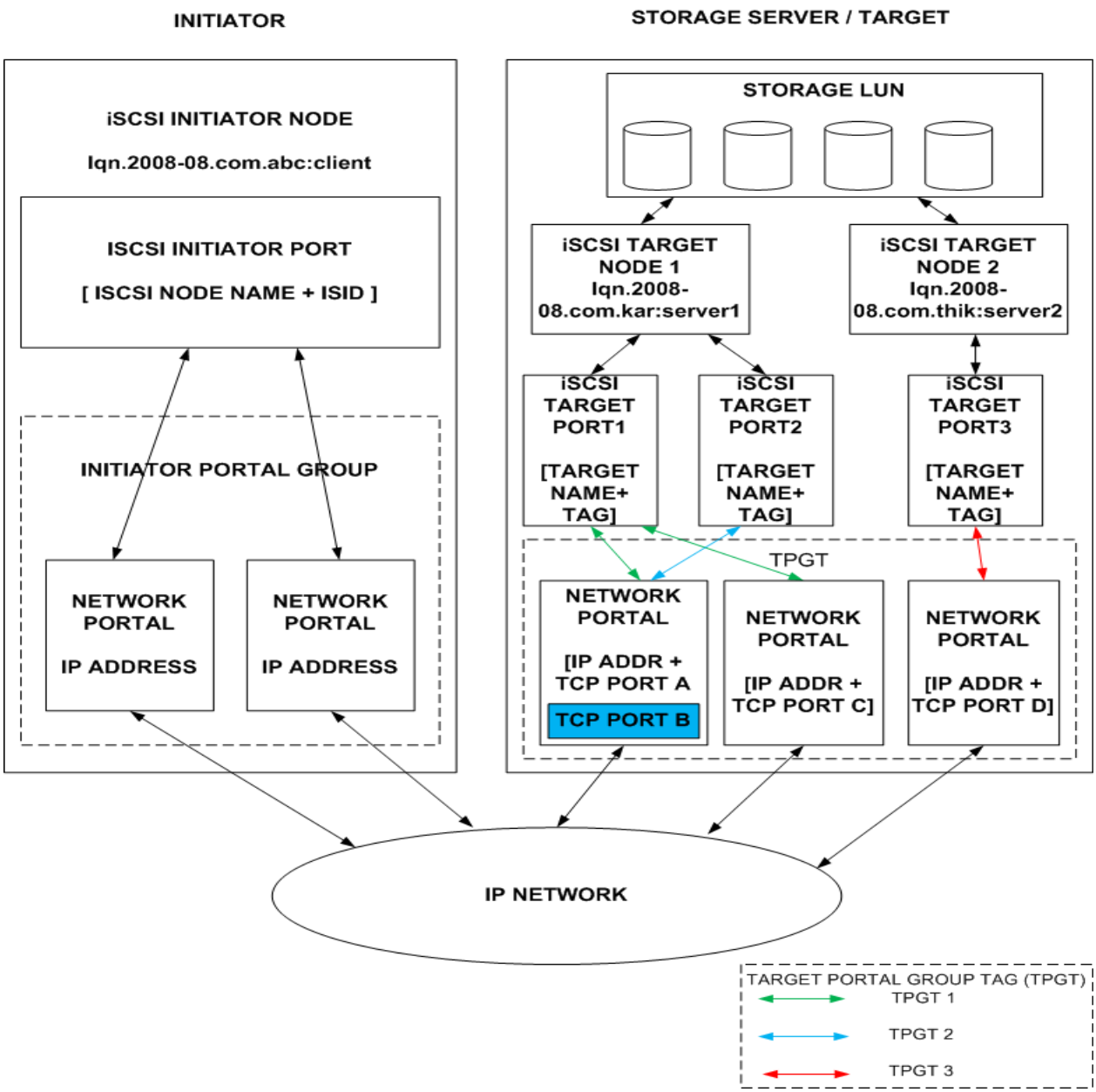

Figure 5 iSCSI portal relationships

A TCP connection has to be established between the network entities before starting an iSCSI session and there can be of maximum of four connections per session. The session has to go through discovery, login, full feature phase and an optional security phase before exchanging data. Each phase accomplishes a set of functions such as identifying iSCSI nodes, negotiating

parameters and announcing capabilities of each node to set logical channel of communication between SCSI endpoints.

Once the session enters Full Feature phase, data can be exchanged between initiator and target nodes. The target delivers the commands in the same sequence as issued by the initiator node and the data and status message sent back on the same connection in which commands are issued. This is known as a connection allegiance, which is very useful for error recovery mechanisms. An active task within a session is identified by the field set by initiator in iSCSI PDU known as Initiator Task Tag (ITT) [11]. This field not only helps in identification but also useful in direct memory placement of data and buffer management. iSCSI implements its own flow control mechanism by limiting the number of outstanding commands per session. This command window is determined by a difference in the values of iSCSI header field Expected Command Sequence Number (ExCmdSN) and Maximum Command Sequence Number (MaxCmdSN) plus one.

2.3 Network Subsystem

The Linux kernel is coded with rich set of functions which supports basic network services required for a host to advanced features like routing, packet inspection and traffic management capabilities that qualify it for deployment as a high-end router.

2.3.1 Sockets Interface Layer

A socket is identified as the logical end point for the communication channel between two systems over the IP network. It also serves as an interface between applications which want to use networking facilities offered by the kernel. These sockets are also known as Application Programming Interfaces (API) and provide independence for the applications or processes implementing each protocol specific functions to communicate over the network. The APIs are

used by a process in creation of a socket to send and receive data, and the closure of the socket to shut down the service. Before using sockets, the process has to define the characteristics of the communication and a protocol family with an address space that is valid for specific protocol. This layer also acts as a multiplexor where user level sockets call arguments and are mapped to corresponding kernel system calls. Apart from providing network services to the applications, they also include mechanisms to carry out read and write operations to transfer data over networks similar to IO functionality. A process which wants to read or write data over the network can either directly use socket functions or invoke corresponding kernel system calls that will initiate socket calls to perform protocol specific operations. This is achieved by referencing socket with a file descriptor [39].

2.3.2 TCP/IP Stack Overview

The TCP/IP stack forms the core of network operations that enable resources to be shared between two programs running on disparate systems. The layers in the TCP/IP model have peer-to-peer relationship and as application data traverses down the stack, they are encapsulated by each layer's header data that hides application data from the below layers. The transport layer offers two distinct styles of communication. The predominant transport protocols are TCP and User Datagram Protocol (UDP) that provide connection oriented and connectionless service respectively to the application. The Internetwork Protocol (IP) layer is a routed protocol that controls how data should be forwarded out of the network node. IP layer can be further viewed as routing, security, queue management and Internet Message control Protocol (ICMP) logical subsystems, but their functions are strongly integrated with each other. The routing function includes dynamic routing protocols which determine the path to be taken by the packet in an established network.

The layering of the stack is implemented through socket data structures. The three important socket structures are used for storing control information for the opened sockets, and in the buffer to hold data and generic structures used by protocol to maintain their control information [39].

2.3.3 IP Packet Transmission and Reception

When the application has data to send, it writes them to the allocated buffer space and invokes the appropriate socket call. This marks the beginning of the transmission journey within the kernel network stack where first data is copied from user space into kernel space and the corresponding kernel system call is initiated. The socket function implements requested protocol specific mechanism to fill out the socket data structure along with the pointer that locates the data in the memory. If it is TCP, it fills out its header portion of the socket data structure, and makes a decision on transmission and passes the structure pointer to IP transmit function. It looks for the route information, and, based on the forwarding decision, the packet is placed in the output queue of the network interface. All functions related to queue management and traffic control takes place at the queuing layer and scheduler's output function interrupts device drivers transmit routine. The device driver should verify whether any of the options like segmentation offload, IP checksum or scatter-gather flags are enabled [41] and transmit the packet onto wire after fulfilling those operations.

On reception of a packet, the network device checks its list of available socket descriptors and does DMA operation into the kernel memory to place the data and raises an interrupt with device driver. The Interrupt Service Handler (ISR) of the driver determines the type of interrupt and sets the upper layer protocol field in the socket buffer structure. ISR also needs to update state information of the device and descriptors of buffer list in preparation of next DMA transfer.

Since interrupt suspends other activities, New API (NAPI) is introduced in device driver functions for efficient handling of packet reception and interrupts generation. This enhances kernel on systems with heavy network traffic to poll driver and drop a packet even before the packet is processed by network stack functions and saves CPU cycles [41]. The receiving process of the IP layer, a kernel thread known as softIRQ, is invoked by ISR. This thread handles de-queuing of the packet from the input queue and appropriate packet handler operations are performed. This includes basic sanity check of validating the checksum, version, length, etc. to making a routing and forwarding decision on where and which layer to send the packet.

In the case of transport protocol being TCP, the TCP receive handler function is called and it starts processing the protocol information. It runs through the field for protocol errors and updates the Transmission Control Block (TCB) structure with control information. TCP allows two ways to process the input segment known as “slow path” and “fast path” processing [39]. In the slow path processing the socket has two queues namely receive and backlog going through normal processing of data. The fast path uses third queue known as “prequeue” that is serviced by socket in application context rather TCP switching context between kernel and user mode. Once the user task receives signal about pending data in the socket, it invokes kernel system calls to read data from kernel space to user space buffer. This completes the upwards journey of data through the network stack.

2.4 Network Quality of Service

Packet switched networks are designed to deliver best-effort service to all types of traffic generated by the end hosts. However, this best-effort model is not suitable for many applications that demand a certain level of assurance from the network. Also, a network node comes with limited packet processing resources and the available bandwidth of links connecting nodes is

restricted by the physical characteristics of the medium. Therefore it is imperative to employ traffic management measures to achieve even the best-effort service. Thus, the need for Quality of Service (QoS) has evolved from behavioral characteristics from a multitude of applications and the need for more optimized utilization of limited network resources.

2.4.1 QoS Performance Metrics

The parameter to measure the observed level of service varies upon the end application. From a performance perspective, network QoS are characterized by following metrics [26].

Delay: It is the time latency taken between reception of a packet and the transmission of the last bit of the packet out of a physical interface. This includes the serialization delay- time taken to encode the frame, propagation delay- time taken for the bit to travel on a given physical medium and the queuing delay- time spent in the software queues before a packet is actually transferred to hardware interface. Out of these delays, serialization and propagation delay are determined by the physical characteristics of the hardware and queuing delay is a variable parameter based on traffic flow and node resources. Reducing queuing delay in a node is an active research area that has developed various techniques and statistical models to address it.

Packet Jitter: It is described as the time variance between consecutive packets. An end host generating traffic at constant time intervals may not receive with time spacing at other end due to queuing delay factor at the network nodes. For real-time applications like voice traffic, this component plays important role that determines the perceived quality of audio.

Packet Loss or Reliability: Due to the bursty nature of network traffic, a packet may get dropped due to overflow of the queue at the network nodes. This can severely degrade the performance of the certain traffic flows like audio and video applications.

Throughput: It is the achievable transfer from a given link that is shared among different traffic flows.

2.4.2 QoS Implementation Model

The QoS delivery model comprises of collection of policies to enforce application demands such as a guaranteed delivery along with defining performance characteristics like delay, reliability expected from the network on the transmission of IP packets. QoS policies can be implemented in practice by adopting one of the two approaches that identifies traffic either on microscopic scale, per flow basis or on a macroscopic level for aggregated class of traffic. Each of the methods is distinctive and requires careful planning considerations. Based on these methods, two implementation models are constructed. They are the Integrated Services (IntServ) and Differentiated Services (DiffServ) architecture.

2.4.2.1 IntServ Architecture

IntServ model allocates network resources on a per flow basis even before admitting the traffic into the network [30]. The model's primary components are defining service classes, admission control, signaling protocol and packet classification and scheduling. The service classes included in the architecture are best-effort, guaranteed services that offer one way end-to-end delay bound and traffic rate allowed up to defined maximum throughput value and controlled load service which strive to emulate dedicated links with high reliability. The identification process of traffic belonging to a service class is derived from the characteristics of the traffic flow. The flow specifications that inform about the behavior of the traffic are known as TSpec and the list of network resources it needs for a service class is defined by RSpec. Both of these components are agreed on the time of service level agreement.

For a network to conform to these agreements, the edge has to make a decision whether network has sufficient resources to allow a particular flow or if allowed flow can cause degradation of services to existing flows or not. This process is known as Admission Control, is implemented at the network edges. The decision control procedure is comprised of technical information about the availability of network resources at the moment and also includes management policies that specify treatment of flows based on their origination. In order to assist admission control with network state information, a separate signaling protocol, known as Resource Reservation Protocol (RSVP), operates in the network. The important characteristics of RSVP are maintenance of soft state at each network node and allocation the resources based on receiver properties. The soft state is built and refreshed periodically until the flow ceases out. This property retains the robustness feature of a packet switched networks [30]. RSVP has the feature of resource reservation based on receiver's capabilities rather than sender demanding provisions from the network.

Once a flow is identified as subset of a service class and admitted into the network, further granular policies are needed at each node to classify packets belonging to each flow and provide service to those packets with appropriate scheduling mechanisms. A general approach for the classification of packets is to examine header fields of IP and Transport protocols and map them to the established flow. To make sure that the packet is serviced within the agreed delay bounds, care must be taken on implementation of queue management algorithms on the nodes. The algorithm must allow for selecting packets from the queue based on the service class and treated at deterministic service times.

The IntServ was not widely adopted in IP networks due to scalability and complex overhead needed to achieve QoS requirements [29]. The issue of managing several micro flows and their

state information on each node raised serious concerns on scalability. Also, as IP is designed to run across different data link layers, the bandwidth reservation and achieving reliability in decentralized link layer technologies proved to be complex task. These issues gave rise to DiffServ approach that addresses scalability and reduces network node processing overhead to support QoS

2.4.2.2 DiffServ Architecture

This model was constructed to support QoS based on classes of aggregate traffic that provides preferential treatment which is scalable and can be implemented with various levels of serviceability to meet application and business requirements [29]. Thus, a set of network entities that is governed by an administrative policy which differentiates classes of traffic and conformed by all nodes is known as DiffServ domain. In broad terms, DiffServ is implemented in two stages that define a collection of functions to be accomplished by the network edges, known as ingress/egress nodes, to determine traffic into allocated behavior classes and packet handling mechanisms belonging to traffic aggregate at interior nodes of an autonomous network domain.

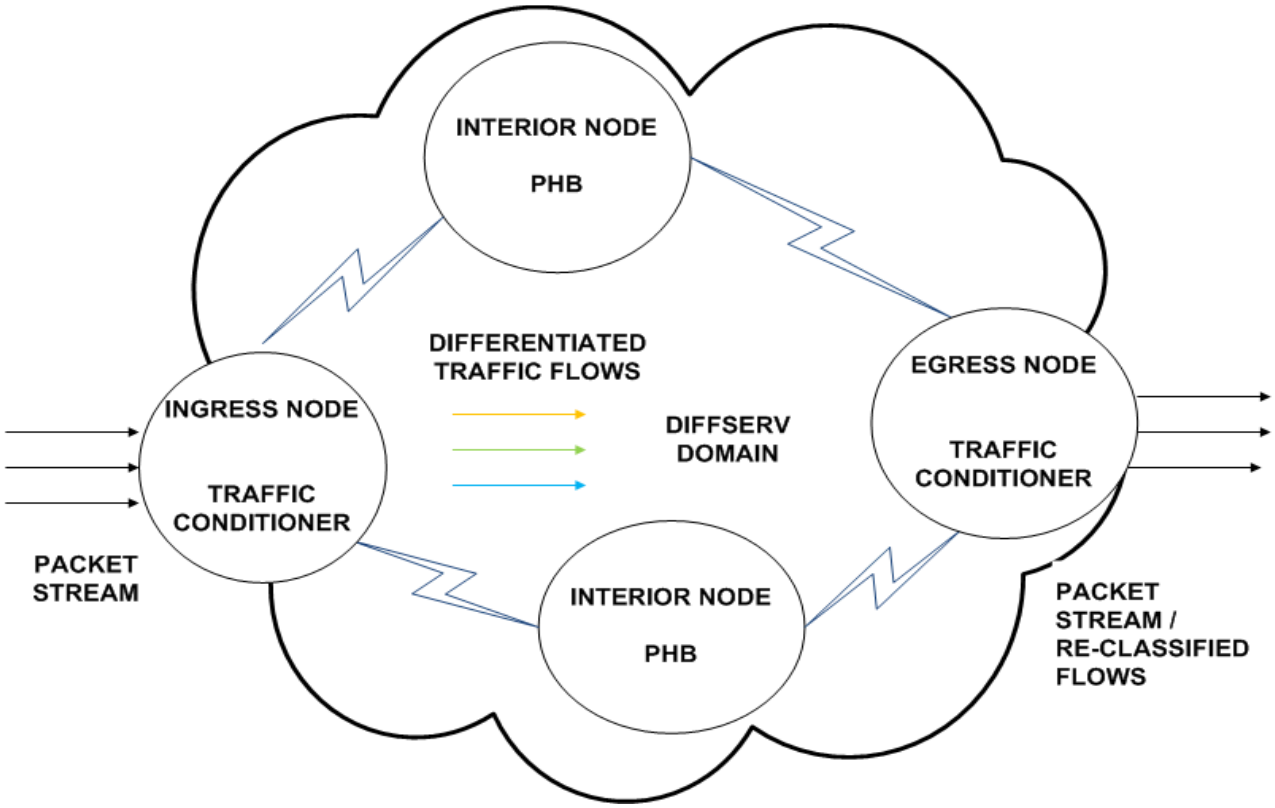

Figure 6 DiffServ network architecture domain

The primary principle operation of DiffServ model is based on the Differentiated Services (DS) field which is actually a redefinition of Type of Service (TOS) field in the IP header [29]. From the eight bits, only the leftmost six bits are defined as Differentiated Services Code Point (DSCP) and the remaining two bits are unused within this QoS architecture.

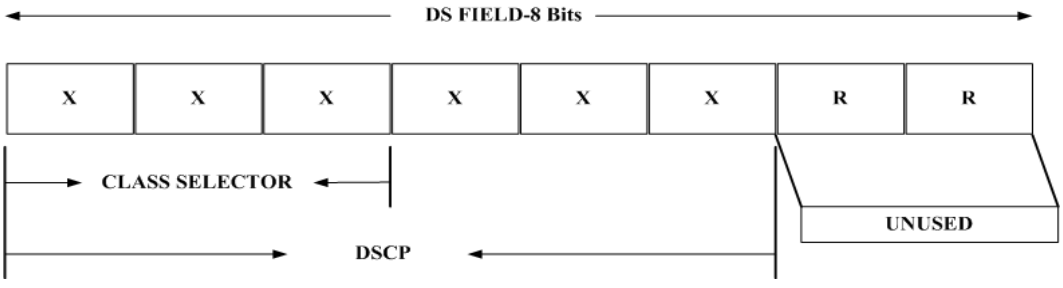

Figure 7 DS byte field

The ingress traffic is categorized into behavior aggregate with one of the values defined by the DSCP value which also dictates the preferential treatment given by the interior nodes.

2.4.2.2.1 Ingress Node – Traffic Conditioner

Before implementing a Diffserv domain, a policy has to be written on what service classes it is going to offer and how each class will be mapped to receive differentiated service along the network path in the domain. The boundary nodes, known as ingress/egress node (depending upon the direction of traffic flow into/leaving the domain), perform traffic conditioning functions that categorize traffic flows into behavior aggregate. A traffic conditioner is composed of four functional blocks- classifier, meter, marker and shaper/dropper [29].

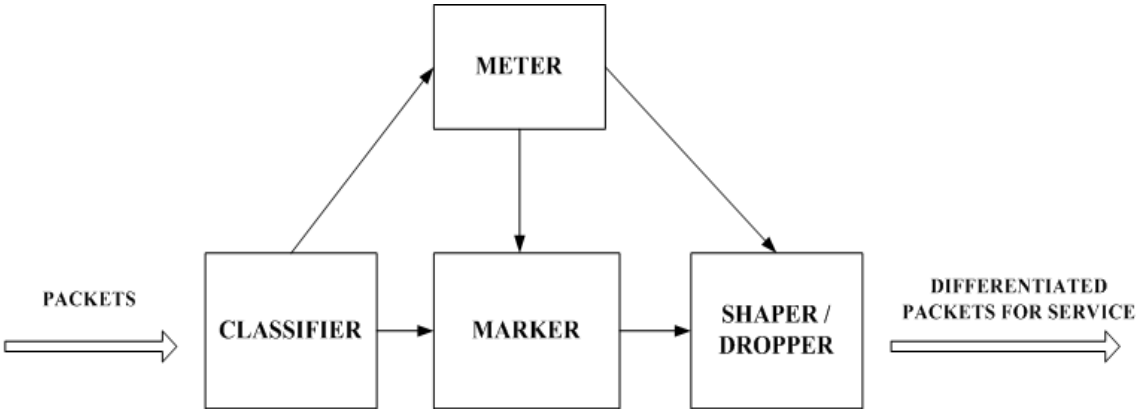

Figure 8 Traffic Conditioner Block (TCB) diagram

The classifier does the role of picking up packets from incoming traffic stream based on the DSCP code point or by looking into fields of IP and transport header for further processing. This is the first step in the domain applied policy execution of the applied policy of the domain for handing out specific treatment detailed in Traffic Conditioning Agreement (TCA) between the domain and other domains or customers. The classified traffic goes through metering functions that verifies whether the stream complies with the agreed traffic profile or not. The accounting activity is possible from the quantifiable properties of the flow and forwards the traffic with state information to the marker. The marker sets the code point based on the acquired state information or re-marks the DSCP value to another value as per the policy. If the marked packets

are in accordance to the traffic profile, it will be sent to the queuing management functions of the node. If the packets are out of profile, they may either get delayed i.e. shaped to comply with the agreement or it can be dropped. The action of dropping the packets to regulate the flow rate is known as policing the traffic.

It is not mandatory for a traffic conditioner to have all of the above functions [29]. It can be implemented with simple static policy with a packet classifier and marker. By design principles, the location of the traffic conditioner is recommended to carry out at ingress/egress nodes. However, it is not limited to these nodes and may also be placed at the interior of a DiffServ domain.

2.4.2.2.2 Interior Node – Per-Hop Behavior (PHB)

The nodes at the core of the Diffserv domain are responsible for carrying out treatment based on the DSCP code point set in the packet. In the current architecture, two behavior groups are defined that can hand out various levels of differential service. One of the important characteristics of the PHB is that they do not re-order packets within the same micro flow belonging to same class. For a DiffServ interior node, the two standardized PHB groups are Assured Forwarding Group (AF) and Expedited Forwarding (EF) group.

2.4.2.2.2.1 Assured Forwarding PHB Group

This group defines four classes of traffic that allows node to allocate unequal resources for carrying out service discrimination and within each class; three levels of drop precedence are designated for the treatment of packet under the event of congestion. The higher drop precedence, higher the probability of packet being dropped. This group delivers in total of 12 kinds of service and is denoted by AF 11(001010), AF12 (001100), AF13 (001110)…AF42 (100100), AF43 (100110). This group does not explicitly suggest using Random Early Detection

(RED) queuing discipline algorithm for congestion management, but requires an algorithm that holds several properties of RED algorithm.

2.4.2.2.2.2 Expedited Forwarding PHB Group

The motivating principle behind the proposed behavior is to confine the variable component of network latency i.e. queuing delay at a node so that experienced delay can be measured quantitatively [27]. With the implementation of this behavior we can achieve low loss, low delay and jitter for packets conforming to traffic flowing in DiffServ network. To guarantee such a service, the flow must be controlled at network boundaries so that its average rate is less than the minimal output link rate along the network node and service rate offered by each node. The forwarding mechanisms have to implement queuing disciplines that can deliver the priority treatment for packets at the interfaces of the node. The recommended code point for EF group is 101110.

2.5 Queuing Discipline

Queuing Discipline is a strategy mechanism that defines how the elements in a queue are organized and selected for service. In networks, the need for such techniques is driven by applications demanding certain performance guarantees from the network [36]. The queuing strategy must also take into the account of providing fair access and maximizing performance to multiplexed resources. Also, on mode of operation, they can be designed with work-conserving or non-work conserving principles. Although there many queuing disciplines available based on these design objectives, the following section discusses two common queuing disciplines- First-In-First-Out (FIFO) for best effort services and Weighted Fair Queuing (WFQ) and its variations for guaranteed service delivery model.

FIFO: This is the most basic queuing mechanism in which packets destined to the output interface of node from all sources of traffic are fed into a single queue and are forwarded out the same order in which they arrive. It is default implementation on all network nodes that are available for best-effort service. The advantage is in the ease of implementation and predictable behavior. The important parameter to consider is the maximum queue length which determines drop rate for a traffic burst. The shorter queue provides reduces average queuing delay whereas it increases drop probability. Longer queue length results in increased average delay with decreased drop probability for short burst of traffic. Linux supports a variation of FIFO termed as pfifo_fast which contains three FIFO queues with three different priority levels. Whenever there are packets in the higher priority queue, they are always served first and then services the next level of priority queue. It provides a simple implementation of service differentiation to packets marked with DSCP values.

Weighted Fair Queuing (WFQ): It is a variation of the Fair Queuing (FQ) technique in which incoming packets are classified into different flows, en-queued and selected for transmission based on their assigned weights. The WFQ algorithm is an approximation of Generalized Processor Sharing (GPS) and a packet is serviced from the queue based upon simulated time calculations. The important property of WFQ is that it empowers a network node to support differentiated services and when it is implemented for a traffic flow shaped at network boundary, it provides guaranteed delivery with end-to-end delay bound. WFQ allows different traffic flows to have different service rates based on their configured weights and protects other traffic flows from another misbehaved flow. Additional enhancements were made to WFQ algorithm to reduce the complexity of implementation and its variants are known as WF2Q and WF2Q+ [43].

2.6 Netfilter Architecture Framework

During the journey of a packet through the network stack, it can be examined at various places to execute functions for the supported features like firewalling, packet mangling and Network Address Translation (NAT). These collection of functions along with inspection points, known as hooks constitute Netfilter framework [38]. Each feature comprises a series of hooks (also known as chains) and is collectively termed as tables. The default tables that are defined within the framework are NAT, Mangle, Raw and Filter. The default hooks comes with framework are known as Pre-routing, Post-routing, Input, Output and Forward chains. A table can comprise a set of chains or all of the supported chains to accomplish the defined operation. Also not all hooks are applicable to a table for inspection of a packet as it is depends upon the journey (whether if the packet is destined to itself or packet needs to be forwarded, etc).

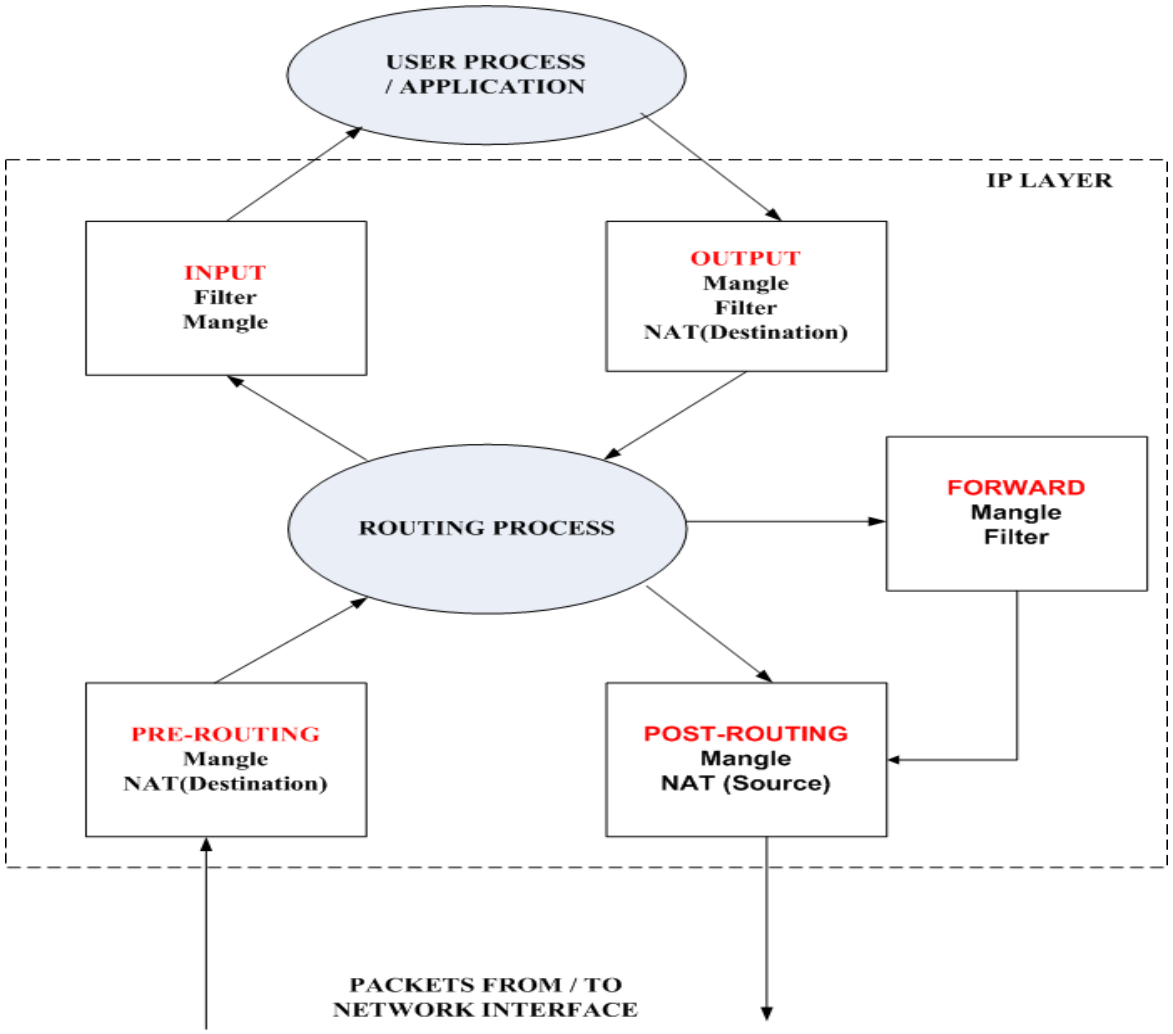

Figure 9 Netfilter architecture framework

For a packet destined to the host, the device driver interrupts the kernel for further processing and the firewall operation initiates the rules created under filter table that involves only input and output hooks. These chains examine the data sent or received from an application and, based on the match result; it will initiate the defined action on the data. To alter a packet, mangle table can have all the hooks applied to the packet based on the characteristics of the journey and configured policies [38]. Apart from the built-in tables and chains, the netfilter framework comes with rich features that support customized tables and rules which pass the packet to user space for further processing.

2.7 Putting it all together

For a process or application running on host, it does not need to be aware of the exact semantics to interact with various network protocols or storage hardware devices and their point of attachment whether it is locally connected to the host or remotely connected via network. This provides portability for an application and all it needs to take care about interfacing with an operating systems kernel for obtaining requested services.

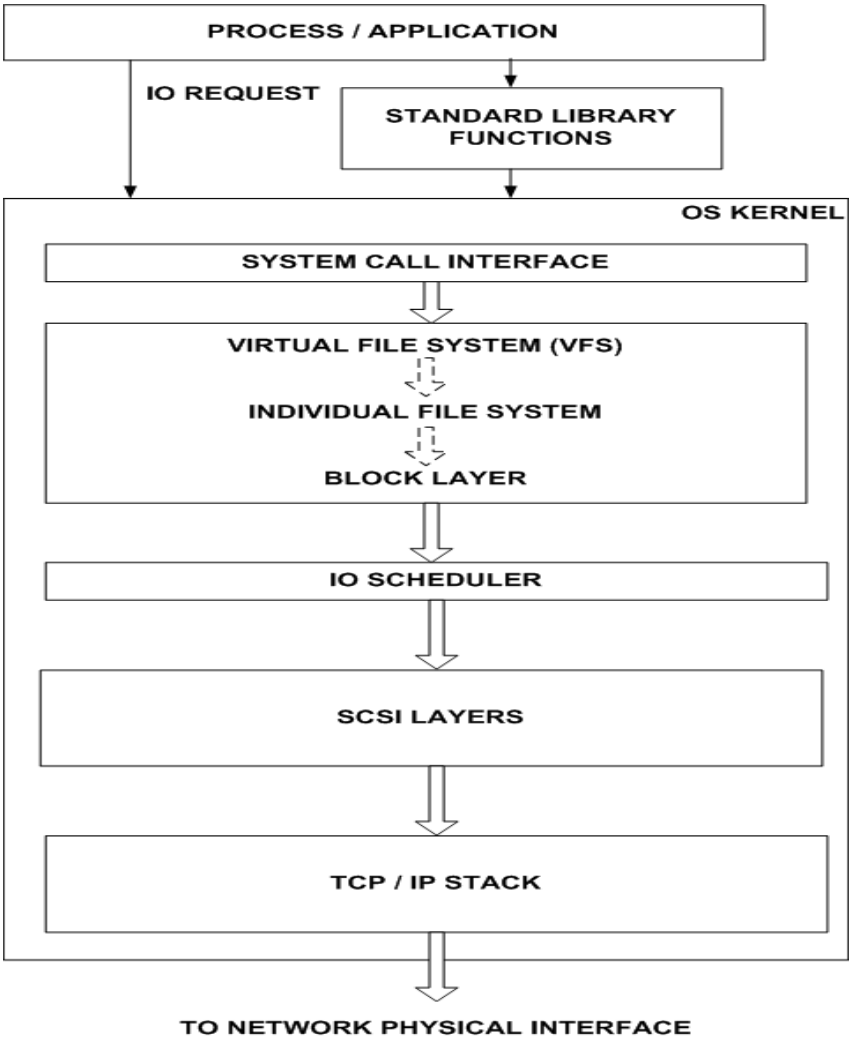

Figure 10 Path of networked storage IO within a Host

When an application generates IO request, the kernel receives and executes it in its space to furnish resources from appropriate systems to accomplish the request. The illustration gives IO

path traversal for a read/write operation to storage device connected over the network. Here, the request has to go to the device to obtain data. The illustration does not include information on cache mechanisms and its location, which is applied at each layer and plays an important role in boosting IO performance.

CHAPTER 3

LITERATURE REVIEW

This chapter provides a brief overview of relevant research efforts to comprehend storage area network design requirements, iSCSI protocol characteristics and in the field of IO scheduling algorithms, network QoS and modeling of disk characteristics.

3.1 Evaluation of SAN design requirements and iSCSI Characteristics

A detailed survey of server IO interconnects and fabrics is presented in [1], where author investigates the existence behind the multitude of interconnects and design factors that motivate them. He also reviews design principles of various interconnects and evaluates architecture implemented in available storage technologies. As IP networks grew in maturity and began to take advantage of its wide scale deployment, research activities led to advent of iSCSI storage protocol and IP based SAN architectures. In [2], P. Sarkar et al brings out the demands to be satisfied by IP storage protocols (i.e. iSCSI) in terms of storage management, performance and security to be competent with FCP standards and provides insight on enhancing iSCSI to be more flexible and outweigh fibre technology in SAN deployment.

Since inception of iSCSI, numerous researchers worked on benchmarking performance of software and hardware (HBA) iSCSI implementations and mechanisms to optimize software implementation by analyzing the protocol behavior. From the work done in [5, 7], it is revealed that naive deployment of iSCSI in storage area networks does not perform well compared to FCP and cannot exploit the abundant network resources or it is restricted by available network system components. They also discuss the overhead added by iSCSI and nature of its IO behavior with host system components over a wide range of application performance. The iSCSI protocol processing overhead in terms of computing and memory requirements are evaluated in-depth by

[6] and explicates architectural characteristics of the protocol responsible for consuming CPU, memory resources and effects of cache misses. The study reveals that processing cost associated with iSCSI are composed of two types of components- constant component that depend on number of iSCSI commands where time taken to process each command remains constant and variable component that is based upon IO size. For iSCSI to be scalable and deliver high performance in cluster environments, it must be capable of utilizing multiple CPUs efficiently. The authors of [10] approached this problem with asymmetric processing model to isolate the network processing from operating system's IO functions and present their results. The design objectives of iSCSI target functions are investigated in [9] and authors provided insight on the implementation aspects of iSCSI target. Their work pioneered the development of iSCSI Enterprise Target software.

3.2 Disk IO scheduling

The processing speed of storage devices did not grow in parallel to the CPU processing rate and a enormous performance gap exists between computing and storage systems. For any system, the overall performance limited by the slowest component and thus improving efficiency of disk based storage devices (composed of mechanical components) has been actively researched to this date. Many researchers responded with novel disk scheduling algorithms that manages IO request primarily on spatial location of disk access and also tends to divergent objectives like improving throughput and fairness among multiple processes [18]. The key step in designing an efficient algorithm involves estimating the type of IO requests to be generated by the applications. This task is not well defined due to the inexplicable nature of end user's data access pattern and disparate application behavior. The comprehensive study by authors of [4] divulges that IO traffic pattern as bursty and exhibiting self-similar characteristics. The principal

objective of any disk IO scheduling algorithm is to optimize disk seek latency by sorting the incoming IO requests. [12] Analyzes available schedulers in Linux operating systems and their design principles. Each scheduler is optimized to handle specific kind of workload and will lose its efficiency if intended to handle IO requests of varying mixtures of disk access pattern. In order to address this issue, [14] introduces the self learning algorithm to adaptively change the scheduling mechanisms to meet the work pattern. The other significant design objectives of schedulers is to provide performance guarantees for qualified requests and at the same instance, achieve fairness among all requests being served by the system. Several researchers take different approaches in the trade-off between the above mentioned design objectives and developed novel algorithms [13, 15, 17], in an attempt to meet these requirements. In particular, recent work done by Paolo et al [17] proves the effectiveness of their algorithm over existing schedulers with a wide range of real time experiments.

3.3 Network QoS

The growth of the internet at phenomenal rate is sustained due to its adoption of the best effort model with packet switching technology as its design principles. However, it is also evolved as a platform for numerous business and time critical applications that demands service guarantees from the network resources. In order to provision QoS in internet, researchers modeled two approaches- the finer service differentiation based on individual flows and a coarse grained QoS based on aggregate classes of traffic. The initial approach was not adopted due to scalability issues [26] and the latter approach known as Differentiated Services architecture aligns with internet design principles. Each network element in DS architecture is independent to implement QoS control congestion mechanisms like queue management and packet scheduling techniques in accordance to serve the framed policy of the network domain.

The pioneers of the architecture summarize the state of QoS in internet in [26] and discuss the potential problems that need to be addressed. One of the complex problems that still remain as an open question is deriving an upper bound for the delay experienced by a traffic flow belonging to an aggregate in DS network domain. The detailed analysis by distinguished researchers in [22, 24, 25, and 32] provides insight on characterizing overall network delay considering different queuing management and scheduling techniques implementation in each network node and modeling their functions for aggregate traffic flows through the system.

3.4 Disk Modeling

Much of the research efforts are being directed to optimize the different functions of software layers that process an IO request that are either of operating system stack or storage protocol data access mechanisms. Precise modeling of disk based storage systems based on their mechanical components is strictly limited by vendor specific standards and their involvement of RAID subsystems. It is evident from the works of [34, 35] where authors have attempted in analytical modeling of disk response time based on the functions of device hardware geometry. The equations obtained through their work bounds the generic response time and results vary with different hardware devices.

CHAPTER 4

QoS PROVISIONING FRAMEWORK WITH DELAY BOUND MODEL

4.1 QoS Provisioning Framework

The objectives of proposed QoS framework are to provide end-to-end guaranteed delivery and deterministic delay bound for completion of an interested task. An interested task is defined as IO request generated from a specific process or a class of task that is produced by all processes which needs superior treatment. The method of determining an interested task and associated process depends upon identified parameters of an application that are derived from careful evaluation of its behavioral characteristics and requirements. This research work assumes that such procedures are well laid down and focuses on provisioning appropriate system resources to deliver better service for the interested task. An important property in furnishing QoS for networked storage systems is to preserve identity of a task as it changes nature of denomination i.e. from IO request in storage system to packets in a network domain. To satisfy the property, the policy needs to include procedures to map an IO request of interested task to packets at network layer. For the rest of the literature, an IO request denotes a task and it is the qualified component deserving better treatment.

The constituents within a networked storage that influence the service time of an IO request are components of the IO scheduling system, network system that induces latency for transporting the request and associated data between storage device and the initiator, and processing time involved in retrieving data from the disk subsystem. The author of this research work proposes modification to existing IO request flow path and introduces technical component at the storage server to reflect the desired service for the requested data.

4.1.1 Marking Onward Journey

The IO request generated by a process is first received by the system call interface of the kernel and is turned into an appropriate file system IO call. From file system point of view, the file is stored in different blocks of data and initiates mapping layer services to retrieve the block location which will be passed as an argument to the block layer with the IO call. The block layer issues individual commands for each requested block of data in an IO request.

Within the storage system, the initial stage for contention of resource takes place where all block commands from different IO request are competing with each other to access the storage device. The role of managing resource contention is handled by the IO scheduler system. Its operation is to manage incoming block commands and schedules the dispatch of the request for service. This is the first place where assurance has to be provided for the commands from interested IO request are met with special service.

The framework calls for adoption of the Time-sliced Complete Fair Queuing (CFQ) scheduler that is capable of providing preferential service to IO request by allocating varied service times based on request owner and passed parameters [46]. As with any application, a read IO request completion is more significant than writing data to the disk as further processing of application depends upon retrieval of requested data [40]. For example, in multimedia stream, playing a video stream requires data frames in order and it is of no use to buffer the latest data frames without completion of previous requested frame. These kinds of IO requests are classified as synchronous IO where calling process has to wait and be idle without proceeding for further data processing. Another characteristic of write IO request is that once intended data is received in kernel memory, it can be acknowledged without committing to disk which does not stall the

application activity [40]. This kind of requests is termed as asynchronous IO and is executed before its expiration and availability of system resources.

The CFQ provides three classes of priority- Real Time, Best Effort and Idle. The processes associated with the real time class receives foremost service than from IOs of other classes [46]. By default all processes' request belongs to best effort class and idle class are reserved for system calls. Within each class, eight sub-levels (0-7) of priority are available with 0 being highest service class. The framework requires interested task to be associated with either higher real time or higher level of priority in best effort class and does strictly recommend specific priority level as it requires analysis of the entire system IO request pattern and insists a higher degree of service than the rest of the workload. The research also considers a Budget Fair Queuing (BFQ) algorithm for the scheduler system as it claims to over-come short comings of CFQ on fairness and delivers differential services based on sectors of data to be fetched [17]. The BFQ is still under the active development phase and is currently being tested for its behavior with different workloads.

The dispatched IO request by the scheduler is handled by the SCSI subsystem which creates equivalent CDBs that are transferred to iSCSI for encapsulation with its header data. This encapsulated request is referred to as a PDU and it is qualified for transporting across the network by TCP using appropriate socket call functions. As the PDU arrives at the IP layer, it might be segmented depending on size by TCP and the policy must enforce identifying packets belonging to the interested task and the IP header information should be marked with required DSCP values. The initial task of classifying packets is handled within the netfilter architecture framework. The rules have to be applied in the pre-routing chain of the mangle table that modifies the header information to desired values [38]. The last stage of the request during its

traversal within the stack of the initiator involves implementing a queuing discipline that is capable of servicing marked packets with higher service rate than rest of the packets which ensures packets of interested request spends minimal time in the output queue waiting for transmission. Among the available queuing mechanisms, the frame work, at minimum recommends implementation of pfifo_fast queuing discipline [37].

4.1.2 QoS Aware Network Domain

This section of proposed research work aids in the selection of technical components by considering the characteristics of a network domain consisting of multiple nodes handling diverse traffic streams in order to realize the concerned packet's special requirements. The policy includes signing a contract with the network service provider describing supported quantity of traffic flow with assured packet delivery and deterministic maximum bound delay. From the network service provider's perspective, the nodes must be capable of differentiating packets to provide preferential treatment by reserving resources for conforming to a high degree of service. The per-hop behavior of a node to deliver such service is standardized as the Expedited Forwarding procedure under Differentiated Services QoS architecture. For a synchronous IO request over the iSCSI network across multiple nodes demanding strong delay bounds, policy framework incorporates EF behavior and it is validated by modeling its behavior [27]. For an EF node to hand out performance in accordance to formulated behavior, the traffic flow for the desired application must be disciplined as per the service level agreement at the network edges. The traffic conditioner must inject identified packets no more than the agreed rate and also need shape the burstiness of flow and include buffer mechanisms to hold misbehaved packets of a flow.

4.1.3 Restoring QoS for Return Path

For the real-time applications on iSCSI storage network, read request from initiator are generally classified as interested task and the number of such requests issued are well within traffic flow rate. At the storage server, the framework introduces new system component within network stack to provision QoS for the response traffic. Also, the requested data retrieved from storage device can vary in size and its traffic flow has to be controlled. The research considers a token bucket filter for conditioning the response traffic at the storage server to preserve QoS and its parameters are studied.

4.1.4 QoS provisioner

At the storage server, the QoS provisioner system framework is responsible for examining differentiated packets for an IO request and tracks IO response while furnishing identical quality of service on its return journey by marking and shaping response data traffic flow. The system adds no overhead to system response delay as it is incorporated within network stack and operates under kernel mode.

4.1.4.1 Netfilter hooks

These are set of functions assisting policy framework to track original IO request from incoming packets to map and log values for the restoration process to deliver QoS for the data path. As packets are received from the NIC card driver, they are inspected for marked bits in the DS byte field and if they carry iSCSI PDU header information or not. This operation can be performed pre-routing stage using u32 match module of iptables [38]. Within iSCSI header, interested bytes are Initiator Task Tag value which denotes unique task in an iSCSI session and the matched value has to be recorded into a log file [8].

4.1.4.2 iSCSI Response Mechanism

The iSCSI protocol at the storage server provides target services that manage the connected Logical Units (ex. storage hard disk) and satisfy incoming requests. In this research work, the iSCSI interface with the sockets to use transport protocol TCP services needs adoption of either qsockets calls [40] or invoking socket options to mark the DS byte field with appropriate DSCP values. The iSCSI decapsulates PDU read request and transfers CDB to the SCSI driver. The SCSI driver processes the command and retrieves the data from the hard disk. The fetched data is handed over to the iSCSI layer which encapsulates with built iSCSI response PDU header data structures. At this phase, certain functions are augmented within iSCSI to enforce the policy. It checks the log file to know whether it is responding to a logged task value or not. If it returns the data to logged task value, iSCSI invokes socket calls with arguments to mark DS byte field with required DSCP values. The research also considers implementing separate socket libraries and sockets, called qsockets, to achieve the requirement which provides additional options for enforcing QoS services [39].

4.1.4.3 Traffic Shaper

The size of the requested data varies with each IO request and bursty transmission of traffic even differentiated packets marked to receive special attention can experience congestion and additional delays by competing traffic flows for network resources. To forbid such circumstances to an interested task, as mentioned earlier, the policy must include service level agreements between the network service provider and server hosting facility. The key negotiated parameter is the required bandwidth to accommodate the traffic flow which is determined based on the application IO characteristics and expected service delays. The analyzed storage server in this research has to comply with agreed traffic flow rates and the implemented technical system

that facilitates to accomplish traffic conditioning which is based on Token Bucket Filter (TBF) algorithm.

With the careful selection of token bucket parameters, it allows ideal implementation of traffic conditioner functions. The proposed research work calls for Token Bucket implemented for classified packets and shape its rate of transmission. The shaping function allocates buffer to queue the packets that exceed the agreed data rate instead of dropping misbehaved packets.

4.1.5 Functional Path of Data Flow

The following diagram illustrates the path taken by IO request from a preferred application for which storage devices are across the network connected by iSCSI.

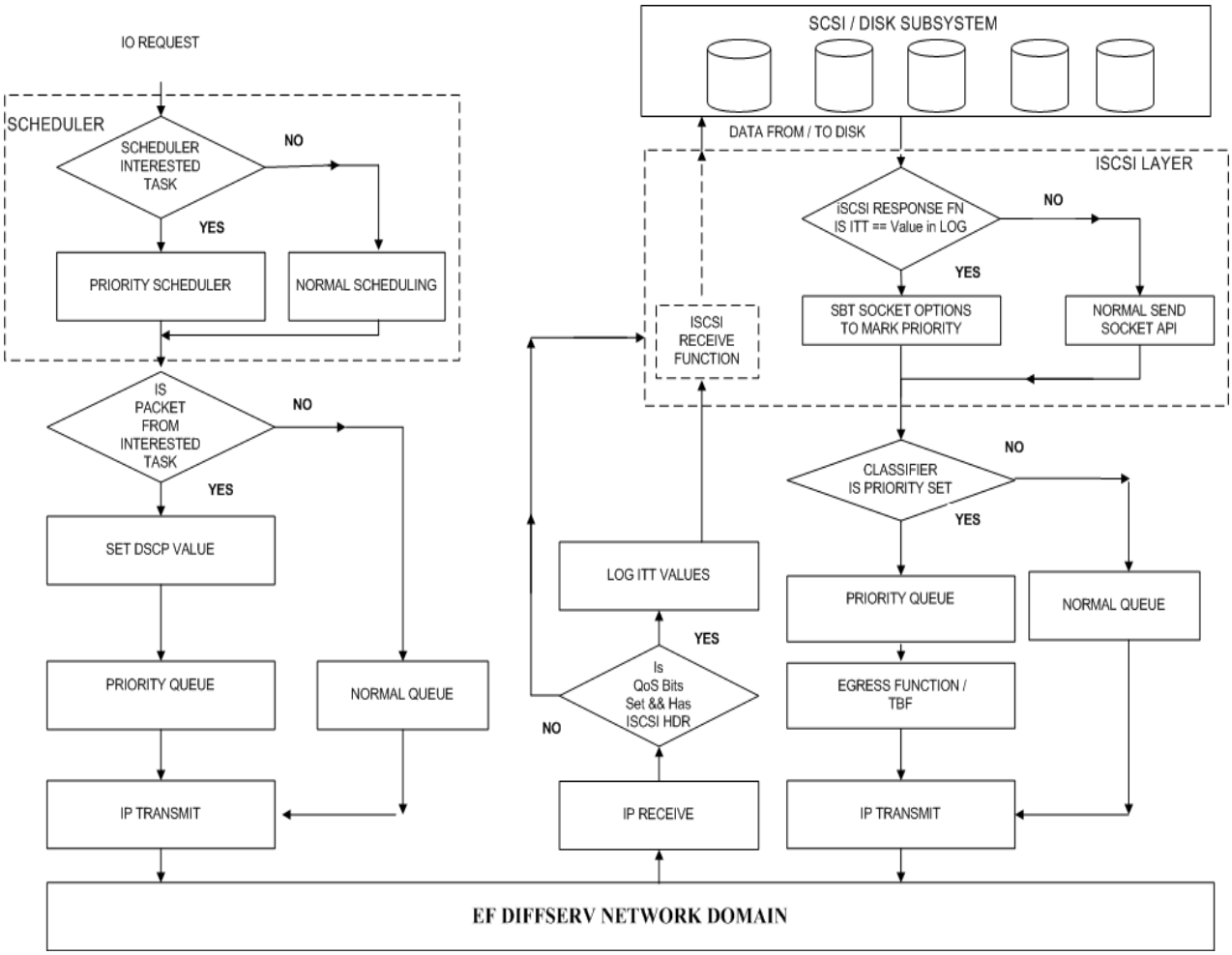

Figure 11 Functional flow diagram of proposed QoS provisioning framework

4.2 Analytical Modeling of Maximum Delay Bound

In this section, the author presents an analytical model that provides bounds for the maximum delay experienced by the iSCSI SAN traffic adopting proposed QoS policy framework. The total delay is defined as the time taken by an IO request to get completed i.e. it is determined from the point an IO request reached IO scheduler system which is issued by the filesystem until iSCSI initiator receives status message from the iSCSI storage server that all requested data is successfully transmitted. The model is based on the premise that for each read IO request, the data has to be fetched from the storage device and does not hit any of the cache located along the layers of path traversed by it. It also demands that the IO scheduler at the storage server for the storage device does not further reschedules or prioritize SCSI commands which are queued at the disk driver interface and implements a FIFO queue with strict service deadlines.

4.2.1 Delay Components

The iSCSI storage area network can be abstracted as three major system components that influences the IO response time.

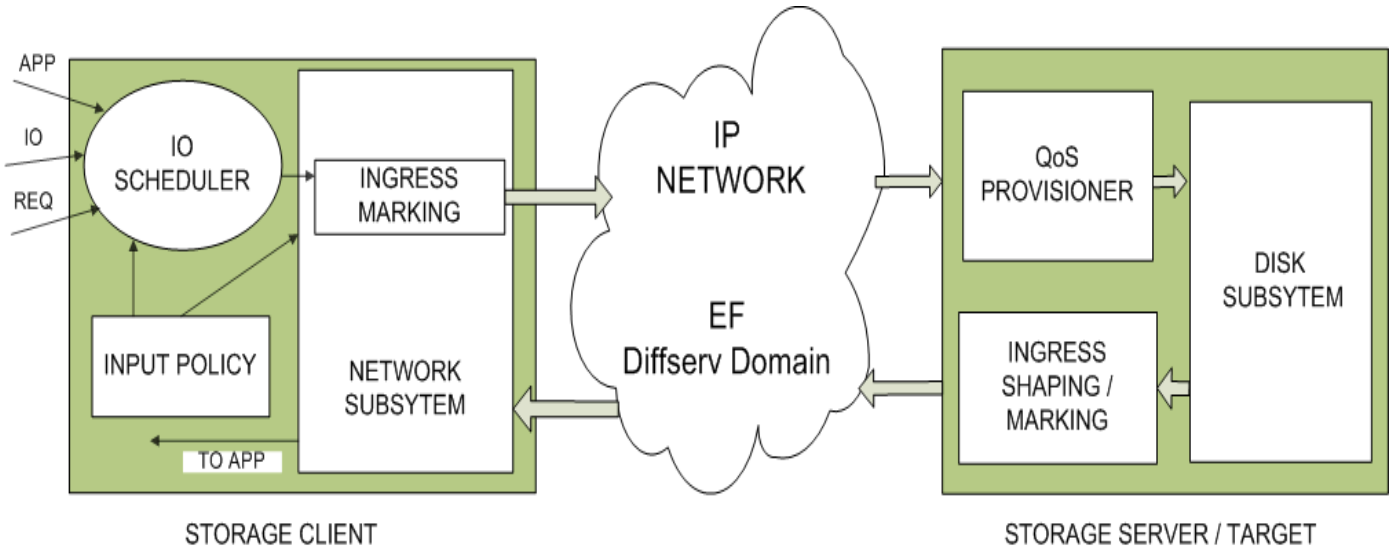

Figure 12 Abstract view of delay components

The three components are IO scheduling system, network domain based on Differentiated Services Architecture and storage device entity. Each of these elements has their authority over the generated IO request and based on their work load and system resources, they impact the time taken for processing IO request as it traverses to and forth between the initiator and server. The experienced latency can range from few milliseconds to several seconds depending on the data size and network state which adversely affects the application performance and end user perception about the service. The application performance guarantees are evaluated in this research work through an analytical model for determining the worst case delay for completion of request which involves analyzing design and operational characteristics of individual components. The end-to-end delay (T_{IO}) is expressed as sum of the delays experienced in IO scheduling system (T_{SCH}), aggregate of time spent in each node of a DS network domain and disk subsystem response time (T_{DISK}) to deliver the data to server kernel from its storage devices.

End-to-End delay = IO scheduling system + DS QoS Network Delay + Disk Subsystem Response Delay

$$\mathbf{T_{IO} = T_{SCH} + T_{NW} + T_{DISK}} \quad (1)$$

In the following sections, the author details the parameter that bound the worst case delay and derives expression for each of the individual component listed in equation (1).

4.2.2 IO Scheduling System

The purpose of IO scheduler is to inspect IO requests based on the type of request whether it is a read/write operation and maintain them in a list of queues where they are sorted and merged depending on the spatial location of accessed data and finally schedules the dispatch of request based on configured policy framework. Such a system is schematically depicted in the following diagram.

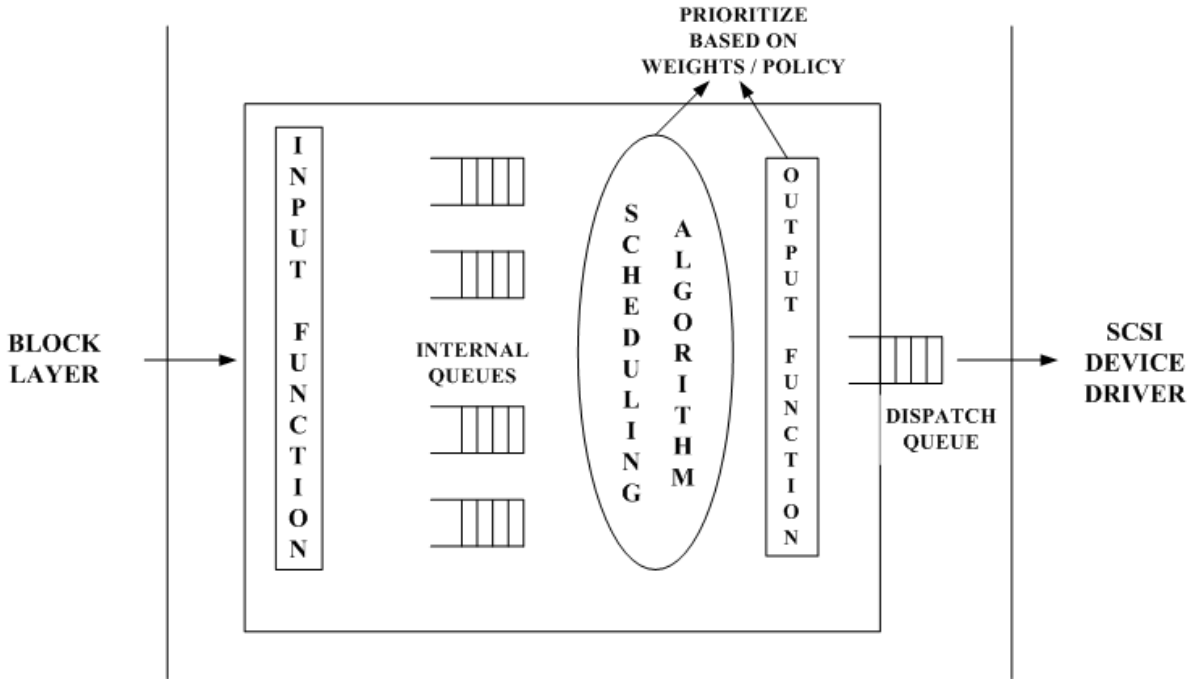

Figure 13 Schematic model of scheduler system

Time spent in scheduler T_{SCH} encompasses both the probabilistic component that defines time spent in the queues of the scheduler waiting for service and deterministic time taken for a request to be serviced which is a constant parameter. However, the current scheduling algorithms implemented in schedulers have negligible servicing time i.e. to make a decision to which request to pick from the queue and place it in the dispatch queue. Therefore, T_{SCH} denotes the time spent by the request in the system queue awaiting service.

The scheduler can be characterized as system in a steady state where the number of departed requests equals the number of the requests arriving into the system. Each of the request is serviced in deterministic time. The system behavior is in accordance with the properties defined by the Little's law. With relevance to the system under consideration, the theorem can be stated as the average number of requests in the system is equal to the product of average arrival rate of the request and average time taken by system to service the request.

$$N = \lambda * T_{SCH} \quad (2)$$

N = average number of requests waiting in the system to be serviced

λ = average rate of request arrival to the system

Re-writing the equation (2), we can obtain the average time spent by a request in scheduler

$$T_{SCH} = N / \lambda \quad (3)$$

To find $T_{SCH,}$ internal structure of the scheduler is viewed as a queuing system that is denoted in Kendall's notation as M/D/1/L queuing behavior.

Where,

M = denotes the Markovian arrival process that characterizes input as exponentially distributed and independent events

D = Deterministic service time, constant for each request to be served

1 = Scheduler has one server to process the incoming request

L = denotes the finite maximum number of input request that can be queued in scheduler

There is numerous amounts of research work being conducted to model different queuing system behavior and the author of this research work incorporates the result of M/D/1/L queuing system from [31] after extensive literature study.

$$T_{SCH} = \sum_{k=1}^{L+1} k * P_k^{(L)} / \lambda \quad (4)$$

Where,

k = number of requests waiting in the queue for service

L = maximum length of the queue,

$P_k^{(L)}$ denotes the steady state probability for the queuing model M/D/1/L

Equation (4) gives the average time spent by each request in the IO scheduler system.

Figure (14) illustrates the relationship between the probability, average number of IOPS arrival rate and the delay experienced in the system for a partial filled queue of arbitrarily chosen value

(K=100). In Linux based schedulers, the maximum queue length (L) is defined by one of the descriptors “nr_requests” of IO request data structure.

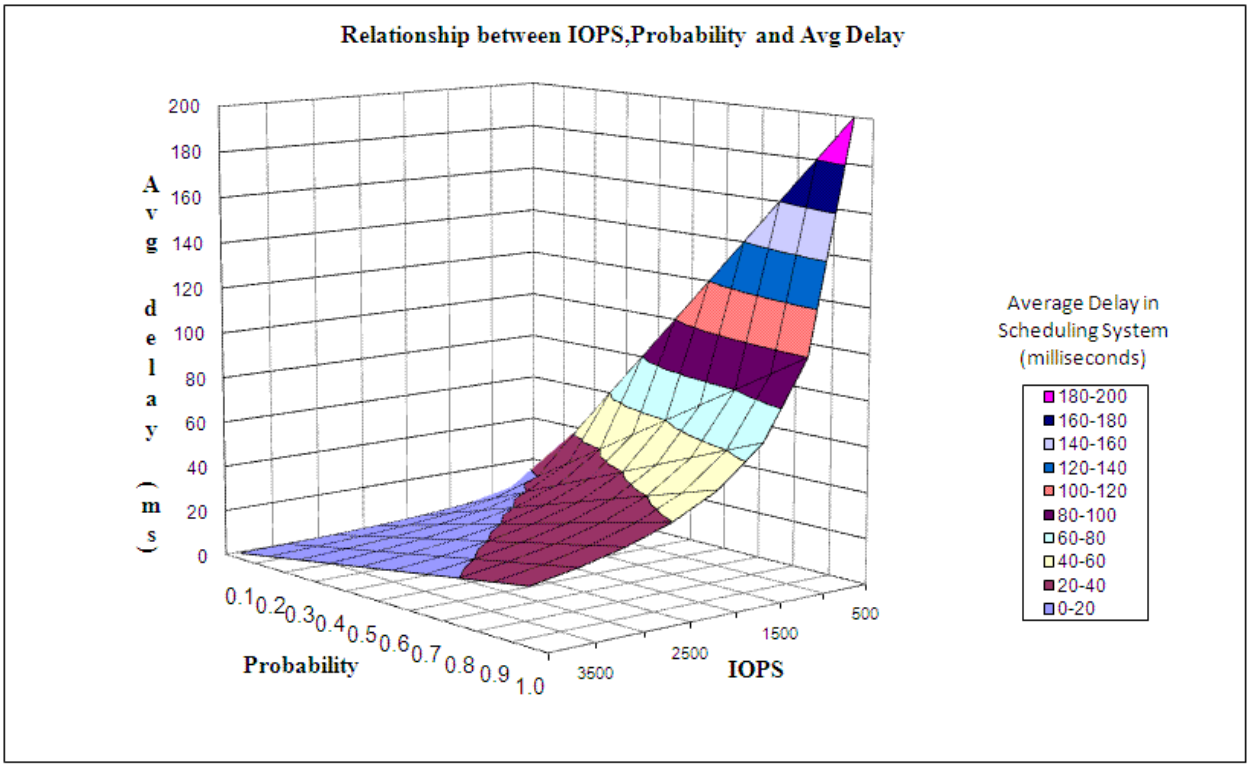

Figure 14 Average delay for IO request in scheduler system

The equation might seem to draw the conclusion that by increasing average IOPS, the time spent in the system can be minimized as they are inversely proportional to each other. However, more detailed study conducted by the author of the research about IO scheduler characteristics elucidates that it is the average time for each IO request with respect to their arrival rate and several key system design parameters like achieving fairness, efficiency and memory constraints for queuing request and maintaining state variables of queue, underlying the output medium bound arrival rate into the system. The arrival rate is also controlled by the number of requests of each thread of application can be kept in pending status before issuing next IO request.

4.2.3 EF DiffServ Network Model

Estimation of network delay has been a subject of extensive research that needs to comprehend overall system behavior and diverse characteristics of an individual system's components which are operating under different work load conditions over constantly evolving system architectures. From a performance perspective, network system is measured in terms of delay experienced by the traffic flowing through it. This parameter is dominated by crucial system resources like link bandwidth, buffer space and computational power of a node and traffic management algorithms like queue management and scheduling techniques implemented by each node. The existence of various network topologies and inexplicable nature of traffic flowing into the network, forces delay estimation models to consider design environments constrained by the objectives of problem under investigation [24, 26, and 30]. In this research, Differentiated Service network architecture with nodes conforming to EF behavior is analyzed. Even though service differentiation and aggregate behavior characteristics are well standardized, the problem of precise modeling of delay bounds for qualified traffic flow still remains as an open challenge due to above mentioned parameters [24, 28]. With regards to the proposed QoS framework, the network can be schematically represented as follows.

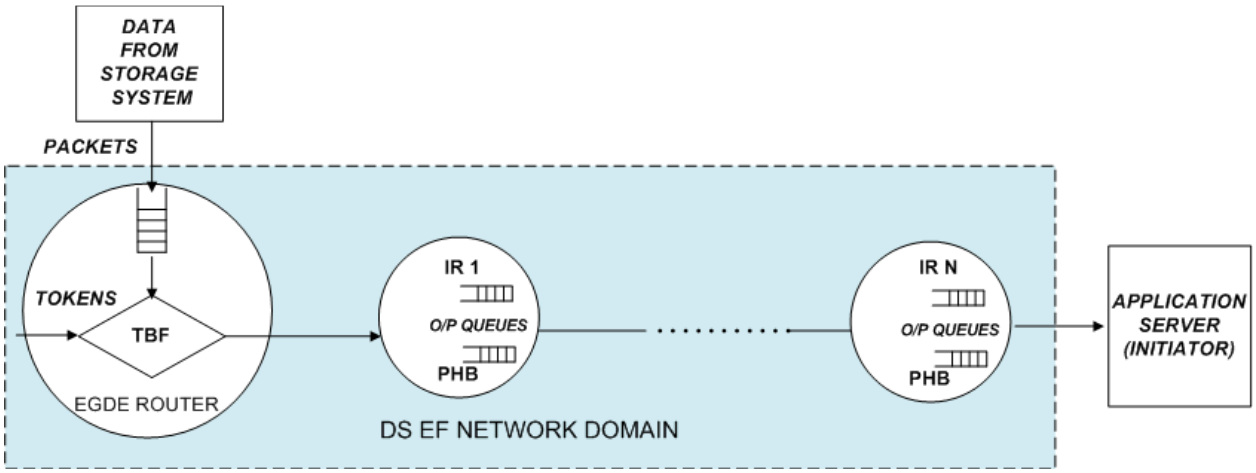

Figure 15 Abstract model of EF-DS network domain

In the above diagram, the storage system is represented as a data emitting source and its input rate into the network is disciplined by Token Bucket Filter (TBF) implemented at the storage server or edge of the network and each network element functions as guaranteed rate servers that adopts scheduling and queuing mechanisms to impart preferential treatment for qualified traffic [27]. The author assumes that the traffic flows offered from other resources are confined to maintain the steady state operation of a network and its resources are available to accommodate the aggregate traffic classes of EF behavior. It also makes assumptions that micro-flows within EF aggregate are not reordered and the link output rate at each node exceeds the offered aggregate traffic flow rate. With these premises, the author presents a simple analytical model to estimate the end-to-end delay (T_{NW}).

The cumulative delay for a packet can be broken down into following components

$$T_{NW} = N * (T_{prop} + T_{node}) \quad (5)$$

Where,

N = number of hops the packet has to travel through network between its source and destination

T_{prop} = time taken for single bit to travel across the physical link of distance ‘d’ meters at given propagation speed ‘s’ meters/sec between two network nodes

$$T_{prop} = d / s \text{ seconds} \quad (6)$$

The propagation speed, s is characteristic of the physical medium. With current link layer technologies, the propagation delay in wide area networks are on the order of few milliseconds.

$$T_{node} = T_{trans} + T_{proc.} + T_{queuing} \quad (7)$$

T_{trans} = transmission delay – amount of time required to transmit all bits of a packet ‘L’ into the link of transmission rate R

$$T_{trans} = L / R \text{ seconds} \quad (8)$$

L = packet size in bits

R = transmission rate of the link, bits per second (also known as bandwidth)

For the given a network domain, the total transmission delay can be obtained as sum of delays in each node.

$$\text{Total } T_{trans} = \sum_{i=1}^{N} \left(\frac{L}{Ri}\right) \quad (9)$$

In today's internet, it typically ranges from microseconds to few milliseconds.

T_{proc} = time taken by router to process a packet

The router functions include error validation, route look-up, forwarding decision and updating packet header fields. It is dependent on the hardware architecture of the router and introduces delay ranges from nanoseconds to microseconds.

To analyze the queuing delay, the equations (5), (7) can be re-written as

$$T_{NW} = N*\underbrace{(T_{prop} + T_{trans} + T_{proc.})}_{\text{Constant}} + N * \underbrace{(T_{queuing})}_{\text{Variable}} \quad (10)$$

As evident from the above analysis the total network delay can be categorized into deterministic and variable elements. For a given network, the deterministic components are constituted by propagation, transmission and processing delays which are dependent on the physical characteristics of the link and hardware architecture of each network node. The variation in the total delay is heavily influenced by the queuing delays at intermediate routers and it is a subject of extensive research to optimize the design of any servicing system [25, 30].

Queuing delay in a network system is observed in terms of traffic intensity flowing through it and is bounded by the finite resources available at each network component. In regards to the

network under consideration, the queuing delay can be expressed as the delay experienced at ingress node shaper and per hop queuing delay due to competent traffic flows.

4.2.3.1 Token Bucket - Ingress Traffic Shaper

The traffic generated by the storage server is intrinsically limited by the data size and the average rate of transmission depends upon the application IO characteristics. In order to guarantee the packet delivery and deterministic delay bounds, the long term average rate has to be conformed to traffic rate as per service level requirements and the burstiness must be accommodated with appropriate buffering mechanisms. The Token Bucket algorithm can be implemented to regulate traffic and shape the excess traffic arriving in burst. The traffic conditioning behavior of token bucket can be modeled as Linear Bounded Arrival Processes (LBAP) with the following parameters [43].

C capacity of the Token bucket

ρ token arrival rate i.e. output traffic conditioning rate

t time interval over which rate is averaged

Then work load offered by the ingress node to the network domain i.e.

Total number of bits transmitted over time-interval t, $= \rho t + C$ (11)

For a generated packet of N bytes, it is checked for the availability for the token in the bucket. If $N < C$, it is transmitted instantaneously, otherwise if it a fraction of available tokens, then it can be either buffered or dropped as per implemented policy. In order to avoid packet loss, the token bucket conditioner can be designed to accommodate traffic exceeding the agreed rate and delayed in data buffer to conform it to the traffic flow as per service level agreement. To achieve objectives of service level agreements, the token bucket design parameters of queue size and

token bucket size needs careful evaluation and trade-off between available resources and allowable delay for intended traffic intensity generated by the IO requests.

The estimated latencies for traffic is subjected to shaping are calculated using following method.

$$\text{Delay} = \text{Queue Size} / \text{Transmission Rate} \quad (12)$$

The following graph depicts queuing delays for token bucket design with bucket size of 200 KB for various practical transmission rates of the data.

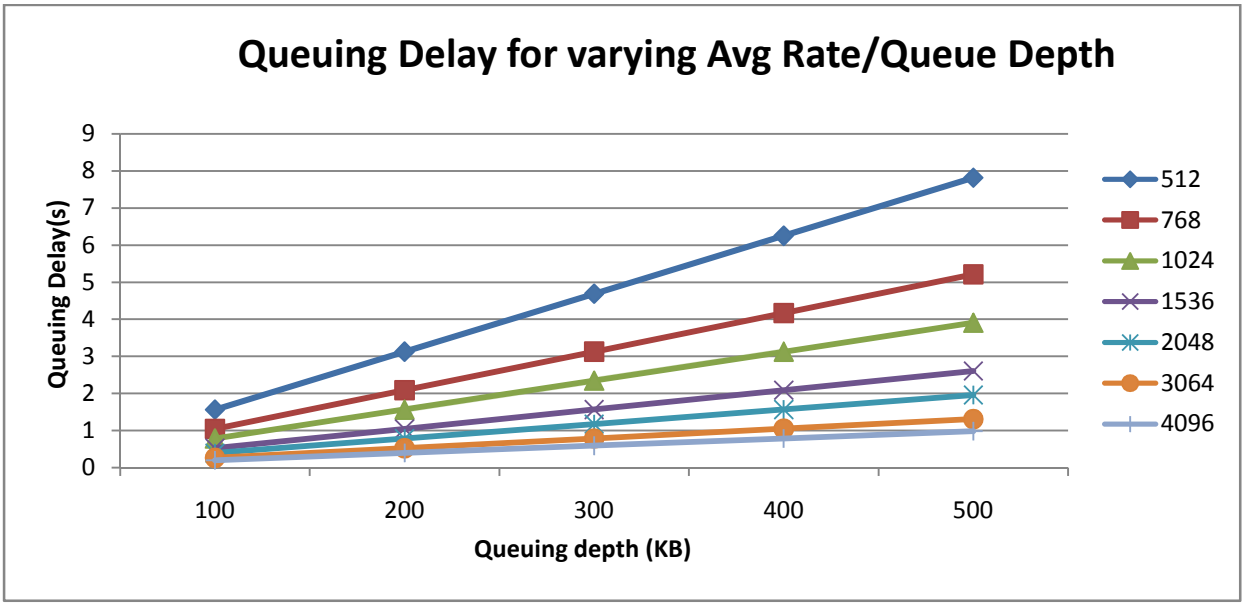

Figure 16 Queuing delay in traffic shaper

4.2.3.2 Node Delay

The mechanisms implemented in a node, are designed to exert control at packet level and they have to satisfy mutually contradictory objectives of fairness among multiple traffic flows and efficiently utilization of the link bandwidth. The issue becomes more complex where the node has to serve the best effort traffic class and guaranteed traffic flows conforming to DS policy at the same time. Upon receiving a packet, several functions are involved in handling the transmission of the packet to the egress interface. The predominant delay occurs due to the time spent by the packet at the output queues of the node waiting for its service. The problem is

addressed by two closely related mechanisms - Active Queue Management (AQM) which manages the queue size and proactively operates to avoid congestion and the second one - packet scheduling algorithm determines which packet to be served from the queue. The implementation of AQM with Random Early Detection (RED) and its variants is desired and must be considered for stability of the network during periods of congestion [49]. Even though there are wide range of schedulers available in the literature, due to implementational cost constraints, very few of them like WFQ and it variants, Class Based Queuing (CBQ) and its variants are implemented in the routers [43].

The author assumes that each node in the domain follows implements same set of AQM configuration principles and scheduling algorithm to deliver differentiated services. Even in such environments, the problem of precise determination of delay bounds in a node serving both best effort and differentiated traffic flows still remains an open challenge which is primarily due to bursty nature of traffic flowing through the network [28, 49]. Thus the resulting node, would have implemented many FIFO queues with RED queue management techniques and scheduling algorithm supporting differentiated services.

Under ideal isolation between different traffic flows, the delay bound of the interested traffic flow shaped by token bucket traffic shaper parameters of bucket size C (bytes) and configured rate of R bps is borrowed from the literature [28].

$$\text{Delay} = C / R + \text{Scheduling Service Rate} \quad (13)$$

Where service rate for,

$$\text{FIFO scheduling} = \text{MTU} / \text{BW of the link} \quad (14)$$

$$\text{WF2Q scheduling} = \text{MTU} / \text{BW of the link} + \text{MTU} / R \quad (15)$$

In a perfect model, the task of calculating total queuing delay is trivial where it is represented as sum of delays experienced in each individual node of the network domain. However, delay in a node is influenced by various factors and significantly determined by input burstiness of traffic flowing through the network system. Again, the author borrows the important result from [43] to obtain maximum end-to-end delay for traffic flowing through a connection of WFQ schedulers.

Delay bound for traffic flow serviced by WFQ

$$= \text{C / Sch. Service Rate} + \sum_{i=1}^{N-1} \left(\frac{\text{MTU}}{\text{Sch. Service Rate}}\right) + \sum_{i=1}^{N} \left(\frac{\text{MTU}}{\text{R}}\right) \qquad (16)$$

The equation gives worst case delay for traffic flowing through a series of WFQ schedulers that is shaped by above mentioned token bucket parameters.

4.2.4 Disk Storage System

The disk system is the slowest constituent of any storage system (which includes DAS or SAN or NAS) due to processing delays of mechanical components. The crucial property of magnetic hard disk is the ability to seek sectors in random manner. Although this functionality qualifies for many first tier storage systems, randomness can lead to very poor performance with the random IO work load pattern that keeps the head of the disk to jump back and forth when seeking the sector location rather than performing actual data transfer. Even though the implementation of IO scheduling algorithms alleviates randomness in IO pattern, the response time for completion of an IO request is still greatly influenced by hardware geometry and characteristics of the disk. From previous research works [4, 15 and 33], it is difficult to predict the precise nature of IO work load pattern for a given application as it is influenced by numerous parameters. The disk access pattern by application varies with time, users, and burstiness over a short time interval and type of request over seeking sectors of data based on spatial location on

the disk. Beyond a generic estimation of data access pattern, detailed analysis are performed to determine the IO throughput required by application to operate in satisfactory manner. This motivates research for modeling maximum response time to complete IO for real-time applications that require stringent response delays.

In this section, the author of the research formulates a generic model to derive response time taken to complete data transfer in terms of sectors that incorporates mechanical parameters of the disk. The model assumes zero hit-ratio for the disk buffer cache and all requested data by read operations has to be fetched from the disk. Also, the time taken by the disk controller to parse the command is considered negligible as they are implemented in electronic circuits. The total time taken is combination of seek time, rotational delay, transfer time and disk controller overhead.

$$T_{DISK} = \text{Avg } T_{seek} + \text{Avg } T_{rot} + T_{Transfer} \quad (17)$$

T_{seek} average time required for the head arm to move to the desired track

T_{rot} average time taken for the head to move desired sector

$T_{Transfer}$ time required to transfer requested number of sectors of data from the disk to controller buffer

The equation () gives the total time taken for transferring the data from the disk and each of the individual values are derived in [35] detail that incorporate hardware geometry and physical properties of magnetic media. Almost, all modern hard disk implements Zone Bit recording (ZBR) layout for disk structure and are formatted with each sector to hold maximum of 512 bytes of data [3, 35].

The following simulation was performed to analyze the worst case response time for the disk read time under various IO workload pattern. The test bed consisted of Dell Optiplex 745 system running Linux kernel 2.6.25 that is attached with Samsung SATA disk drive. The workload was

generated using Flexible IO (fio) tool that provides options to manipulate request parameters. Each test was repeated five times to reduce the error margin and response time was observed for varying IO request parameters like different IO request patterns, type of requests and application IO queue size. For the entire simulation, a block size of 4096 bytes and direct IO calls were used that bypasses buffer cache and invalidates it before creating the test file.

The first test involves evaluation of the disk response time for different IO request patterns when they are executed individually and in mixed environment. The latter pattern simulates a real-time workload and individual test conducted earlier, serve as a benchmark for the disk performance. The IO work load was composed of three threads issuing synchronous read IO each following one of the request pattern- sequential, random and a thinker access pattern that stalls for certain time between two concurrent IO requests. The threads are run individually and mixed environment and the following graph depicts the response time for a given file size of 8 MiB.

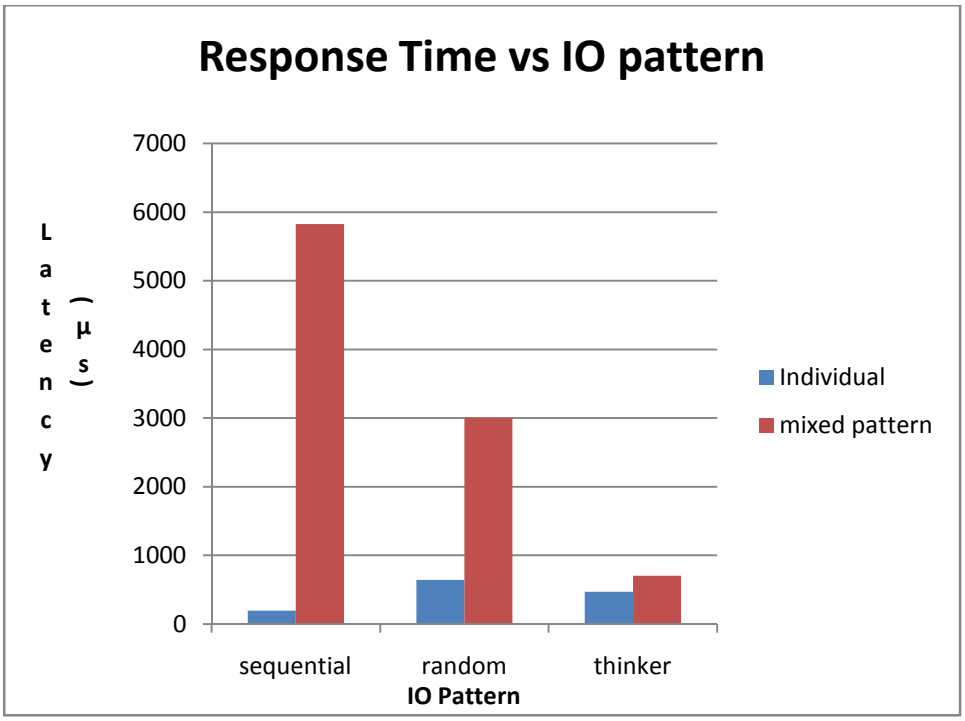

Figure 17 Response time for varying IO workload pattern

As modeled, each individual request pattern, the completion of task takes less time than when the threads are run together. The completion latency of sequential read task is affected adversely as their request is stalled due to other interfering request that forces the disk to seek sectors from location of the drive. This result reinforces the need of IO scheduler to organize request to improve the performance by minimizing disk seek times and at the same time achieving fairness among all threads issuing IO requests. At the SATA disk level, Native Command Queuing (NCQ) techniques are implemented to further optimize the mechanical workload that reduces seek time.

The second test was performed to observe response time for different application IO intensity while performing read operation of different file sizes. IO depth is the IO queue size of number of request that can be kept pending at the application level and also indicates the IO intensiveness of the application. To accommodate application issuing multiple IO request, asynchronous read IO calls are used.

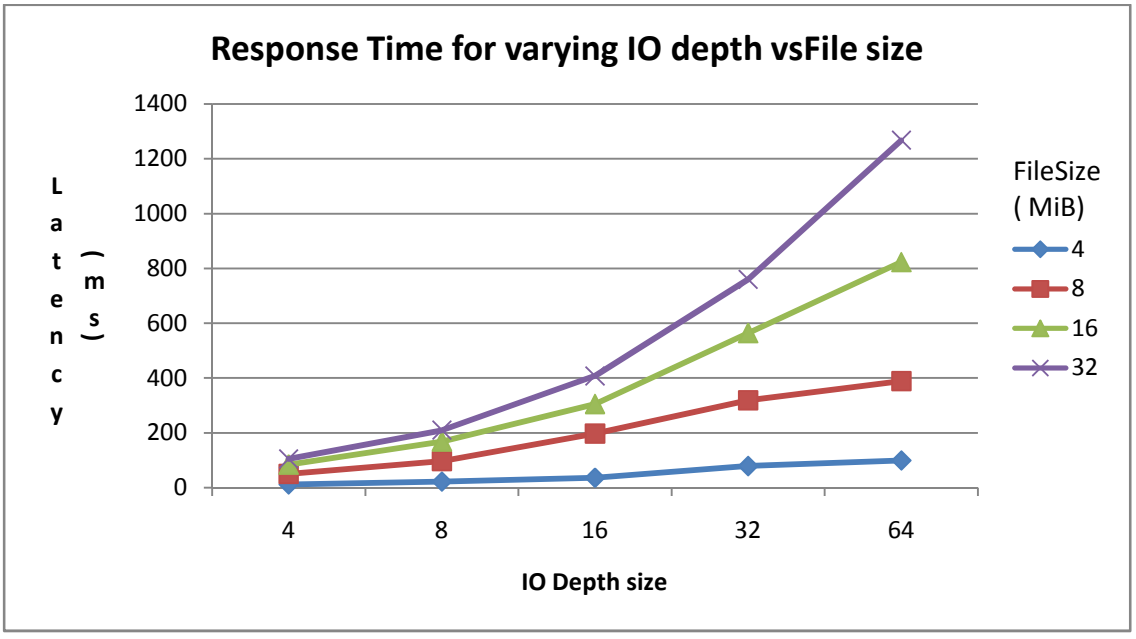

Figure 18 Response time for different IO depth and file size

The threads with different IO depth are run together for a given file size and results indicate that more latency is experienced by IO heavy application as file size increases, which causes disk head to move back and forth seeking sectors at random location and at the same time disk resources are competed for by other threads. This test estimates the suitability of specified mechanical parameters of disk drive for expected IO activity.

4.3 Summary

In this chapter, the author proposed a QoS policy framework to achieve end-to-end performance guarantee for the desired IO traffic in iSCSI SAN environment. The research introduced QoS provisioner mechanism at iSCSI target server for furnishing QoS for the requested data by the IO with no additional overhead protocol structure or need of implementing new system component. Subsequently, the framework is evaluated by analytical modeling of individual system components for maximum delay bounds and compared the results with the existing literature.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusion

In this thesis, a QoS policy framework is proposed to achieve end-to-end QoS for iSCSI storage data traffic. The study analyzed structural components of policy and identified places of bottleneck where service can be degraded due to resource contention and recommended provisioning of resources to satisfy application performance requirements without starving other contending traffic flows. To furnish symmetric QoS for request and data, the author developed QoS provisioning algorithm at iSCSI storage server integrated with network stack and iSCSI functions. Based on conducted research and analysis, it is shown that the developed QoS component does not introduce significant additional delay as it operates in kernel mode avoiding the context switch of CPU and salient design features are that it does not add overhead to the data payload and while not modifying underlying protocol semantics. The implementation of this policy delivers a guaranteed transmission of data belonging to traffic flow of an interested application.

The delay experienced by the preferential traffic in the framework is evaluated analytically and equations for predicting maximum end-to-end latency were obtained. The model incorporates the relationship between the IO request in storage subsystem and generated IP packets for the network system. The equations are validated using queuing simulators and FIO tool and results are presented for delay bounds under different scenarios.

5.2 Future work

The scope of future work in this research area is multifold as it encompasses two broad technical domains- storage and network.

The immediate goal would be to identify additional system parameters that aid the algorithm to provision appropriate QoS for the requested data while traversing back to the application. This brings in the ability to dynamically furnish resources and gives the capability to sense congestion in the network system.

The conducted research was operating systems specific and further steps can be taken to extend the ideas of QoS provisioner to be independent of end host platforms. This would greatly enhance the applicability of the framework in virtualized environment.

Another active research field is to explore possible applications of flash storage media as a primary storage device and enhancing data access techniques involving mechanical components that reduce seek time.

The existing IPv4 network addressing architecture will be sub-sequentially replaced by IPv6 which brings out numerous issues to be addressed. From network QoS perspective, IPv6 Label Switch Architecture (LSA) - a proposed standard to IETF can be explored for providing QoS performance.

As the enterprise IT infrastructure prepares to embrace cloud computing architectures, the proposed policy framework can serve as prototype model for delivering different classes of service and additional methods of design framework can be developed to meet varying end requirements.

REFERENCES

LIST OF REFERENCES

[1] Renato John Recio, "Server I/O Networks Past, Present, and Future" in Workshop on Network-I/O Convergence: Experience, Lessons, Implications (NICELI) SIGCOMM Conference, August, 2003

[2] P. Sarkar, K. Voruganti, K. Meth, O. Biran, and J. Satran, "Internet Protocol Storage Area Networks" IBM Systems Journal, July, 2003

[3] J. Buttress and D. Reinsel, “Worldwide Hard Disk Drive 2005-2009 Forecast: The Growth Is Sustainable, but the Industry Is in Transition”, Technical Report 33432, International Data Corporation, June 2005.

[4] W. W. Hsu and A. J. Smith, "Characteristics of I/O traffic in personal computer and server workloads" IBM Systems Journal, 2003, Vol. 42, No 2

[5] Stephen Aiken, Dirk Grunwald and Andrew R. Pleszkun, "A Performance Analysis of the iSCSI Protocol" in Proc. of 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, April 2003

[6] Khosravi, H.M., Abhijeet Joglekar and Ravi Iyer, "Performance Characterization of iSCSI Processing in a Server Platform", in Proc. of 24th IEEE, International Performance, Computing, and Communications Conference, April 2005, pp. 99 - 107

[7] Xinidis, D. Bilas, A. Flouris, M.D., "Performance Evaluation of Commodity iSCSI-based Storage Systems" in Proc. of 22nd IEEE / 13th NASA Goddard Conference on Mass Storage Systems and Technologies, April 2005, pp. 261- 269

[8] iSCSI parameters, http://www.iana.org/assignments/iscsi-parameters

[9] Fujita Tomonori and Ogawara Masanori, "Analysis of iSCSI Target Software", in Proc. of International workshop on Storage network architecture and parallel I/Os, ,2004, pp. 25 - 32

[10] Foong, A., McAlpine, G., Minturn, D., Regnier, G. and Saletore, V., "An architecture for software-based iSCSI on multiprocessor servers" in Proc. of 19th IEEE International Parallel and Distributed Processing Symposium, April, 2005

[11] J. Satran, K. Meth, C. Sapuntzakis,M. Chadalapaka, and E. Zeidner, “Internet Small Computer Systems Interface (iSCSI),” April 2004, RFC 3720

[12] S. Seelam, R. Romero, P. Teller and B. Buros, "Enhancements to Linux I/O Scheduling" in Proc. of the Linux Symposium, July 2005, vol. 2, pp. 175-192

[13] John Bruno, Jos6 Brustoloni, Eran Gabber, Banu Ozden and Abraham Silberschatz "Disk Scheduling with Quality of Service Guarantees" in Proc. IEEE International Conference on Multimedia Computing and Systems, Jun 1999, vol. 2, pp.400-405

[14] Yu Zhang and Bharat Bhargava, "Self-Learning Disk Scheduling" accepted for publication in IEEE Transactions on Knowledge and Data Engineering, 2008.

[15] Ajay Gulati, Arif Merchant, Mustafa Uysal and Peter J. Varman, "Efficient and Adaptive Proportional Share I/O Scheduling", HP Labs Technical Reports, 2007

[16] Silviu S., Craciunas Christoph M. and Kirsch Harald Rock, "I/O Resource Management through System Call Scheduling", in Proc. of ACM Operating System Review, Special Issue on Research and Developments in the Linux Kernel, July 2008

[17] Paolo Valente and Fabio Checconi, "High Throughput Disk Scheduling with Deterministic Guarantees on Bandwidth Distribution" submitted to Transactions on Computers, URL: http://algo.ing.unimo.it/people/paolo/disk_sched/

[18] Seetharami R. Seelam and Patricia J. Teller, "Fairness and Performance Isolation: an Analysis of Disk Scheduling Algorithms" in Proc. of IEEE International Conference on Cluster Computing, Sept.2006, pp. 1-10

[19] Junkil Ryu and Chanik Park, "Controlling Network Bandwidth to Support Storage QoS", in Proc. Of Storage Network Architecture and Parallel I/Os, Sep 2007

[20] Zoran Dimitrijevic and Raju Rangaswami, "Quality of Service Support for Real-time Storage Systems" in Proc. of International IPSI-2003 Conference, Oct., 2003

[21] El-Bahlul Fgee, William J. Phillips and William Roberts,"Comparison between a Proposed QoS mathematical Model and other IP QoS models" in Proc. of 6th Annual Communication Networks and Services Research Conference, May 2008, pp. 595-599

[22] Ferrari, T.,Pau, G. and Raffaelli, C., "Measurement Based Analysis of Delay in Priority Queuing" in Global Telecommunications Conference,(GLOBECOM '01), Nov. 2001, Vol. 3, pp. 1834-1840

[23] May, M., Bolot, J.-C., Jean-Marie, A. and Diot, C. "Simple Performance Models of Differentiated Services Schemes for the Internet", in Proc. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies, Mar 1999, Vol.3, pp. 1385-1394

[24] A.Charny and J.-Y.Le Boudec, "Delay Bounds in a Network with Aggregate Scheduling", in Proc. First International Workshop on Quality of Future Internet Services", Germany, 2000

[25] Yuming Jiang, "Delay bounds for a network of guaranteed rate servers with FIFO aggregation" in Proc. of IEEE International Conference on Communications (ICC), 2002, vol. 2, pp. 1149- 1153

[26] Firoiu, V., Le Boudec J.-Y., Towsley, D. and Zhi-Li Zhang, "Theories and models for Internet quality of service " in Proc. of the IEEE, Sep 2002, Vol. 90, Issue 9, pp. 1565 - 1591

[27] B. Davie, A. Charny,J.C.R. Bennett,K. Benson, J.Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu and D. Stiliadis " An Expedited Forwarding PHB", March 2002, RFC 3246

[28] Charny, J.C.R. Bennett, K. Benson, J.Y. Le Boudec, A. Chiu, W. Courtney, S. Davari,V. Firoiu, C. Kalmanek and K.K. Ramakrishnan, "Supplemental Information for the New Definition of the EF PHB (Expedited Forwarding Per-Hop Behavior),March 2002 RFC 3247

[29] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang and W. Weiss, "An Architecture for Differentiated Services", Dec. 1998, RFC 2475

[30] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja and X. Xiao, "Overview and Principles of Internet Traffic Engineering", May 2002, RFC 3272

[31] Prasant Mohapatra and Chita R. Das, "A Queuing Model for Finite-Buffered Multistage Interconnection Networks",in Proc. of International Conference on Parallel Processing,Volume 1, Aug 1993 pp.210 - 213

[32] Matthew Andrews, "Instability of FIFO in session-oriented networks", Journal of algorithms, Volume 50, Issue 2,Feb. 2004, pp. 232 - 245

[33] J. Wilkes. "Traveling to Rome: QoS specifications for automated storage system management", in Proc. Intl. Workshop on Quality of Service (IWQoS'2001), June 2001, pp.75–91

[34] Mustafa Uysal, Guillermo A. Alvarez and Arif Merchant, "A Modular, Analytical Throughput Model for Modern Disk Arrays" in Proc. of Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS-2001), August 15-18, 2001

[35] V. Brocal, M. Masmano, I. Ripoll, and A. Crespo "Bounding Disk I/O Response Time for Real-Time Systems"in Proc. of 9th Real Time Linux Workshop, Nov. 2007

[36] Leonardo Balliache, Differentiated Service on Linux, August 2003,
URL: http://opalsoft.net/qos

[37] Bert Hubert,Linux Advanced Routing & Traffic Control, 2003,
URL: http://lartc.org/

[38] THE NETFILTER PROJECT: firewalling, NAT, and packet mangling for Linux
URL: http://www.netfilter.org

[39] Thomas F. Herbert, The Linux TCP/IP Stack: Networking for Embedded Systems, Charles River Media, 2004

[40] Daniel P. Bovet and Marco Cesati, Understanding the Linux Kernel, O'Reilly, 3 ed., 2005

[41] Christian Benvenuti, Understanding Linux Network Internals, O'Reilly, 1 ed., 2005

[42] W. Richard Stevens, TCP/IP Illustrated-The Protocols, Volume 1, Addison Wesley, 1993

[43] S. Keshav, An Engineering Approach to Computer Networking, Pearson Education, 2004

[44] Alma Riska, James Larkby-Lahet, and Erik Riedel, "Evaluating Block-level Optimization through the IO Path" in Proc. of USENIX Annual Technical Conference, 2007, pp. 247-260

[45] Robert Love, Kernel Korner - I/O Schedulers 1 Feb., 2004,
URL: http://www.linuxjournal.com/article/6931

[46] Jens Axboe, "CFQ IO Scheduler", presentation at linux.conf.au, Jan. 2007,
University of New South Wales, Australia,
URL: http://lca2007.linux.org.au/talk/123.html

[47] M. Tim Jones, “Anatomy of the Linux SCSI subsystem”, 14 Nov 2007
http://www.ibm.com/developerworks/linux/library/l-scsi-subsystem/index.html

[48] Henry Newman , “System Calls and the I/O Path “
URL: http://www.samag.com/documents/s=9367/sam0206g/0206g.htm

[49] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker,J. Wroclawski, L. Zhang "Recommendations on Queue Management and Congestion Avoidance in the Internet", April 1998, RFC 2309