

# PERFORMANCE EVALUATION OF ISCSI FOR IP STORAGE AND TRANSPORT PROTOCOLS

BY

SAJID HUSSAIN

A Dissertation Submitted to  
the Faculty of Graduate Studies  
In Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Department of Electrical and Computer Engineering  
University of Manitoba  
Winnipeg, Manitoba

© Sajid Hussain, July 2004

**THE UNIVERSITY OF MANITOBA**  
**FACULTY OF GRADUATE STUDIES**  
\*\*\*\*\*  
**COPYRIGHT PERMISSION**

**Performance Evaluation of iSCSI for IP Storage and Transport Protocols**

**BY**

**Sajid Hussain**

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

**Manitoba in partial fulfillment of the requirement of the degree**

**Of**

**DOCTOR OF PHILOSOPHY**

**Sajid Hussain © 2004**

**Permission has been granted to the Library of the University of Manitoba to lend or sell copies of this thesis/practicum, to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film, and to University Microfilms Inc. to publish an abstract of this thesis/practicum.**

**This reproduction or copy of this thesis has been made available by authority of the copyright owner solely for the purpose of private study and research, and may only be reproduced and copied as permitted by copyright laws or with express written authorization from the copyright owner.**

# Contents

<b>Acknowledgement</b>	<b>7</b>
<b>1 Introduction</b>	<b>9</b>
1.1 Storage Interface . . . . .	9
1.1.1 Small Computer Systems Interface (SCSI) . . . . .	10
1.1.2 Advanced Technology Attachment (ATA) . . . . .	10
1.2 Storage Systems . . . . .	10
1.2.1 JBOD . . . . .	11
1.2.2 RAID . . . . .	11
1.2.3 Tape Subsystems . . . . .	11
1.3 Network Storage . . . . .	11
1.3.1 Network Attached Storage (NAS) . . . . .	12
1.3.2 Storage Area Networks (SAN) . . . . .	12
1.4 The iSCSI Protocol . . . . .	12
1.5 Motivation . . . . .	13
1.6 Summary . . . . .	15
<b>2 The iSCSI Protocol</b>	<b>16</b>
2.1 Protocol Stack . . . . .	16
2.2 iSCSI Naming and Discovery . . . . .	16
2.2.1 Name, Address, and Alias . . . . .	17

2.2.2	iSCSI Discovery Mechanisms . . . . .	18
2.3	Login Phase and Full Feature Phase . . . . .	18
2.3.1	Connection Allegiance . . . . .	20
2.4	Data Transfer . . . . .	21
2.5	iSCSI Numbering . . . . .	21
2.5.1	Command Numbering . . . . .	22
2.5.2	Status Numbering . . . . .	22
2.5.3	Data Numbering . . . . .	23
2.6	iSCSI PDUs . . . . .	23
2.6.1	Basic Header Segment (BHS) . . . . .	23
2.6.2	SCSI Command PDU . . . . .	24
2.6.3	SCSI Response . . . . .	25
2.6.4	Data-Out/Data-In and R2T PDUs . . . . .	26
2.6.5	SNACK PDU . . . . .	27
2.6.6	Reject, Nop-Out, and Nop-In PDUs . . . . .	27
2.7	iSCSI Error Recovery . . . . .	27
2.7.1	Mechanisms for Error Recovery . . . . .	28
2.8	Message Synchronization and Steering . . . . .	29
2.8.1	Fixed Interval Markers . . . . .	29
2.8.2	Upper Layer Protocol Framing . . . . .	30
2.9	Summary . . . . .	30
<b>3</b>	<b>Related Research</b>	<b>31</b>
3.1	Fibre Channel . . . . .	31
3.2	Performance Analysis of iSCSI . . . . .	32
3.2.1	Efficient implementation of iSCSI . . . . .	33
3.3	Summary . . . . .	34

<b>4</b>	<b>Simulation</b>	<b>35</b>
4.1	Introduction to Network Simulator (ns-2)	35
4.2	The iSCSI simulation classes	37
4.2.1	The iSCSI initiator and target classes	37
4.2.2	The iSCSI session and connection classes	41
4.2.3	iSCSI support classes	41
4.2.4	Timer Handler classes	42
4.3	Simulation Scenarios	43
4.3.1	Simulation of multiple commands	43
4.3.2	Simulation of WRITE command	44
4.3.3	Illustration of the WRITE command simulation.	45
4.4	Simulation of Error Recovery	46
4.4.1	Error Recovery Algorithm	46
4.4.2	Simulation of Error Recovery for the READ command	47
4.5	Simulation of Message boundaries within TCP byte stream.	49
4.6	Simulation of iSCSI over UDP	50
4.7	Traffic Generation	51
4.7.1	Probability Distributions	51
4.7.2	Trace Files	52
4.8	Incorporating Errors in iSCSI Simulation	54
4.9	Verification of the Simulation	54
4.10	The protocol features not simulated	56
4.11	Summary	56
<b>5</b>	<b>Performance Evaluation</b>	<b>58</b>
5.1	Performance Metrics	58
5.1.1	Response Time	58

5.1.2	Throughput and Utilization . . . . .	61
5.2	Command window size . . . . .	64
5.3	Bit Error Rate and Throughput . . . . .	66
5.4	iSCSI over UDP . . . . .	66
5.5	Summary . . . . .	68
<b>6</b>	<b>CRC Errors and iSCSI</b>	<b>70</b>
6.1	Introduction . . . . .	70
6.1.1	CRCs . . . . .	72
6.2	Aliased Packets . . . . .	74
6.3	Aliased Packet Probabilities . . . . .	76
6.4	TCP Error Control . . . . .	78
6.5	CRC Simulations . . . . .	81
6.5.1	iSCSI Error Control . . . . .	81
6.6	Experimental Results . . . . .	82
6.7	Discussion and Summary . . . . .	83
<b>7</b>	<b>Summary and Future Work</b>	<b>84</b>
7.1	Summary . . . . .	84
7.2	Future Work . . . . .	85
	<b>Glossary</b>	<b>87</b>
	<b>Bibliography</b>	<b>93</b>

# List of Figures

1.1	An iSCSI storage network. . . . .	13
2.1	iSCSI protocol layering model. . . . .	17
2.2	Login/Text Operational Keys . . . . .	20
2.3	48-Byte Basic Header Segment. . . . .	24
2.4	SCSI Command PDU. . . . .	25
4.1	A basic ns-2 script. . . . .	36
4.2	The class hierarchy of the iSCSI simulation classes. . . . .	38
4.3	A simulation script to create the iSCSI initiator and the iSCSI target simulated objects. . . . .	40
4.4	A sample of C++ code to bind C++ class variables to OTcl variables. . . . .	40
4.5	The error recovery algorithm used in the simulation of error recovery of missing PDUs. . . . .	48
4.6	Error Recovery Processing of Data-In PDUs for a READ Command. . . . .	49
4.7	A simulation script to create Marker PDU Aligned (MPA) objects . . . . .	50
4.8	Cumulative distribution function (CDF) of the command sizes. . . . .	52
4.9	Records of a trace file . . . . .	53
5.1	Response Time Definition . . . . .	59
5.2	An experimental setup for the iSCSI initiator and the iSCSI target. . . . .	61
5.3	Response time vs Expected Data Transfer Length. . . . .	62

5.4	Utilization for different buffer sizes. . . . .	62
5.5	Utilization Vs Expected Data Transfer Length. . . . .	63
5.6	A set of different input commands with the same total number of bytes. . .	64
5.7	Throughput for varying number of commands with constant data payload. .	65
5.8	Throughput and Command Window Size. . . . .	66
5.9	Throughput and Bit Error Rate (BER). . . . .	67
5.10	Response times for TCP and UDP for different values of EDTL. . . . .	68
5.11	Response times for TCP and UDP where the TransferContext timeout value for UDP is 0.1 seconds and the TransferContext timeout value for TCP is 1 second. . . . .	69
6.1	Typical Internet scenario showing various nodes and connectivities. . . . .	71
6.2	An LFSR generating a 3 bit CRC . . . . .	73
6.3	An alternative CRC implementation for the same function as shown in Fig- ure 6.1.1 . . . . .	74
6.4	State space trajectory of a detected packet in error and an aliased packet. . .	75
6.5	Signature for an input data stream. . . . .	75
6.6	Markov model for a 3 bit CRC . . . . .	76
6.7	Probability of packets aliasing . . . . .	78
6.8	Aliasing transient from 20, 16 and 8 bit CRCs . . . . .	79
6.9	Probability of TCP checksum aliasing . . . . .	80



# Acknowledgement

This work could not have been accomplished without the ardent and impetuous support from my adviser, Bob McLeod, who was always available for any assistance, whose critiques were very valuable, whose office door was never closed (until 4:30pm, even in his absence!). I will always remember our coffee break meetings at Tim Horton's and Starbucks!

I am also thankful to committee members Professors Ekram Hossain, Peter Graham, and Micaela Serra. Professor Hossain was a source of encouragement and gave valuable comments regarding performance evaluation based on TCP and UDP. I am obliged to Professor Graham for his untiring guidance in compiling this dissertation. He is also my role model for being an excellent teacher. Professor Serra, who is from the Computer Science Department at the University of Victoria, critically reviewed the dissertation and assisted in formulating future research directions.

I am fortunate to be blessed with a wonderful family. My mother, Zubaida Begum, and my father, Talib Hussain Awan, were very keen for my studies, from kindergarten to this dissertation. My sister, Mohni Awan, and my brother, Saqib Hussain Awan, were more concerned about my dissertation than myself! I am extremely grateful to my sweet wife, Sadia Dar, for her endless support. I can't wait anymore to see her refreshing smiles on the completion of this dissertation. I love to thank my little son, Mohammed Omar Awan, for his rewarding company. He also spent several evenings in my lab and drew innovative diagrams on the white board, when my wife was busy in evening shifts of her family medicine residency. I am extremely thankful to my family members for their patience and cooperation.

Working with other students, such as Imran and Dong, was a delightful experience. Dong was extremely helpful in this research. My other friends such as Hazem, Farook,

Alieu, Ashraf, Adel, and Faraj will always be remembered.

The technical and administration staff members were very cooperative. Guy Jonatschick, the UNIX guru, was always willing to install or update software packages and tools. He did a superb job in blocking “spam” emails. Jefferey Anderson and Andora Jackson were “industrial liaisons” for our lab. Marcia Labiuk and Karin Kroeker facilitated the research by sharing the burden of printing, filling the forms, and reminding of deadlines.

I am thankful to the Electrical and Computer Engineering Department, the Internet Innovation Center, and the University of Manitoba for the research facilities and the valuable financial support.

# Chapter 1

## Introduction

The storage industry has shown remarkable performance by providing high end storage devices at significantly lower prices; the storage cost of terabytes (trillion bytes) and petabytes (quadrillion bytes) is within the budget range of medium size companies. Furthermore, the telecommunication industry with relentless technology breakthroughs is currently announcing the arrival of Gigabit Ethernet (GbE) and 10 Gigabit Ethernet (10GbE). Due to the recent advancement in the storage and telecommunication technologies, IP (Internet Protocol) Storage has become a feasible alternative to the traditional fibre channel (FC) storage area networks (SANs), for medium as well as high end enterprise applications. Although high performance computing has also shown significant improvements in the last few decades, the system data path seems to be the limiting factor for data intensive applications of high speed networks.

### 1.1 Storage Interface

Small computer system interface (SCSI) devices provide efficient enterprise storage systems for server applications such as web servers, database management systems, and other on-line processing applications. SCSI and Serial Advanced Technology Attachment (SATA) interfaces for storage devices are described in the following subsections.

### **1.1.1 Small Computer Systems Interface (SCSI)**

Small Computer Systems Interface (SCSI, pronounced as skuzzy) is an interface to request services from I/O devices such as hard drives, tape drives, compact disks (CDs), printers, scanners and interface cards. It is based on a client/server architecture where the device that initiates a request is called the *initiator* and the one that receives the request is called the *target*. For example, when reading a file from a SCSI hard disk, the SCSI interface card (the initiator) requests data from the SCSI hard disk (the target), and the hard disk (the target) responds to the request by sending the data. SCSI devices can act as initiators and as well as targets.

### **1.1.2 Advanced Technology Attachment (ATA)**

SCSI disks are often compared with Advanced Technology Attachment (ATA) disks. The two devices differ not only in their interfaces but also in their mechanics, materials, electronics and firmware. Although SCSI disks are more expensive than ATAs, the SCSI disks provide better reliability, faster random access, and higher connectivity than the ATA disks. The ATA disks are designed for personal storage but SCSI disks are more suitable for enterprise applications [ADR03].

The Serial ATA (SATA) is an improved ATA interface that is more reliable and efficient than ATA. The SATA devices can be the most cost effective storage devices for the backup storage of enterprise applications [LoB02].

## **1.2 Storage Systems**

A storage system with a group of several hard disks is more economical than manufacturing one huge hard disk. Several storage systems are discussed in the following sub-sections.

### **1.2.1 JBOD**

Just a Bunch Of Disks (JBOD) is an enclosure with multiple disk drives installed on a common backplane. The disks are addressed individually because there is no front-end logic to manage the distribution of data over the disks. Since the JBOD enclosure consolidates multiple disks sharing power supplies and fans, JBOD is more cost effective than a set of individual hard disks.

### **1.2.2 RAID**

Redundant Array of Independent Disks (RAID) is an intelligent storage array with specific methods to distribute data on multiple disks. Embedded in the enclosure, implemented in either hardware or in software, an intelligent controller performs RAID functions and stands between the external interface to the host and the internal configuration of disks. RAID provides an economical storage solution for low or medium size enterprise industries.

### **1.2.3 Tape Subsystems**

Although tape subsystems are quite slow as compared to disks, the tape subsystems provide high capacities at very low cost. They use block SCSI I/O to transfer large volumes of data. The tape subsystems are used for archives and periodic backup.

## **1.3 Network Storage**

A direct attached storage (DAS) device is a storage device that is directly attached to the host system. Although DAS systems are less expensive, the maintenance cost of DAS systems is relatively higher. Furthermore, the storage allocation is quite challenging in DAS systems. On the other hand, network storage makes the storage devices independent of any host, resulting in highly efficient storage maintenance and allocation strategies. The

two common types of network storage are described in the following sub-sections.

### **1.3.1 Network Attached Storage (NAS)**

NAS systems use file-oriented delivery protocols such as Network File System (NFS) and Common Internet File System (CIFS). Although NAS also has a block component where blocks are addressed on a per-file basis, the block access methods are hidden in the NAS enclosure. A NAS device is a server of files and directories, providing the sharing of storage resources over a common network.

### **1.3.2 Storage Area Networks (SAN)**

Although NAS devices serve assembled files, SANs serve blocks of data. The block-oriented service has the advantage of using more efficient serial SCSI transport that requires minimal central processing unit (CPU) resources for protocol processing. Traditionally, Fibre Channel (FC) was the only mechanism to implement SANs; however, due to the arrival of Gigabit Ethernet (GbE) and 10 Gigabit Ethernet (10GbE), IP networks are becoming cost effective alternatives to the traditional FC-SANs.

## **1.4 The iSCSI Protocol**

The RFC 3720 [SSCZ04] describes the iSCSI protocol that is an Internet standards track protocol to send SCSI commands over the Transmission Control Protocol (TCP). Thus, iSCSI can be used to interconnect SCSI disks, disk arrays, and also SANs through TCP/IP. Figure 1.1 illustrates a storage network where all servers and storage resources support an Ethernet interface (or Gigabit Ethernet) and an iSCSI protocol stack. The iSCSI server at node *A* uses the Internet to connect to iSCSI disks at node *B* and the iSCSI tape library at node *D*. Hence, iSCSI eliminates the traditional limitation of shorter distances of the SCSI cable.

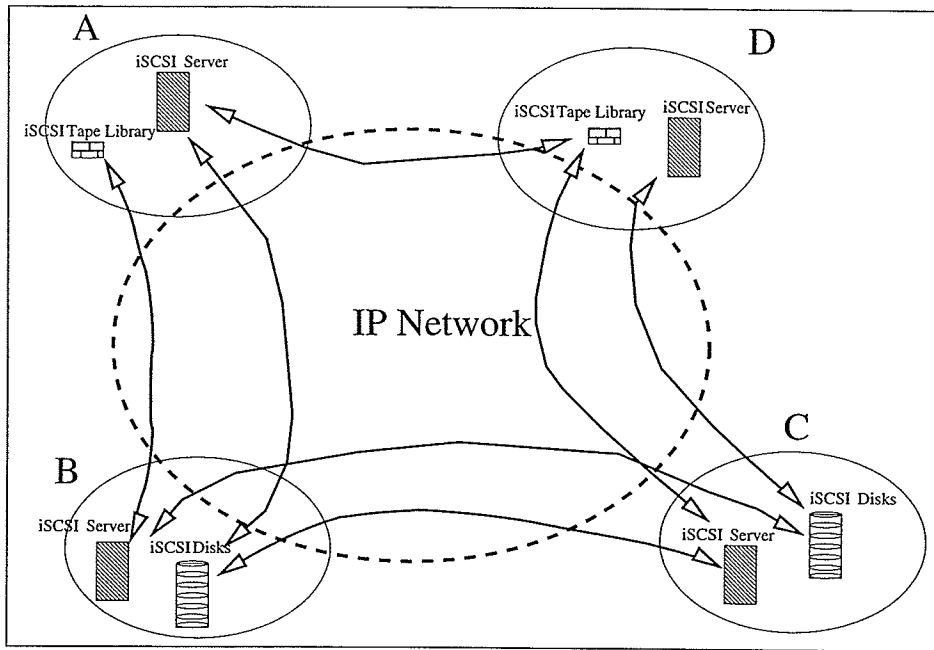


Figure 1.1: An iSCSI storage network.

## 1.5 Motivation

The iSCSI protocol provides the IP storage that is significantly less expensive than Fibre Channel SANs. IP storage synthesizes two mature technologies: SCSI and IP networks. Thus, the iSCSI protocol has the potential to meet the demands of emerging storage applications.

Since iSCSI is designed to operate over the TCP protocol, which can be the limiting factor for the end-to-end system performance for data intensive applications, one of the goals of the research is to help extend iSCSI to individual hosts as an efficient end-to-end process. This includes identifying redundancies in layered protocols and inefficiencies in terms of error and flow control. The iSCSI performance analysis for wide area networks (WANs) and local area networks (LANs) can assist in protocol tuning required for efficient operation.

Although there are a few research projects doing iSCSI performance evaluation us-

ing kernel level tools, it appears to be cumbersome to correlate the iSCSI protocol with telecommunication protocols using the current kernel level tools. Since the network simulator (ns-2) provides simulation of almost all the Internet protocols, including variations of the TCP protocol, the simulation and modeling of the iSCSI protocol was also a desirable goal. The simulation of iSCSI will facilitate future research involving iSCSI and other Internet protocols.

As such, the dissertation objectives are as follows:

1. Simulate the iSCSI protocol in an environment where it is more effective to study the correlation of the Internet protocols and the iSCSI protocol. Since the Network Simulator ns-2 is widely used for the Internet protocols research, the simulation of the iSCSI protocol in ns-2 will provide opportunities for the Internet protocols tuning for IP storage.
2. Evaluate the performance of the iSCSI protocol for various network conditions. The iSCSI performance in local area networks is compared to wide area networks. The effect of different network parameters, such as effective maximum segment size (EMSS), maximum transmission unit (MTU), and delay bandwidth product, over the iSCSI performance metrics, such as response time, command throughput, and data throughput is analyzed.
3. Tune the transport protocols parameters, such as those of TCP and UDP. The efficient implementation and tuning of TCP can be a significant factor in the end systems performance enhancement.
4. Analyze the performance of iSCSI error control mechanism when iSCSI is deployed over unreliable transport protocols such as UDP. If iSCSI is used to connect SANs in a local environment, UDP can provide more efficient transfer of SCSI commands because of reduced errors of the LAN environment. iSCSI can run effectively without



requiring all of the overhead imposed by the TCP reliability mechanisms.

5. Estimate packet aliasing using techniques such as Markov model analysis, simulation, and experimental measurements.

## 1.6 Summary

This chapter provides the motivation for the performance evaluation of iSCSI for IP storage and transport protocols. The remaining dissertation is organized as follows:

- Chapter 2 provides the basic understanding of iSCSI to develop a model for the simulation.
- Chapter 3 briefly discusses related research for IP Storage.
- Chapter 4 describes the simulation of iSCSI protocol within ns-2.
- Chapter 5 provides the system performance evaluation using the iSCSI layer in ns-2.
- Chapter 6 provides an analysis of potential error control redundancies when layering iSCSI over TCP.
- Chapter 7 concludes the dissertation and outlines the directions of future research.

# Chapter 2

## The iSCSI Protocol

This chapter briefly describes the iSCSI protocol to facilitate the simulation and performance evaluation of the protocol.

### 2.1 Protocol Stack

The iSCSI protocol is based on a client server architecture. The SCSI initiator builds and sends the blocks, which are related to the SCSI command, to the iSCSI initiator, which encapsulates the SCSI blocks in iSCSI Protocol Data Units (PDUs) and sends the iSCSI PDUs to the iSCSI target by using TCP. There is also an optional layer of data synchronization and data steering mechanism, ensuring in-order receipt of iSCSI PDUs. The data synchronization framing mechanism is needed to preserve the boundaries of iSCSI PDUs in the TCP byte stream. The iSCSI specification allows a lower functional level layer on top of Internet Protocol (IP), to provide services such as IPsec data encryption. Figure 2.1 shows the protocols involved to send/receive the SCSI commands over the Internet.

### 2.2 iSCSI Naming and Discovery

The RFC-3721, iSCSI Naming and Discovery [BHH<sup>+</sup>04], discusses the naming and discovery of iSCSI storage resources. An iSCSI node can be either an initiator, or a target, or both.

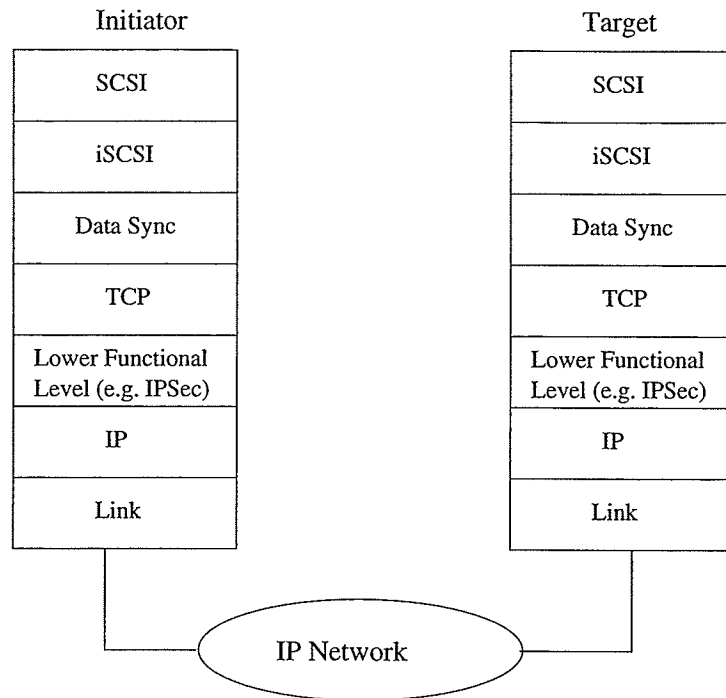


Figure 2.1: iSCSI protocol layering model.

### 2.2.1 Name, Address, and Alias

An iSCSI name is a unique name for an iSCSI node. It is also the SCSI device name of the iSCSI device. The iSCSI name is never modified because it is associated with the iSCSI node instead of the network adapter card. The iSCSI name refers to a logical software entity, which is not tied to a port or hardware that can be changed. For example, the iSCSI qualified name from an equipment vendor is *iqn.2001-04.com.acme:disk-arrays-sn-a8675309*, where *iqn* is the type, *2001-04* is the date, *com.acme* is the organization naming authority, and *disk-arrays-sn-a8675309* is the subgroup naming authority.

An iSCSI address specifies a single path to an iSCSI node. The iSCSI address format is  $\langle \text{domain-name} \rangle [:\langle \text{port} \rangle]$ , where  $\langle \text{domain-name} \rangle$  can be an IPv4 address, an IPv6 address, or a fully qualified domain name.

An iSCSI alias is a string that can be used as a descriptive name for an initiator, or

a target. It cannot be used for identification during login. Furthermore, the iSCSI alias does not follow the uniqueness or other requirements of the iSCSI name. For instance, *LocalDisk* can be an alias for *iqn.2000 – 04.com.acme : sn.5551212.target.489*.

### **2.2.2 iSCSI Discovery Mechanisms**

The iSCSI protocol supports discovery mechanisms, such as “Static Configuration”, “SendTargets configuration”, and “Zero-Configuration”.

Static Configuration assumes that the IP address, the TCP port number and the iSCSI target name are already available to the initiator. The initiator does not need to perform any discovery.

SendTargets Configuration assumes that the IP address and TCP port information are available to the initiator. The initiator issues a SendTargets text command to query information about the iSCSI targets available at the particular network entity.

Zero-Configuration assumes that the initiator does not have any information about the target. The initiator can either multicast discovery messages directly to the targets, or it can send discovery messages to storage name servers.

## **2.3 Login Phase and Full Feature Phase**

The login phase is the first phase to establish a connection between the initiator and the target. The login sequence is used to negotiate and exchange parameters between the initiator and the target, and may invoke a security routine to authenticate allowable connectivity. If successful, the target will issue a login accept to the initiator; otherwise, the login is rejected and the connection is broken.

The iSCSI login uses text fields to negotiate allowable parameters between the initiator and the target. These fields are associated with keys, which are followed by their corresponding values. The text fields are also used to exchange names and aliases of the target and initiator, as well as negotiated parameters such as security protocol, maximum data

payload size, unsolicited data support, the allowable length of unsolicited data, and time-out values. For instance, if the initiator and target have negotiated the key *InitialR2T* to *No* during login, unsolicited data can be sent to the target.

An initiator logging on to a target would include its iSCSI name and an initiator session ID (ISID), the combination of which would be unique within its host network entity. A target, responding to the login request, would generate a unique target session ID (TSID). A single ISID/TSID session pair may have multiple TCP connections, where the maximum number of connections is negotiated during the login phase. When the login phase is completed, the iSCSI session enters into the full feature phase, where the initiator sends SCSI commands and data to the various Logical Units (LUs) on the target. The text fields (or keys) that are negotiated between the initiator and the target can be divided into following categories:

- *LO - Leading Only*: Keys that can only be carried on the leading connection and cannot be changed after the leading connection login.
- *IO - Initialize only*: Keys that can be used only during login.
- *ALL* : Keys that can be used in both the login phase and the full feature phase.
- *FFPO - Full Feature Phase only* : Keys that can only be used during full feature phase.
- *Declarative* : Keys that do not require an answer.

Figure 2.2 shows a few keys and the corresponding values for category, scope, and value-range. The key scope is indicated as either session-wide (SW) or connection-only (CO). The combined effect of *ImmediateData* and *InitialR2T* is described as follows:

- If both *ImmediateData* and *InitialR2T* are negotiated as *Yes*, the initiator sends immediate data with the command but no unsolicited data-out PDUs are transferred.

Key	Category	Senders	Scope	Value range	Default	Result
MaxConnections	LO	Initiator/Target	SW	<1-to-65535>	1	Min.
TargetName	IO by initiator and FFPO by Target	Initiator/Target	SW	<iSCSI-name>		
InitiatorName	IO, Declarative, Any-Stage	Initiator	SW	<iSCSI-name>		
TargetAlias	ALL, Declarative, Any-Stage	Target	SW	<local-name>		
InitiatorAlias	All, Declarative, Any-Stage	Initiator	SW	<local-name>		
TargetAddress	All, Declarative, Any-Stage	Target	SW	domain name[:port] [,portal-group-tag]		
TargetPortalGroupTag	IO by target, All, Declarative, Any-Stage	Target	SW	<16-bit-binary>		
InitialR2T	LO	Initiator/Target	SW	<boolean-value>	Yes	OR
ImmediateData	LO	Initiator/Target	SW	<boolean-value>	Yes	AND
MaxRecvDataSegmentLength	All, Declarative	Initiator/Target	CO	<512-to-(2 <sup>24</sup> -1)>	8192	
MaxBurstLength	LO	Initiator/Target	SW	<512-to-(2 <sup>24</sup> -1)>	256 KB	Min.
FirstBurstLength	LO	Initiator/Target	SW	<512-to-(2 <sup>24</sup> -1)>	256 KB	Min.

Figure 2.2: Login/Text Operational Keys

- If *ImmediateData* is *No* and *InitialR2T* is *Yes*, the initiator sends neither immediate nor unsolicited data.
- If both *ImmediateData* and *InitialR2T* are *No*, the initiator sends unsolicited data-out PDUs but no immediate data is sent with the command.
- If *ImmediateData* is *Yes* and *InitialR2T* is *No*, the initiator sends immediate data as well as unsolicited data-out PDUs.

### 2.3.1 Connection Allegiance

If a session contains multiple TCP connections, individual command/response pairs must flow over the same TCP connection. This connection constraint ensures that specific read or write commands are fulfilled without the additional overhead of monitoring multiple connections to observe the request completion and to maintain the order of arrival. An iSCSI write, for example, would be performed over a single connection until all data was transmitted. Unrelated transactions, however, could simultaneously be issued on different connections during the same session.

## 2.4 Data Transfer

An initiator can send SCSI data as either solicited or unsolicited. The unsolicited data can be sent as a part of an iSCSI command PDU (immediate data), and/or in separate iSCSI data PDUs that are sent immediately after the command PDU. Although unsolicited data is sent without any response from the target, the solicited data can be sent only after receiving Ready-to-Transfer (R2T) PDUs from the target.

The *FirstBurstLength* field specifies the maximum number of bytes that can be sent as unsolicited data; however, the *MaxBurstLength* field specifies the maximum number of bytes that can be sent in a sequence of solicited data. The values of *FirstBurstLength* and *MaxBurstLength* are negotiated during the login phase. For each sequence of data transfer, the initiator must send a number of bytes equal to either *FirstBurstLength* (for the unsolicited data) or *MaxBurstLength* (for the solicited data); unless it is the last sequence.

Command and unsolicited data PDUs may be interleaved on a single connection but ordering requirements must be maintained. For example, the command  $N + 1$  may be sent before unsolicited data PDUs for the command  $N$  but unsolicited data PDUs of the command  $N$  must precede unsolicited data PDUs of the command  $N + 1$ .

## 2.5 iSCSI Numbering

iSCSI uses several numbering schemes for identification of different entities such as sessions, connections, command PDUs, data PDUs and other iSCSI PDUs. The numbering is used in the error recovery mechanisms of iSCSI.

A session is formed by the group of TCP connections that link an initiator and a target. A session is identified by a session ID that comprises an initiator session ID (*ISID*) and a target session ID (*TSID*).

## 2.5.1 Command Numbering

Command Numbering is session-wide and is used to ensure ordered command delivery over multiple connections. It can also be used for command flow control over a session. The command number is carried by the iSCSI PDU as a *CmdSN* field, which is incremented by 1 for each additional command except for commands marked for immediate delivery. Commands meant for immediate delivery are marked with an immediate delivery flag and they carry the current *CmdSN*. If immediate delivery is used with task management commands, the commands are allowed to reach the target before the tasks on which they are supposed to act.

The iSCSI PDUs sent by the target contain fields *ExpCmdSN* and *MaxCmdSN*. The *ExpCmdSN* field specifies the next command expected by the target; therefore, the target acknowledges all the commands up to, but not including, this number. The *MaxCmdSN* field specifies the maximum command number that can be sent by the initiator; thus, the queuing capacity of the receiving iSCSI layer is  $MaxCmdSN - ExpCmdSN + 1$ . The expression  $MaxCmdSN - ExpCmdSN + 1$  is referred as the command window for the initiator. If the command window is greater than zero, the initiator is allowed to send the commands to the target; otherwise, the initiator must wait for the increase in the command window size.

## 2.5.2 Status Numbering

The *StatSN* field is a status sequence number, which is a field in iSCSI PDUs sent by the target. It is used to enumerate the responses sent by the target. The *ExpStatSN* field is an expected status sequence number, which is used by the initiator to acknowledge the responses received from the target. The status numbering starts with the login response of the first login request of the connection. The login response includes an initial value for the status numbering. A large absolute difference between *StatSN* and *ExpStatSN* may



indicate a failed connection.

### 2.5.3 Data Numbering

The *DataSN* field is used for data sequencing. It starts with 0 and is incremented by 1 for each additional data PDU. There is only one data sequence for the READ command; however, there can be multiple data sequences for the WRITE command. At most one data sequence is allowed for the unsolicited data, although several sequences are possible for solicited data because the initiator sends one unsolicited data sequence for each R2T received from the target. The R2Ts are identified using *R2TSN* field defined in the R2T PDUs sent by the target.

## 2.6 iSCSI PDUs

All iSCSI PDUs have one or more header segments and, optionally, a data segment. The Basic Header Segment (BHS) is the first segment in all iSCSI PDUs. It may be followed by Additional Header Segments (AHSs), a Header-Digest, a Data Segment, and/or a Data Digest.

### 2.6.1 Basic Header Segment (BHS)

The BHS header is 48 bytes long. The *Opcode* and *DataSegmentLength* fields appear in every iSCSI PDU header. If the *InitiatorTaskTag* and the *LogicalUnitNumber* are used, they always appear in the same location of the header.

#### Fields of Basic Header Segment (BHS)

Figure 2.3 shows the format of the BHS header. The *I* bit identifies PDUs that are marked for the immediate delivery. If the *I* bit is set to 1 then the delivery is immediate; otherwise it is queued. The *Opcode* field specifies the type of iSCSI PDU. There are two types of opcodes: initiator opcodes and target opcodes. The initiator opcodes are in request PDUs

Byte	0				1				2				3											
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
00-03	I			Opcode				F	Opcode-specific fields															
04-07	Total AHS Length								Data Segment Length															
08-11	LUN or Opcode-specific fields																							
12-15																								
16-19	Initiator Task Tag																							
20-47	LUN or Opcode-specific fields																							

Figure 2.3: 48-Byte Basic Header Segment.

from the initiator; whereas, the target opcodes are in response PDUs from the target. The *F* bit identifies the end of the sequence. If the *F* bit is 1, it indicates that the PDU is the final PDU of the sequence.

The *TotalAHSLength* field specifies the total length of all the AHS header segments in units of 4-byte words including padding, if any. The *DataSegmentLength* field specifies the data payload in bytes, padding is excluded. The *InitiatorTaskTag* field identifies each iSCSI task issued by the initiator. See RFC 3720 [SSCZ04] for the remaining fields of the BHS header.

## 2.6.2 SCSI Command PDU

The initiator sends the SCSI command PDU to initiate the command at the target. The format of a SCSI command PDU is shown in Figure 2.4.

### Fields specific to the SCSI command PDU

The *R* bit is set to 1 when the command is expected to input (read) data. If *R* is set to 0, no data will be read from the target. The *W* bit is set to 1 when the command is expected to output data.

The *ExpectedDataTransferLength (EDTL)* field, bytes 20 – 23, specifies the total amount of data to be transferred for the command. For unidirectional (either  $W = 1$  or  $R = 1$ ) operations, the *EDTL* field contains the number of bytes of data for this SCSI

Byte	0								1								2								3							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
00-03	I		0x01						F	R	W	0	0	ATTR				Reserved														
04-07	TotalAHSLength								DataSegmentLength																							
08-11	LUN - Logical Unit Number																															
12-15																																
16-19	Initiator Task Tag																															
20-23	Expected Data Transfer Length																															
24-27	CmdSN																															
28-31	ExpStatSN																															
32-47	SCSI Command Descriptor Block (CDB)																															
...	AHS (Optional)																															
...	Header Digest (Optional)																															
...	Data Segment, Command Data (Optional)																															
...	Data Digest (Optional)																															

Figure 2.4: SCSI Command PDU.

operation. For bidirectional operations (both R and W are set to 1), this field contains the number of data bytes for the write transfer. For bidirectional operations, an additional header segment must be present in the header sequence that indicates the Bidirectional Read Expected Data Transfer Length.

The *CmdSN* field, bytes 24 – 27, is a command sequence number, which is used to enable ordered delivery of commands across multiple connections in a single session. The *ExpStatSN* field, bytes 28 – 31, is the expected status sequence number, which acknowledges the command responses received on the connection. See RFC 3720 [SSCZ04] for the remaining fields of the iSCSI command PDU.

### 2.6.3 SCSI Response

The target sends the SCSI response PDU to the initiator, indicating the command completion.

## Fields specific to SCSI response PDU

The *Response* field contains the iSCSI service response code. The value 0 shows the command completion at the target, whereas the value 1 indicates the command failure. The *Status* field is used to report the SCSI status of the command. [SAM02] provides the complete list of *Status* codes and their definitions.

The initiator sends a SNACK PDU (Sequence Number Acknowledgement PDU) when the response PDU is received from the target. The *StatSN* field is a status sequence number that the iSCSI target generates per connection. It enables the initiator to acknowledge the reception status. The *StatSN* field is incremented by 1 for every response/status sent on a connection except for responses sent as a result of a retry or SNACK. In the case of response retransmission, the value of *StatSN* remains unchanged, the value is same as used in the original transmission, unless the connection has since been restarted.

The *ExpCmdSN* field stands for the expected command sequence number, which is returned to the initiator to acknowledge command reception. If *ExpCmdSN* equals to  $MaxCmdSN + 1$ , the target cannot accept new commands. The *MaxCmdSN* field specifies the maximum command sequence number, which the iSCSI target sends to the initiator to indicate the maximum value of *CmdSN* that can be sent by the initiator.

### 2.6.4 Data-Out/Data-In and R2T PDUs

The data-out/data-in PDUs are used to transfer data that is associated with the SCSI command. The data-out PDU is used for the write command and the data-in PDU is used for the read command.

The target uses R2T PDUs to request the sequences of data blocks to be delivered in the order that is convenient for the target. The initiator sends *MaxBurstLength* amount of data-out PDUs for each R2T received from the target, other than the last R2T, which specifies only the remaining bytes.

## **2.6.5 SNACK PDU**

The SNACK, which stands for sequence number acknowledgement, is used by the initiator to request retransmission of missing PDUs. SNACK PDU may be issued against missing data, response, or R2T PDUs.

Digest errors represent the corruption of the iSCSI PDU. There are two types of digest errors: 1) the data digest error, which means that the data payload is corrupted, and 2) the header digest error, which identifies the corruption of the header. If data digest error is detected then SNACK is sent for the corrupted/missing PDUs. A single SNACK PDU can be used for retransmission of several contiguous missing/corrupted PDUs.

## **2.6.6 Reject, Nop-Out, and Nop-In PDUs**

Format error means that an individual PDU has missing or inconsistent fields within the frame. When format errors occur, a reject iSCSI PDU is sent, which contains an offset indicator for the first bad byte detected in the PDU header. The reject PDUs can be ignored or saved for any future analysis of the implementation and/or network conditions.

The Nop-out/Nop-in PDUs are sent when there is no other iSCSI PDU to be sent. Thus Nop-out/Nop-in PDUs ensure the connectivity and the session establishment between the initiator and the target.

## **2.7 iSCSI Error Recovery**

One of the primary requirements for iSCSI error handling and recovery is the ability of both initiators and targets to buffer commands and responses until they are acknowledged. In a SCSI write, for example, the initiator should keep the transmitted data in its buffer until another R2T is received from the target, which indicates that the target has received the previous data and is ready for more. At minimum, the iSCSI end devices must be able to selectively rebuild the missing or corrupted PDU for retransmission.

The iSCSI error detection and recovery contains several levels of hierarchy. The lowest level is within command error recovery, where the missing or corrupted PDU is retransmitted. The next level is within connection recovery, where commands are restarted. The next level is within session recovery, where connections are rebuilt. Finally, a session recovery provides restarting the session through login.

### 2.7.1 Mechanisms for Error Recovery

The error recovery mechanisms of the initiator differ from the error recovery mechanisms of the target. The initiator error recovery mechanisms are described as follows:

1. *Nop-Out* is a no operation PDU sent from the initiator when there is no iSCSI PDU to be sent. The nop-out PDU probes sequence numbers from the target.
2. The *command retry* mechanism supports the retransmission of the command PDU.
3. The *SNACK* (Sequence Number Acknowledgement) mechanism requests the retransmission of missing status, data, and R2T PDUs.

The target mechanisms for error recovery are as follows:

1. *Nop-In* is the target's no operation PDU that is sent in response to nop-out PDU received from the initiator. The nop-in PDU is transmitted when there is no iSCSI PDU to be sent.
2. *Recovery R2T* PDU is used to request the retransmission of missing data-out PDUs. The *R2T* PDU's *DesiredDataTransferLength* field indicates the number of bytes to be transferred and the *BufferOffset* field indicates the address of the first missing byte. Since *R2T* PDU indicates only the starting address and the total bytes, the number of bytes to be retransmitted must be contiguous.

## 2.8 Message Synchronization and Steering

This section describes the mechanisms to identify boundaries of iSCSI PDUs in the TCP byte stream. iSCSI provides a mapping of the SCSI protocol onto TCP. The iSCSI headers contain a message length field, *DataSegmentLength*, which serves to indicate the end of the current message as well as the beginning of the next message. However, relying only on the *DataSegmentLength* field of the iSCSI header is not sufficient because it is possible to lose the TCP segment that contains the *DataSegmentLength* field. Since the missing TCP segment(s) must be received before any of the following segments can be steered to the correct SCSI buffers, temporary buffers are required to save the out-of-order segments, which can be copied to the SCSI buffers after the arrival of the missing headers.

### 2.8.1 Fixed Interval Markers

The Fixed Interval Markers (*FIM*) scheme is used for synchronization and/or data steering. The *FIM* scheme inserts markers at fixed intervals in the payload stream. Under normal conditions (no PDU loss, or out-of-order data reception), iSCSI data steering can be accomplished by using the identifying tag (Initiator Task Tag), the iSCSI header's data offset fields, and the TCP sequence number. Although the identifying tag associates the PDU with a SCSI buffer address, the data offset and TCP sequence number are used to determine the offset within the buffer. If the TCP segment containing the iSCSI PDU header is delayed or lost, markers are used to minimize the damage. Markers indicate the beginning of the next iSCSI PDU and enable continued processing when the iSCSI headers are dropped due to the iSCSI level data errors. Moreover, markers reduce the amount of data to be stored by the TCP/iSCSI layer because while waiting for the arrival of the late TCP segments, the markers may assist to find later iSCSI PDU headers that may steer the related data to the SCSI buffers.

## **2.8.2 Upper Layer Protocol Framing**

The Upper Layering Protocol (ULP) framing mechanism works as a “shim” between TCP and higher-level protocols [CT90]. If the record is less than or equal to the maximum transmission unit (MTU), the ULP mechanism preserves the higher-level protocol record boundaries. Framing-aware TCP implementations indicate the path maximum segment size (MSS) to the framing protocol. This size may change during the course of the connection due to the changes in the path MTU. The framing protocol must notify the ULP sender of the changes in the MSS. The framing protocol provides the current value of the path MSS to the ULP.

## **2.9 Summary**

This chapter provides sufficient background and understanding of the iSCSI protocol to develop a model for simulation and undertake performance evaluation studies. First, the iSCSI protocol stack, the naming and discovery mechanism, and the login and full feature phases were explained. Then, several iSCSI PDUs such as command, response, data in/out, SNACK, reject, and Nop in/out were briefly described. Finally, the iSCSI error recovery and message synchronization and steering mechanisms were discussed. Thus, the chapter provides the protocol basis for further simulation and performance evaluation.



# Chapter 3

## Related Research

This chapter briefly discusses research related to storage over IP networks.

### 3.1 Fibre Channel

Fibre Channel (FC) Storage Area Networks (SANs) are commonly used for high-end enterprise applications. Although FC-SANs use the SCSI interface, the fibre channel protocol stack is quite different from the TCP/IP stack. The FC protocol stack contains FC specific protocols for physical, data link, network and transport layers.

The Internet Engineering Task Force (IETF) has proposed two standards to connect existing FC-SANs using TCP/IP based networks. Fibre Channel over TCP/IP (FCIP) [RRW02] connects FC-SANs by a virtual FC link, where encapsulated FC frames are sent over the TCP/IP connection. The other standard, the Internet Fibre Channel Protocol (iFCP) [MTJM02] specifies an architecture and gateway-to-gateway protocol for the fibre channel functionality implementation over an IP network. The lower-layer FC transport is replaced with TCP/IP and Gigabit Ethernet. The FC session is terminated at the local gateway/switch and converted to a TCP/IP session via iFCP.

Thus, FCIP and iFCP allow the connectivity of existing FC-SANs using TCP/IP based networks. The former creates a virtual FC link to send encapsulated FC frames, whereas the latter replaces the lower-layer FC transport with TCP/IP and Gigabit Ethernet.

Voruganti and Sarkar [VS01] compared iSCSI with Fibre Channel (FC) and Infiniband Architecture (IBA). They concluded that for efficient iSCSI performance at gigabit wire speeds, support for framing mechanism is necessary for iSCSI network cards, and support for zero-copy, framing and interrupt coalescing is necessary if iSCSI is implemented using commodity gigabit network cards. When SANs are deployed across WANs (Wide Area Networks), iSCSI is more suitable than FC and IBA protocols because of its flow control, and dynamic time-out calculation mechanism. The study also suggests the support of jumbo frame sizes in all three protocols because storage applications usually perform large block size transfers (4K and 8K).

### **3.2 Performance Analysis of iSCSI**

Several studies have been done to compare the performance of iSCSI and Fibre Channel. Lu and Du [LD03] compared the iSCSI with the Fibre Channel for block I/O and file I/O over various configurations and concluded that iSCSI performs better in local environments. Aiken et al. [AGPW03] concludes that iSCSI protocol must be tuned to utilize the network resources effectively.

Storage Performance Evaluation Kernel Module (SPEK) [HY03] is a block level benchmarking performance evaluation tool for SCSI storage devices. The I/O requests are sent directly to the SCSI layer, thus bypassing the file system cache and the buffer cache. SPEK is reported to be more accurate and efficient than other file system performance evaluation tools, which usually work in user space. The tool also provides a java graphical user interface for parameter input. Performance metrics such as response time and throughput (I/Os per second, or bytes/second) can be analyzed. Both the average and the runtime values of the performance metrics can be obtained. Thus, SPEK appears to be an accurate and efficient benchmarking performance evaluation tool for SCSI devices.

In addition to SPEK, there are several storage systems benchmarking tools, such as

PostMark [Kat01], Iozone [CN], Iometer [DSE<sup>+</sup>], Bonnie++ [Cok]. PostMark is a file-system benchmark tool, which addresses the performance evaluation of traffic loads with smaller files, such as electronic mail and web-based commerce. Another file-system benchmark tool is Iozone, designed for the performance evaluation of various file operations. Iometer is an I/O subsystem measurement and characterization tool for single and clustered systems. Bonnie++ is a disk I/O benchmark tool, which tests for file operations such as `creat()`, `stat()`, and `unlink()`.

The above studies were kernel level studies to measure and benchmark the iSCSI performance. It is not easy to change the network parameters and study the correlation of iSCSI protocol with other protocols. Thus, in this study iSCSI is simulated in ns-2 to facilitate the performance evaluation of iSCSI in IP networks.

### **3.2.1 Efficient implementation of iSCSI**

Current TCP implementations have the problem of making intermediate copies of the data because data from the user buffers is copied to the buffers in the transmit/receive queues. Since this problem is quite significant in networks with larger delay-bandwidth products, Direct Data placement (DDP) [SPRC04] can be used to avoid extra copying overhead. The iSCSI PDU header contains data placement information, such as Initiator task tag (ITT), which DDP can use to place iSCSI PDUs directly into the main memory. Remote Direct Memory Access (RDMA) is a copy avoidance technique that uses network interface cards (NICs) to steer incoming data directly into user-specified buffers [CGY01].

The iSER protocol [CSE<sup>+</sup>03], iSCSI Extensions for RDMA, is proposed to provide Remote Direct Memory Access (RDMA) mechanism for iSCSI. Since RDMA requires the message boundaries to be known, there is a need for an additional layer to insert message boundaries because TCP is a stream service with no message boundaries. The Marker PDU Aligned (MPA) framing layer inserts record boundaries by creating Framed Protocol Data Units (FPDUs). The FPDUs are delivered by the TCP agent to the MPA receiver. The MPA

receiver extracts the iSCSI PDUs and delivers them to the DDP receiver. The DDP receiver places the iSCSI PDUs in the main memory, with no intermediate copying.

He and Yang [HY02] introduced a storage architecture that couples reliable high-speed data caching with low over-head conversion between SCSI and IP protocols. The storage devices were used to cache the data while the intelligent processing units carried out caching algorithm, protocol conversion, and self-management functions.

### **3.3 Summary**

Since the deployment and maintenance costs of IP storage are significantly less as compared to the costs associated with the traditional fibre channel SANs, there was a huge industrial pressure for the standardization of the iSCSI protocol. However, FCIP and iFCP have also been proposed to leverage existing FC storage systems.

Several IP storage performance evaluation studies have been conducted to verify the proposition that IP storage is more feasible rather than FC SANs. Consequently, performance analysis and benchmarking of iSCSI was performed for various storage systems. Since numerous network conditions and traffic loads are required to study the correlation of iSCSI protocol with other protocols, the simulation of iSCSI is seen to be highly desirable.

Our work is more focused on the simulation of iSCSI within ns-2 to facilitate the investigation of the tuning of the iSCSI protocol for use with available network resources. Specifically, we are more interested in the performance of iSCSI within a telecommunication environment than with iSCSI storage mechanisms.

# Chapter 4

## Simulation

This chapter describes an implementation of the iSCSI simulation in the ns-2 simulator.

### 4.1 Introduction to Network Simulator (ns-2)

Since simulation based performance evaluation of iSCSI requires a simulation of Internet protocols such as Transmission Control Protocol (TCP), Internet Protocol (IP), and User Datagram Protocol (UDP), a simulation of iSCSI is implemented in the simulator ns-2 [FV04]. ns-2 is an open-source event-driven simulator extensively used for TCP/IP research.

The simulator is written in an object oriented architecture where C++ is used for frequently executed code and OTcl interpreter is used where the code is executed fewer times. The OTcl also provides the framework for the runtime variation of input parameters to facilitate the execution of numerous simulation scenarios. Since a C++ class in ns-2 can be associated with the corresponding OTcl class, the C++ class variables can be bound to the variables of the corresponding OTcl class. As a consequence, the values of C++ class variables can be dynamically modified by the modification of the corresponding OTcl class variables.

Figure 4.1 illustrates a partial OTcl script of ns-2 to simulate data transfer between two nodes. First, an object of the simulator is created as shown in line 1. The simulator

```

Line 01  set ns [new Simulator]
Line 02  #----- Create Nodes
Line 03  set node1 [$ns node]
Line 04  set router1 [$ns node]
Line 05  set router2 [$ns node]
Line 06  set node2 [$ns node]
Line 07  #----- create links
Line 08  $ns duplex-link $node1 $router1 100Mb 1ms DropTail
Line 09  $ns duplex-link $router1 $router2 10Mb 10ms RED
Line 10  $ns queue-limit $router1 $router2 100
Line 11  $ns duplex-link $router2 $node2 100Mb 1ms DropTail
Line 12  #----- create transport agents
Line 13  set tcp1 [new Agent/TCP/FullTcp]
Line 14  set tcp2 [new Agent/TCP/FullTcp]
Line 15  $ns attach-agent $node1 $tcp1
Line 16  $ns attach-agent $node2 $tcp2
Line 17  $ns connect $tcp1 $tcp2
Line 18  #----- setup TCP connection}
Line 19  $tcp2 listen
Line 20  $tcp1 set window_ 100
Line 21  #----- create FTP application}
Line 22  set ftp [new Application/FTP]
Line 23  $ftp attach-agent $tcp1
Line 24  $ns at 1.0 "$ftp send 71680"

```

Figure 4.1: A basic ns-2 script.

object *ns* is used to start the simulator and to connect other objects such as nodes, transport agents and applications. Then, four node objects are created to simulate the two end nodes and the two intermediate routers, as shown in lines 3 – 6. The nodes are connected using the *duplex – link* operation of the *ns* object. Several queue management strategies are available for node buffers; for example, drop tail queue management is used in the link from *node1* to *router1* whereas random early detection (RED) queue management is used in the link from *router1* to *router2*. The creation and connectivity of nodes are followed by the creation and connection setup of transport agents as shown in lines 13 – 20. Line 20 is an example of a transport agent parameter adjustment, which illustrates the protocol tuning provision in ns-2.

Since a traffic source is needed to generate the data to be transferred, an object of file

transfer protocol (FTP) application is created as shown in line 22 of Figure 4.1. The FTP object is attached to the transport agent associated with *node1*, thus data can be sent from *node1* to *node2*. The operation *at* of the *ns* simulator object is used to schedule events for the simulator. When the simulator clock time is 1.0 seconds, the *send* operation of the *ftp* object is scheduled, which starts the transfer of 71680 bytes, as shown in line 24. Thus a script for an ns-2 simulation of FTP file transfer is demonstrated in Figure 4.1.

## 4.2 The iSCSI simulation classes

The following subsections describe the iSCSI simulation classes. First, the iSCSI simulation classes for the iSCSI initiator and the iSCSI target are described. Secondly, the simulation classes for iSCSI sessions and connections are discussed. Thirdly, the other support classes for the simulation are described. The support classes describe the state information for the active commands, the sequences of data transfer, and the iSCSI header. Finally, the timer handler classes associated with different iSCSI simulated objects are discussed.

### 4.2.1 The iSCSI initiator and target classes

The ns-2 *Application* class provides the basic functionality for the simulation of traffic generators and applications as shown in the chapter “*Applications and transport agent API*” of [FV03]. Several traffic generators such as exponential, pareto, and constant bit rate (CBR) are simulated as subclasses of the *Application* class. Similarly, applications such as the telnet application and the file transfer protocol (FTP) application are simulated as subclasses of the *Application* class by *TelnetApp* and *FTP* respectively. Since iSCSI is also an application that generates traffic for transport agents, the *IscsiInitApp* and the *IscsiTargetApp* are created as subclasses of the *Application* class to model the iSCSI initiator and the iSCSI target respectively. Figure 4.2 shows the class hierarchy of the iSCSI simulation classes.

In ns-2, several applications such as FTP and TelnetApp simulate their packet transfer

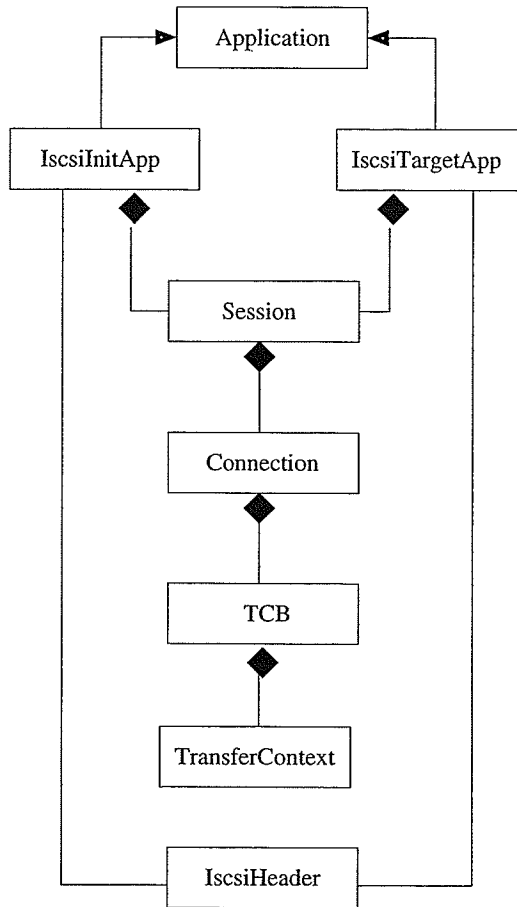


Figure 4.2: The class hierarchy of the iSCSI simulation classes.



by the objects of the *Packet* class. Since the *Packet* class provides packet transfer simulation by computing the transfer time from the values of the packet length and the available bandwidth, there is no real transfer of packets or bytes; the transfer is virtual. On the other hand, application level data transfer can be simulated by using application-level data units (ADUs) of ns-2. For example, in the simulation of a web cache application, the ADUs are used to simulate the transfer of the hyper text transfer protocol (HTTP) tags as shown in the chapter “*Web cache as an application*” [FV03]. Since simulation of iSCSI requires the exchange of header contents between the iSCSI initiator and the iSCSI target, the ADUs are used to simulate the transfer of the iSCSI protocol data units (PDUs).

Figure 4.3 shows the OTcl script where objects of *IscsiInitApp* and *IscsiTargetApp* are created and the values of C++ class variables are modified from the corresponding OTcl class variables. The *IscsiInitApp* and the *IscsiTargetApp* objects are named as *iscsi - init* and *iscsi - target* respectively. The values of class variables such as *first\_burst\_length\_*, *max\_burst\_length\_*, *immediate\_data\_*, and *initial\_r2t\_* can be changed from the OTcl script as shown in Figure 4.3.

The associative array object, *data*, provides the values of the class variables, as shown in Figure 4.3. The associated values of *data* can be obtained by modifying the OTcl script or by reading a text file. The logical *OR* operation is used to negotiate the value of *initial\_r2t\_* but the logical *AND* operation is used for the *immediate\_data\_* negotiation; *immediate\_data\_* and *initial\_r2t\_* are boolean variables.

Figure 4.4 shows a code segment of the *IscsiInitApp* constructor where the C++ class variables are bound with their corresponding OTcl class variables. Although the same variable names are used for both C++ and OTcl classes, different names could also be used. The value shown as comments at the end of each line is only for documentation purposes but the same value is also provided as the default value for the OTcl class variable in the file *ns - default.tcl*.

```

#----- Create iSCSI initiator/target objects ---

set iscsi-init [new Application/IscsiInitApp]
set iscsi-target [new Application/IscsiTargetApp]

#----- Setup for the iSCSI Initiator -----

$iscsi-init set first_burst_length_ $data(first_burst_length)
$iscsi-init set max_burst_length_ $data(max_burst_length)
$iscsi-init set rdsl_ $data(rdsl)
$iscsi-init set immediate_data_ $data(immediate_data)
$iscsi-init set initial_r2t_ $data(initial_r2t)

#----- Setup for the iSCSI Target -----

$iscsi-target set edtl_ $data(edtl)
$iscsi-target set rdsl_ $data(rdsl)
$iscsi-target set first_burst_length_ $data(first_burst_length)
$iscsi-target set max_burst_length_ $data(max_burst_length)
$iscsi-target set immediate_data_ $data(immediate_data)
$iscsi-target set initial_r2t_ $data(initial_r2t)
$iscsi-target set max_cmd_sn_ $data(max_cmd_sn)

#--- Start the simulation (establish an iSCSI session) ----

$ns at 0.0 "$iscsi-init start"
$ns at 0.0 "$iscsi-target start"

```

Figure 4.3: A simulation script to create the iSCSI initiator and the iSCSI target simulated objects.

```

bind("immediate_data_", &immediate_data_); // 1(true)
bind("initial_r2t_", &initial_r2t_); // 1 (true)
bind("max_cmd_sn_", &max_cmd_sn_); // 7
bind("max_burst_length_", &max_burst_length_); // 8
bind("first_burst_length_", &first_burst_length_); // 8

```

Figure 4.4: A sample of C++ code to bind C++ class variables to OTcl variables.

## 4.2.2 The iSCSI session and connection classes

The iSCSI sessions and connections are simulated by the *Session* and *Connection* classes respectively. For example, *CmdSN* is simulated by the *cmd\_sn\_* variable of the simulated session object, and *MaxRecvDataSegmentLength* is simulated by the *rdsl\_* variable of the simulated connection object. Since multiple sessions can be established between the iSCSI initiator and the iSCSI target, a linked list of session objects is maintained by each of the objects, *iscsi – init* and *iscsi – target*. Similarly, multiple connections of a session are simulated by a linked list of connection objects within each session object. Each of the objects, *iscsi – init* and *iscsi – target*, contains a *start* operation, which creates a *Session* object containing a linked list of *Connection* objects. Figure 4.2 shows that the connection objects are contained in the session object.

Figure 4.3 and Figure 4.4 demonstrate the setup of login and session parameters. Static Configuration, a discovery mechanism where each of the initiator and the target knows the name and the address of the other one, is used for the simulation of the iSCSI discovery mechanism. Simulation of the negotiation of login and session parameters is done by the *start* operations of the *iscsi – init* and *iscsi – target* objects. After the execution of the *start* operations, the initiator and the target are at the full feature phase. Hence, the initiator can send the iSCSI commands to the corresponding target.

## 4.2.3 iSCSI support classes

The *TCB* class models a task control block to store the state information for the active command in the connection. The *TCB* object simulates the command parameters such as *CmdSN*, *ITT*, and *EDTL* by *cmd\_sn\_*, *itt\_*, and *edtl\_* variables respectively. Since several commands can be simultaneously active, a linked list of *TCB* objects is maintained in the connection objects of *iscsi – init* and *iscsi – target*. Figure 4.2 shows that the *TCB* objects are contained in the connection object.

The *TransferContext* class manages the transfer of a sequence of data PDUs. A write command can be accomplished by several sequences of data PDUs depending on the *TCB* object's *edtl\_* and the connection object's *rds\_* values. Hence, each *TCB* object contains a linked list of *TransferContext* objects to simulate several sequences of data PDUs. Figure 4.2 shows that the *TransferContext* objects are contained in the *TCB* object.

The *IscsiHeader* class is used to simulate all types of iSCSI headers such as command, data, R2T, and response. The *BasicHeaderSegment* fields such as Opcode, DataSegmentLength, and Initiator Task Tag are simulated by *op\_code*, *data\_segment\_length*, and *itt* respectively. Furthermore, the opcode-specific fields such as ExpectedDataTransferLength, DesiredDataTransferLength, BufferOffset, CmdSN, TargetTransferTag, and StatSN are simulated by *edtl*, *buffer\_offset*, *cmd\_sn*, *ttt*, *ddtl*, and *stat\_sn* respectively.

#### 4.2.4 Timer Handler classes

The simulation of iSCSI uses several timer classes such as *SendTimer*, *CmdTimer*, *EndTimer*, *TCTimer*, *NopOutTimer*, and *NopInTimer*; all timer classes are subclasses of the *TimerHandler* class.

The initiator object contains the *SendTimer* object to simulate the arrival of SCSI commands to the initiator. The arrival of a SCSI command is simulated by scheduling the *SendTimer* object to expire on the arrival time of the associated SCSI command. The *SendTimer* object's *resched* operation schedules the command arrival event and its *expire* operation calls *iscsi - init*'s *send\_cmd* operation to initiate the SCSI command.

Each *TCB* object of the target contains *CmdTimer* and *EndTimer* objects. The *CmdTimer* object provides a timeout mechanism to start the data transfer. When the data transfer is not started within the specified time limit, the *CmdTimer* object times out and the command is retransmitted. The *EndTimer* object ensures the command completion. If the response of the command is not acknowledged within the specified time limit, the

*EndTimer* object times out and the response is retransmitted.

The *TransferContext* object contains the *TCTimer* object to ensure that the transfer of the sequence of data PDUs is completed within the specified time limit.

The initiator object contains the *NopOutTimer* object that sends a nop-out PDU if no iSCSI PDU is sent from the initiator object within the specified time limit. Similarly, the target object contains the *NopInTimer* object that sends a nop-in PDU when no iSCSI PDU is sent from the target object within the specified time limit.

### 4.3 Simulation Scenarios

This section describes the simulation of few features of the iSCSI protocol. First, the simulation of sending multiple commands is described. Then the simulation of a write command is discussed. Finally, the illustration of the simulation of the write command is provided.

#### 4.3.1 Simulation of multiple commands

The simulated iSCSI initiator object, *iscsi - init*, is allowed to send multiple commands depending on the current values of *max\_cmd\_sn\_* and *exp\_cmd\_sn\_* received from the simulated iSCSI target object, *iscsi - target*. The *iscsi - target* object increases *exp\_cmd\_sn\_* when the expected command arrives, and increases *max\_cmd\_sn\_* when the target is ready to receive an additional command. In the iSCSI simulation, *max\_cmd\_sn\_* is immediately increased on the command completion; however, command processing time can be simulated by delaying the *max\_cmd\_sn\_* increase. The maximum number of commands that can be sent is given by the expression  $max\_cmd\_sn_ - exp\_cmd\_sn_ + 1$ , which is also known as a command window. For example, if *max\_cmd\_sn\_* is 10 and *exp\_cmd\_sn\_* is 7, then the command window is 4, which means that *iscsi - init* can send 4 more commands to *iscsi - target*. If the command window is greater than zero, then commands waiting in *iscsi - init*'s command queue can be sent to *iscsi - target* when either of the follow-

ing two events occur at the initiator: 1) the *send\_cmd* operation of *iscsi – init* is called, which indicates a new command has arrived at the initiator; or 2) there is an increase in *max\_cmd\_sn\_* received from *iscsi – target*.

### 4.3.2 Simulation of WRITE command

The data for WRITE command can be transferred as unsolicited and/or solicited data.

#### Simulation of unsolicited data transfer

Although command processing begins in full feature phase, the immediate and/or unsolicited data transfer is negotiated during the login phase. Thus, the simulated iSCSI initiator object, *iscsi – init* and the simulated iSCSI target object, *iscsi – target*, have already agreed on values of *immediate\_data\_*, *initial\_r2t\_*, *first\_burst\_length\_*, and *max\_burst\_length\_* even before the beginning of the WRITE command.

If *iscsi – init*'s *initial\_r2t\_* is *FALSE*, the initiator object can send unsolicited data PDUs to the target object. The sum of all *data\_segment\_length*'s of all unsolicited data PDUs must be less than the *first\_burst\_length\_* value.

A portion of unsolicited data can be sent with the WRITE command as immediate data. When *iscsi – init*'s *immediate\_data\_* is *FALSE*, no data is sent with the command. In this case, the *data\_segment\_length\_* variable of the *IscsiHeader* object, *header*, is set to 0. However, if *immediate\_data\_* is *TRUE*, the value of *data\_segment\_length\_* variable is set to the amount of data being sent with the command. The amount of immediate data cannot be greater than the *rdsi* value of the connection object.

The *final\_bit* of the *header* object is set to *TRUE* when there is no more data to be sent for that sequence of data PDUs. The *final\_bit* is associated with one sequence of data PDUs. If the total data is transferred in five sequences, the last *header* of each sequence will have its *final\_bit* set to 1. The *TCB* object for the command creates the *TransferContext* object to manage the sequence transfer. The *TransferContext* object

creates the object of *TCTimer*, which ensures the sequence transfer completion in the specified time limit.

### Simulation of solicited data transfer

The simulated iSCSI initiator, *iscsi – init*, sends a sequence of data-out PDUs for each R2T received from the simulated iSCSI target, *iscsi – target*. The *TCB* object creates the *TransferContext* object for each sequence of data-out PDUs sent to the target. The amount of data sent in each sequence is *max\_burst\_length\_* bytes except for the last sequence, where the amount of data is the remaining bytes. The *data\_segment\_length\_* field of each data packet is determined by the connection object's *rdsl*. *iscsi – init* maintains the *TCB* object until the command response is received from the target.

### 4.3.3 Illustration of the WRITE command simulation.

The simulated iSCSI initiator, *iscsi – init*, sends a WRITE command to the simulated iSCSI target, *iscsi – target*, where *edtl* is set to 168 KB. The parameters negotiated between the initiator and the target as follows: *first\_burst\_length\_* = 32KB, *max\_burst\_length\_* = 64KB, *immediate\_data\_* = True, *initial\_r2t\_* = False, connection object's *rdsl* = 1KB.

The *IscsiHeader* object, *header*, simulates the header of iSCSI PDUs. Since *immediate\_data\_* is True, *iscsi – init* sends immediate data with the command PDU. Similarly, as *initial\_r2t\_* is False, the command PDU is followed by 21 additional data-out PDUs. The *final\_bit* field of *header* of the last data-out PDU is set to True. Since the *header*'s *edtl* is less than *first\_burst\_length\_*, the target will send three *R2T*s to trigger the initiator to send the remaining data (168K-32K = 136K). The initiator will send a sequence of data-out PDUs for each R2T it will receive from the target. Each of the data sequences that will be triggered by the first two *R2T*s will be *max\_burst\_length\_* (64 KB) length, 44 PDUs. The third *R2T* requires 6 data-out PDUs. The above data transfer example is summarized as follows:

Sequence	PDU's Transferred	Bytes Transferred
Unsolicited Sequence	22	32K
For the first R2T	44	64K
For the second R2T	44	64K
For the third R2T	6	8K

When the target object receives all the data-out PDUs, the target object increases its *stat\_sn\_* variable, sends a response PDU, and can increase the *max\_cmd\_sn\_* to allow the initiator object to send more commands. The initiator object acknowledges the *stat\_sn\_* by increasing the value of *exp\_stat\_sn\_*.

## 4.4 Simulation of Error Recovery

The numbering of iSCSI PDUs is used to recover the missing and corrupted PDUs. The receiver object maintains a linked list of missing packets. For example, iSCSI target's *list* is the linked list of missing commands. Similarly, the iSCSI initiator's *list* is the list of missing responses; the *TransferContext* object's *list* is the list of missing data PDUs; the initiator's *TCB* object's *list* is the list of missing *R2T*s. In addition to *list*, the simulated objects contain *exp* variable. The *exp* variable indicates that each PDU with sequence number less than *exp* has been received. Since iSCSI has the provision of direct memory access (DMA), where data from the network interface card can be directly transferred to the application memory buffers, the out-of-order PDUs are delivered to the higher layers without any temporary storage. Hence, there is no need to simulate the storage of out of order PDUs. Finally, the *max* variable of the simulated objects indicates the maximum sequence number received so far.

### 4.4.1 Error Recovery Algorithm

Figure 4.5 describes the algorithm for the recovery of missing or corrupted PDUs in the iSCSI simulation. Initially, the missing list, *list*, is empty and the values of *exp* and *max* are 0 and -1 respectively. If all the PDUs arrive in order then *list* remains empty. Although



the values of *exp* and *max* are increased accordingly, *exp* will remain greater than *max*. If the out of order PDU is received, the error recovery algorithm will be activated and *max* will become greater than *exp*, as shown in case 1 of Figure 4.5 where  $r > i$ . The error recovery algorithm is active when *max* is greater than *exp*. If the expected PDU arrives and error recovery algorithm is active, there is no change in the value of *max*. However, the value of *exp* is modified as shown in Figure 4.5. If any missing PDU arrives, *max* and *exp* remain same but *list* is modified because the received PDU is removed from the *list*. If the sequence number of the received PDU is greater than *max*, the value of *max* is changed to the received sequence number and *list* is also modified accordingly, as shown in case 2 of Figure 4.5.

#### 4.4.2 Simulation of Error Recovery for the READ command

Figure 4.6 shows an example of the recovery of missing data-in PDUs for a READ command. It is assumed that the target requests a data-ack when 4 in-order packets arrive at the initiator. The target sends data-in PDUs with sequence numbers 0...5. When the initiator receives all data-in PDUs, the initiator will send data-ack of 4, *max* will be changed to 5, *exp* will be changed to 6, and *list* will remain empty. When the initiator receives a data-in PDU with sequence number 9, the initiator will request retransmission of missing PDUs by sending a SNACK PDU for the missing data-in PDUs from 6 to 8, the value of *max* will be increased to 9, the value of *exp* will remain as 6, the *list* will contain sequence numbers from 6 to 8. When the initiator receives 10...12, *max* will be changed to 12; *exp* will remain as 6; *list* will remain as 6, 7, 8. In this case, a data-ack PDU will not be sent because there are missing PDUs. When the initiator receives 16, *max* will be changed to 16, *exp* will remain as 6, the *list* will be increased to contain 6, ... 8, 13, ... 15.

When the initiator receives 14, *max* will remain as 16, *exp* will remain as 6; the *list* will be reduced to contain 6, ... 8, 13, and 15. When the initiator receives the expected packet 6, *max* will remain as 16; *exp* will be increased to 7; *list* will be reduced to contain

*Initial condition:*  $exp = 0$ ;  $max = -1$ ; and  $list = \{\}$ .

*Pre-condition:*  $exp = s_i$ ;  $max = s_j$ ; and  $i < j$ .

**Case 1**  $list = \{\}$

IF  $r == i$

$exp := s_{i+1}$ ;

IF  $r > i$

Let  $r = q$ ; and without any loss of generality  $i < j < q$ .

$max := s_q$ ;

$list := \{s_{j+1}, s_{j+2}, \dots, s_{q-2}, s_{q-1}\}$ .

END IF

**Case 2**  $list = \{s_m, s_{m+1}, \dots, s_{n-1}, s_{n+1}, \dots, s_{p-1}, s_p\}$ ;

where  $i < m < n < p < j$ .

IF  $r == i$

$exp := s_m$ ;

IF  $m \leq r \leq p$

Let  $r = o$ ; and without any loss of generality  $n < o < p$ .

$list := list - \{s_o\}$

$list = \{s_m, s_{m+1}, \dots, s_{n-1}, s_{n+1}, \dots, s_{o-1}, s_{o+1}, \dots, s_{p-1}, s_p\}$ .

IF  $r > p$

Let  $r = q$ ; and without any loss of generality  $p < j < q$

$max := s_q$ ;

$list := \{s_m, s_{m+1}, \dots, s_{n-1}, s_{n+1}, \dots, s_{p-1}, s_p, s_{j+1}, s_{j+2}, \dots, s_{q-2}, s_{q-1}\}$ .

ENDIF

Figure 4.5: The error recovery algorithm used in the simulation of error recovery of missing PDUs.

Remarks	Received	max	exp	list
Initial Values		-1	0	{}
Receives in-order PDUs Sends DataACK 4.	0, 1, ..., 5	5	6	{}
Sends SNACK for 6 – 8.	9	9	6	{6, 7, 8}
Receives in-order PDUs DataACK is not sent.	10, 11 and 12.	12	6	{6, 7, 8}
Sends SNACK for 13 – 15.	16	16	6	{6, 7, 8, 13, 14, 15}
Receives the missing PDU.	14	16	6	{6, 7, 8, 13, 15}
Receives the expected PDU.	6	16	7	{7, 8, 13, 15}
Receives all the PDUs except one.	7, 8 and 15	16	13	{13}
Receives the last missing PDU. Sends DataAck 17.	13	16	17	{}

Figure 4.6: Error Recovery Processing of Data-In PDUs for a READ Command.

7, 8, 13, and 15. When the initiator receives 7, 8, and 15, *max* will remain as 16, *exp* will be changed to 13; *list* will be reduced to contain only 13. Finally, when the initiator receives 13, *max* will remain as 16; *exp* will be changed to 17; and *list* will become empty.

## 4.5 Simulation of Message boundaries within TCP byte stream.

In iSCSI over TCP model, an additional layer is needed to insert message boundaries because TCP is a byte stream protocol with no message boundaries. The iSCSI application layer sends the iSCSI PDU to the Markers PDU Aligned (MPA) framing layer. The MPA layer assembles one or more iSCSI PDUs into one Framing Protocol Data Unit (FPDU). MPA peers exchange FPDUs using TCP and deliver iSCSI PDUs to the associated iSCSI initiator/target.

MPA layer is implemented by *MpaInit* and *MpaTarget* classes, which are subclasses of the *Application* class. Figure 4.7 shows the creation of the *MpaInit* object, *mpa1* and the *MpaTarget* object, *mpa2*. *mpa1* and *mpa2* are attached to the iSCSI initiator object, *iscsi – init* and the iSCSI target object, *iscsi – target*, respectively. The TCP objects *tcp1* and *tcp2* are attached with MPA objects *mpa1* and *mpa2* respectively, as shown in Figure

```

#----- Create MPA objects -----

set mpa1 [new Application/MpaInit $tcp1 $data(outfile)]
set mpa2 [new Application/MpaTarget $tcp2 $data(outfile)]

$mpa1 connect $mpa2

#----- Attach MPA objects to the iSCSI objects -----

$mpa1 attach-ulp $iscsi-init
$mpa2 attach-ulp $iscsi-target

```

Figure 4.7: A simulation script to create Marker PDU Aligned (MPA) objects

#### 4.7.

Since there is no transfer of actual packets in ns-2, the FPDU headers are stored in a queue that is shared by both MPA peers, *mpa1* and *mpa2*. The target's MPA object, *mpa2*, stores the bytes received from its associated transport agent, *tcp2*. When the number of bytes delivered from *tcp2* are equal or greater than the size of the FPDU then the next available FPDU is retrieved from the queue. Since TCP is a reliable transport agent, all the FPDUs are delivered in order. *mpa2* extracts the iSCSI PDUs from the received FPDU and delivers the iSCSI PDUs to the attached iSCSI target object, *iscsi-target*. Consequently, the insertion of iSCSI message boundaries in the TCP byte stream is simulated in ns-2.

## 4.6 Simulation of iSCSI over UDP

The simulation of iSCSI over the User Datagram Protocol (UDP) is implemented by *MpaInitUdp* and *MpaTargetUdp* which are subclasses of *MpaInit* and *MpaTarget* respectively. The objects of *MpaInitUdp* and *MpaTargetUdp* are attached to the iSCSI initiator and target objects respectively. The FPDUs created by MPA objects, *mpa-init-udp* and *mpa-target-udp*, are transferred using UDP. Since message boundaries are not lost in UDP data transfer, there is no need to simulate temporary storage buffers to assemble FPDUs, as the temporary buffers were required for the iSCSI over TCP simulation.

Since the information regarding application memory buffers is known in the iSCSI PDUs, the PDUs can be moved directly to the application memory buffers. Therefore, there is no need to buffer the out-of-order iSCSI PDUs transferred by the UDP agent. As UDP is an unreliable transport protocol, the iSCSI error recovery mechanism is used to recover the lost or corrupted iSCSI PDUs.

## 4.7 Traffic Generation

The SCSI commands for the iSCSI initiator can be simulated by probability distributions and trace files.

### 4.7.1 Probability Distributions

The SCSI commands can be generated by either an arbitrary distribution of probabilities or a well defined probability distribution function. For command sizes, the arbitrary distribution of probabilities is commonly done in benchmarking and performance evaluation experiments. For example, [HY03] describes the probability mass function (pmf) of the command size as follows:

$$p_X(x) = \begin{cases} 0.1, & x = 8KB \\ 0.2, & x = 16KB \\ 0.3, & x = 32KB \\ 0.4, & x = 64KB \end{cases} \quad (4.1)$$

The cumulative distribution function (CDF) is defined as follows:

$$F_X(t) = \sum_{x \leq t} p_X(x), \quad -\infty < t < \infty \quad (4.2)$$

Figure 4.8 shows the probability distribution function of the command sizes described in Equation 4.1.

The interarrival times between successive SCSI commands can be considered as exponentially distributed. The memoryless property of the exponential distribution implies

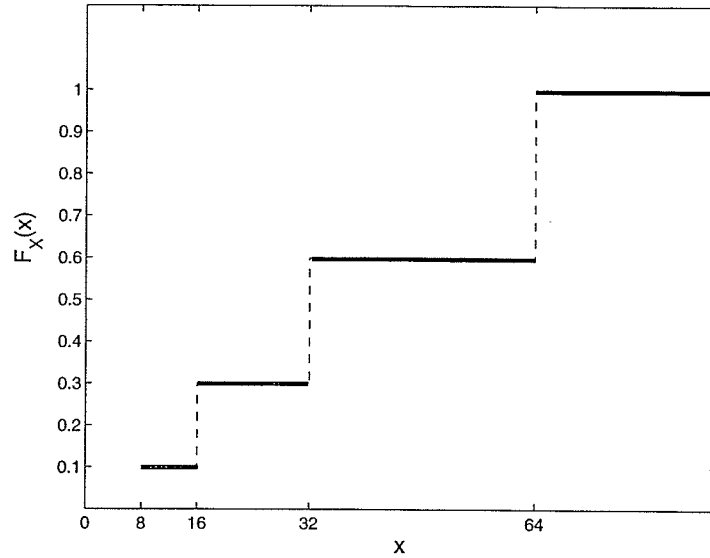


Figure 4.8: Cumulative distribution function (CDF) of the command sizes.

that the time to wait for the arrival of a new SCSI command is independent of the time being spent in waiting for the command arrival. The exponential distribution function is described as follows:

$$F(x) = \begin{cases} 1 - e^{-\lambda x}, & 0 \leq x < \infty \\ 0, & \textit{otherwise} \end{cases} \quad (4.3)$$

The probability density function (pdf) of the interarrival times is given by the the following equation:

$$f(x) = \begin{cases} \lambda e^{-\lambda x}, & x > 0 \\ 0, & \textit{otherwise} \end{cases} \quad (4.4)$$

## 4.7.2 Trace Files

Trace files containing I/O traces are useful in system verification and performance evaluation. The same trace file can be used for several simulation experiments to illustrate the impact of several parameters for similar load.

```
0,20941264,8192,W,0.551706,Alpha/NT
0,20939840,8192,W,0.554041
0,20939808,8192,W,0.556202
1,3436288,15872,W,1.250720,0x123,5.99,test
1,3435888,512,W,1.609859
1,3435889,512,W,1.634761
0,7695360,4096,R,2.346628
```

Figure 4.9: Records of a trace file

The Storage Performance Council (SPC), <http://www.storageperformance.org>, provides several trace files that can be used as input to simulation experiments to measure system performance. The trace files are available at <http://traces.cs.umass.edu/storage/>. A trace file, *Financial1.txt*, which records I/O traces from OLTP (Online Transaction Processing) applications from two large financial institutions is used in the simulation experiments. Figure 4.9 shows few records from a trace file, where each trace file record contains the following fields:

1. Application Specific Unit (ASU): If there are  $n$  logical units (LUs) then the number may vary from 0 to  $n - 1$ .
2. Logical Block Address (LBA): The block address for the data transfer.
3. Size: The number of bytes to be transferred for this command.
4. Opcode: The type of command, R or W.
5. Timestamp: The relative time in seconds starting from the first record of the trace file.
6. Optional: More optional fields can be added to the trace file.

Furthermore, trace files can be created using probability distributions or any other user defined inputs. Several trace files are created using Poisson distribution and other empirical user defined inputs.

## 4.8 Incorporating Errors in iSCSI Simulation

The simulation of iSCSI provides two types of errors: deterministic and stochastic. The former is used for protocol verification by introducing errors in specific iSCSI PDUs. The latter is used for performance evaluation where errors are introduced based on a given bit error rate, BER.

Deterministic errors are generated using a uniform probability distribution. The probability error rate for all types of iSCSI PDUs can be given in the Tcl script. For example, if the error rate of data-out PDUs is 0.1 then every tenth data-out PDU will be in error. Similarly, if the error rate of R2T PDUs is 0.05 then every twentieth R2T PDU will be erroneous. Hence, the deterministic error mechanism generates errors in the desired packets. Since the output of the deterministic errors is already known, the simulation of the protocol is verified by comparing the simulation output with the expected output.

The stochastic errors are generated from the bit error rate, BER, which is given from the Tcl script. Since all iSCSI PDUs are transferred using the the same channel, the same BER is used for the error generation of all the packets. If the number of bits in each packet is  $B$  and the bit error probability is  $P_{BER}$ , the packet error probability,  $P_{packet}$ , is computed as follows:

$$\text{Packet Error Probability, } P_{packet} = 1 - (1 - P_{BER})^B \quad (4.5)$$

The generation of non-deterministic errors is illustrated as follows: if  $B = 12000$  ( $1500 \times 8$ ) and  $P_{BER} = 10^{-7}$  then  $P_{packet} = 0.0012$ . These non-deterministic errors facilitate the iSCSI performance evaluation for different network conditions.

## 4.9 Verification of the Simulation

The verification of the iSCSI simulation is performed by comparing the simulation output with the expected output. The verification comprises the following scenarios:



- Without errors
- With errors
- Without any underlying lower layer protocol

First, the sample output files are created to verify the simulation output. Several test cases are developed for each of the simulated protocol features such as:

- Immediate data transfer
- Solicited data transfer
- Unsolicited data transfer
- Combined solicited and unsolicited data transfer
- Several sequences of a write command (which means multiple R2Ts)
- Multiple write and read commands
- The command window
- Phase collapse (the response is given in the last data-in PDU)
- Nop In/Out mechanism

Secondly, the error recovery mechanism is verified by generating the deterministic errors. Since the erroneous packets are known in the deterministic error mechanism, the expected output can be obtained before the simulation. Hence, the simulation of error recovery mechanism is verified by comparing the simulation output with the expected output. The test cases for the error recovery mechanism verification are as follows:

- Command recovery

- The target's recovery R2T mechanism, which recovers missing data-out PDUs.
- The initiator's SNACK mechanism, which recovers missing R2Ts, responses, and data-in PDUs.

Finally, the protocol features were verified in the absence of other lower layer protocols. The iSCSI initiator and the iSCSI target were directly connected to verify the simulation of iSCSI protocol features without any effect from the lower layers. A constant delay factor was used to simulate the data transfer time.

Thus, the iSCSI simulation is verified by generating several test cases to compare the simulation output with the expected output.

## **4.10 The protocol features not simulated**

Although the iSCSI simulation includes most of the protocol features such as multiple commands per connection, error recovery within command, solicited and/or unsolicited data transfer, and phase collapse, the protocol features associated with other iSCSI related protocols such as discovery, authentication, task management functions are not implemented. The above features are ignored because their simulation is not essential for the performance evaluation of iSCSI for IP storage and transport protocols.

Furthermore, protocol features such as multiple connections per session, processing overhead, and disk transfer time are also not simulated. Although the above features can be helpful for more accurate performance evaluation, the iSCSI simulation is simplified by avoiding the simulation of these features.

## **4.11 Summary**

A partial implementation of an iSCSI simulation within ns-2 is presented. This chapter has provided the details of classes, data structures, message passing between objects of

different layers, and the simulation of error recovery mechanism. Errors can be simulated by deterministic or stochastic approaches and the SCSI commands can be obtained from a trace file or from a probability distribution function. The iSCSI simulation can be used for performance evaluation of storage systems for various network conditions and several transport layer protocols.

# Chapter 5

## Performance Evaluation

This chapter provides a performance evaluation of the iSCSI protocol using its ns-2 simulation.

### 5.1 Performance Metrics

The following subsections describe the performance evaluation metrics. First, the response time of a command is described. Secondly, throughputs based on commands and data are explained. Finally, the bandwidth utilization is discussed.

#### 5.1.1 Response Time

The response time of a command is an aggregate of several components. Figure 5.1 identifies the most significant points in the timeline of a command completion. The time  $t_1$  is the time when the iSCSI initiator receives the command from the SCSI initiator. The command may be immediately processed or queued depending on the size of command window. Recall that the command window is described by the expression  $MaxCmdSN - ExpCmdSN + 1$ . The time  $t_2$  is the time when the iSCSI command is sent to the target.

The target receives the command at time  $t_3$ . The target will immediately start processing the command. Since command sequence numbers (CmdSNs) are session wide, CmdSNs in a connection are not necessarily contiguous. Consequently, a hole in the command

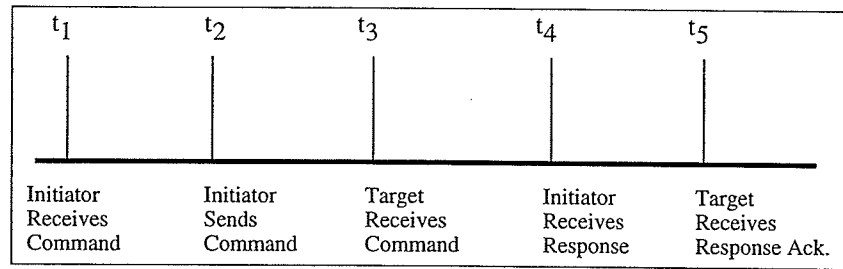


Figure 5.1: Response Time Definition

sequence numbers does not indicate any missing command PDUs. The timeout mechanism at the initiator detects the missing/corrupted command PDUs and retransmits the missing/corrupted commands. If the command is a WRITE command, the target will send one or more *R2T* request packets depending on the size of the command and the state of the target. If a READ command is received, the target sends data-in PDUs. The target can start sending data-in PDUs immediately or with some delay depending on its current available resources. The resource allocation time and the time between successive data-in PDUs are ignored because their values are negligible as compared to other values of time. The target immediately sends a response PDU on the command completion.

The initiator receives the response from the target at time  $t_4$ . The initiator then sends the acknowledgement of the response that is received by the target at time  $t_5$ . The response time from the initiator's perspective is  $t_4 - t_1$ . The target will release any allocated resources at time  $t_5$ . The time intervals during command completion may be described as follows:

1.  $t_2 - t_1$ : The command waits in the queue at the initiator until the target increases the command window size.
2.  $t_3 - t_2$ : The time required to transfer the command. The average time can be reduced by sending several iSCSI commands in one TCP segment.
3.  $t_4 - t_3$ : The time required to transfer the bytes in response to the command.
4.  $t_5 - t_4$ : The time required to send the response acknowledgement from the initiator to the target.

The command completion time for the iSCSI protocol can be described by the following equation:

$$T(S) = kS^\alpha \quad (5.1)$$

After taking log on both sides of Equation 5.1 we get the following equation, Equation 5.2:

$$\log T = \log k + \alpha \log S \quad (5.2)$$

In Equation 5.2, T is the time for the command completion; S is the size of data load, which is called *ExpectedDataTransferLength* in the iSCSI protocol; k is a constant protocol overhead determined by the protocol parameters such as R2Ts, immediate and/or unsolicited data transfer, *FirstBurstLength*, *MaxBurstLength*, *MaxRecvDataSegmentLength*; and  $\alpha$  is a critical exponent, which is a function of network conditions, protocol parameters, and data source.

Figure 5.2 describes the experimental setup to observe the response time variation for different values of *ExpectedDataTransferLength*. The response times for different simulated command sizes are shown in the graph of Figure 5.3, where the response time increases no more than linearly with a linear increase in the command size, as was expected from Equation 5.2.

Figure 5.3 shows that the graph is basically sub-linear or linear, then there is a sharp increase in the response time and then the graph is again linear. Because immediate and/or unsolicited data sent with the command is limited to *FirstBurstLength* bytes while the remaining data must be sent only after receiving an *R2T* PDU from the target, the discontinuities or sharp increases are observed in the command response times. If the target sends all *R2T*s at the same time, the initiator has a choice to send all data-out PDU either at the same time or with some delay. The simultaneous transfer of large number of data-out PDUs will result in intermediate buffers overflow, causing retransmissions of transport layer PDUs. In the above experimental setup, the buffer-queue-limit was 500 packets and

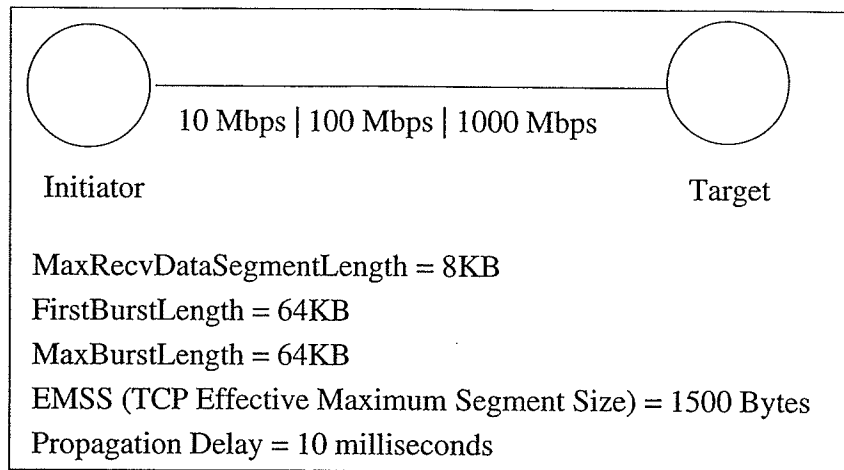


Figure 5.2: An experimental setup for the iSCSI initiator and the iSCSI target.

the TCP *EMSS* was 1500 bytes, resulting in 750 KB buffer capacity. Hence, when the cumulative size of simultaneously transferred data-out PDUs was greater than 750 KB, the packet overflow was observed, resulting in sharp increase in the response time.

Figure 5.4 shows the effect of changing the size of the buffer, which is at the initiator for the link between the initiator and the target. The size of the buffer varies from 100 to 1000 packets. There is a spike when the buffer size is 100 because a large number of data-out PDUs are sent to the transport agent causing packets to be dropped at the node buffer, and the dropped packets must be retransmitted by the transport agent. Since the buffer size of 1000 is large enough to hold all the packets, there is no spike in the curve.

### 5.1.2 Throughput and Utilization

The throughput and utilization are metrics to measure system performance. The observation interval during simulation is  $T$ , the number of commands that arrive in the interval is denoted by  $A$ , the number of commands completed is denoted by  $C$ , the total number of bytes sent is  $L$ , and the bandwidth is  $BW$ . The command throughput  $X_c$ , the data throughput  $X_d$ , the utilization  $U$  and the average command arrival rate  $\lambda$  are defined as follows:

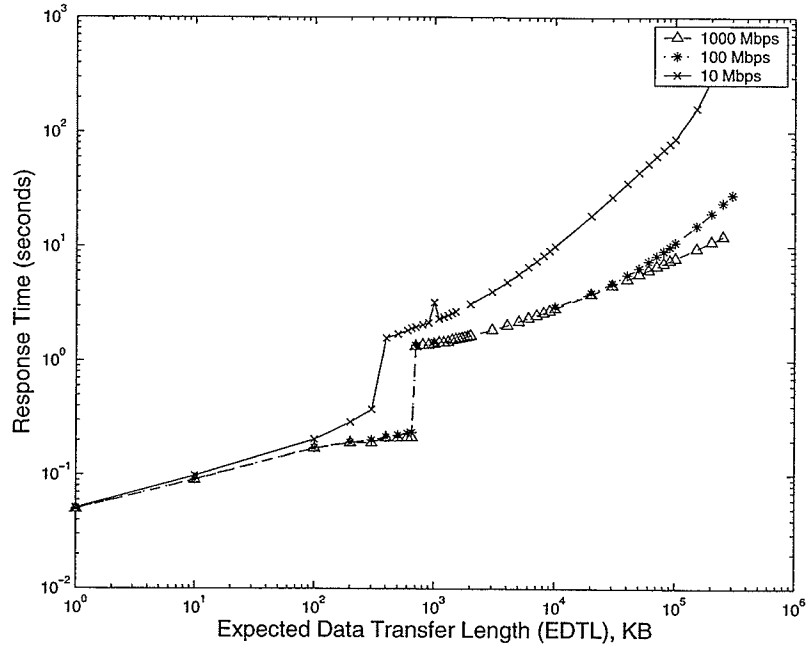


Figure 5.3: Response time vs Expected Data Transfer Length.

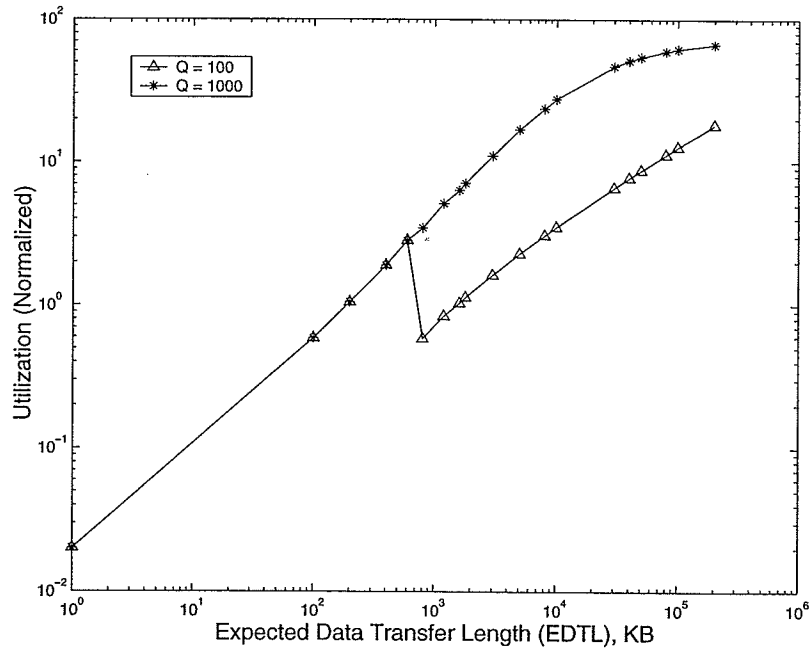


Figure 5.4: Utilization for different buffer sizes.



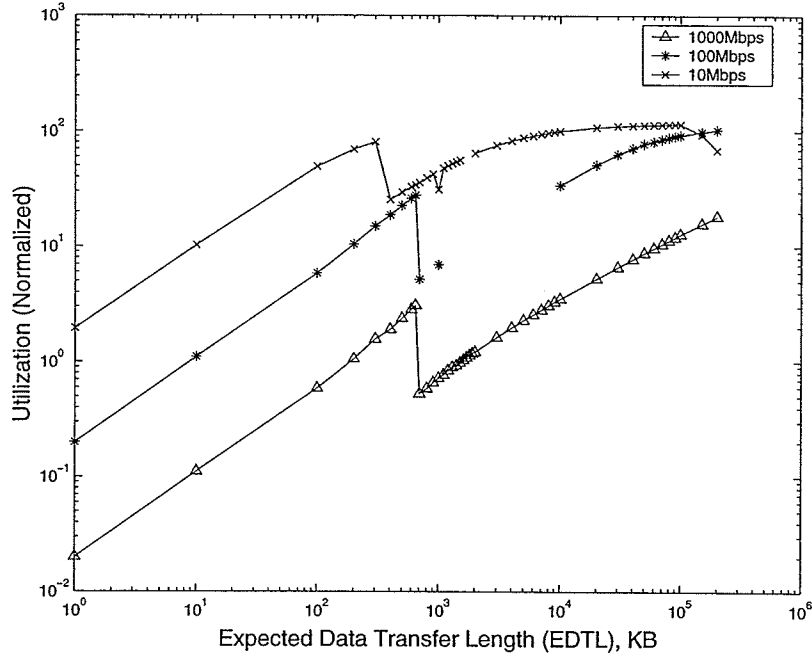


Figure 5.5: Utilization Vs Expected Data Transfer Length.

$$\text{Command Throughput, } X_c = \frac{\text{Commands Completed}}{\text{time}} = \frac{C}{T} \quad (5.3)$$

$$\text{Data Throughput, } X_d = \frac{\text{Total number of bytes}}{\text{time}} = \frac{L}{T} \quad (5.4)$$

$$\text{Utilization, } U = \frac{\text{Throughput}}{\text{Bandwidth}} = \frac{X_d}{BW} = \frac{L/T}{BW} = \frac{L}{T \times BW} \quad (5.5)$$

$$\text{Arrival Rate, } \lambda = \frac{\text{Number of Arrivals}}{\text{Time}} = \frac{A}{T} \quad (5.6)$$

Figure 5.5 shows the utilization of the bandwidth for the experiment shown in Figure 5.2. The utilization increases as the size of the data increases. The utilization of 10Mbps is the highest as compared to 1000Mbps. The utilization of 10Mbps is the first one to saturate, followed by 100Mbps. The bandwidth of 1000Mbps does not saturate because of

Number of commands	Size of each command
1	80MB
2	40MB
4	20MB
8	10MB
16	5MB
40	2MB
80	1MB

Figure 5.6: A set of different input commands with the same total number of bytes.

the *ExpectedDataTransferLength* range chosen for the experiment.

## 5.2 Command window size

Figure 5.6 illustrates a set of different input commands where although the number of commands and their sizes are varied but the total bytes to be transferred remain fixed at 80MB. Figure 5.7 shows that, although the number of commands is increased, there is no change in the system performance as long as the command window size is greater than the maximum number of simultaneously active commands. A straight line of zero slope is obtained when the command window size is greater than the maximum number of simultaneously active commands. However, when the command window size is fixed at 8 and the number of commands is increased, the slope is zero while the number of commands is less than or equal to 8 and then the throughput drops linearly with the linear increase in the number of commands.

Figure 5.8 shows the effect of varying the command window size on the throughput. As expected one can improve the response time by adjusting the command window size. Also evident is the effect of diminishing returns when the command window becomes unnecessarily large. The command throughput reaches the maximum value when command window size is 30, thus any further increase in the command window size will not yield any benefit. The system parameters for the simulation experiment are as follows:

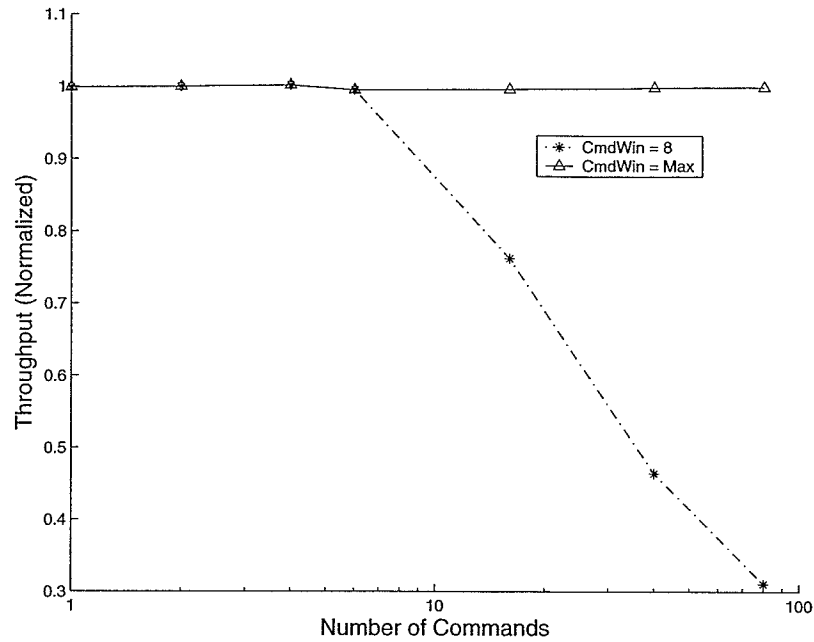


Figure 5.7: Throughput for varying number of commands with constant data payload.

- *MaxRecvDataSegmentLength*=1500 Bytes
- *FirstBurstLength* = 64 KBytes
- *MaxBurstLength* = 64 KB
- *Bandwidth*=1000 Mbps
- *EMSS* = 1500 Bytes
- *PropagationDelay* = 10 milliseconds.

The details of the trace file used in the experiment are as follows: observation interval = 1919.74 seconds; number of requests = 100000; and total bytes requested = 871,516,160 Bytes.

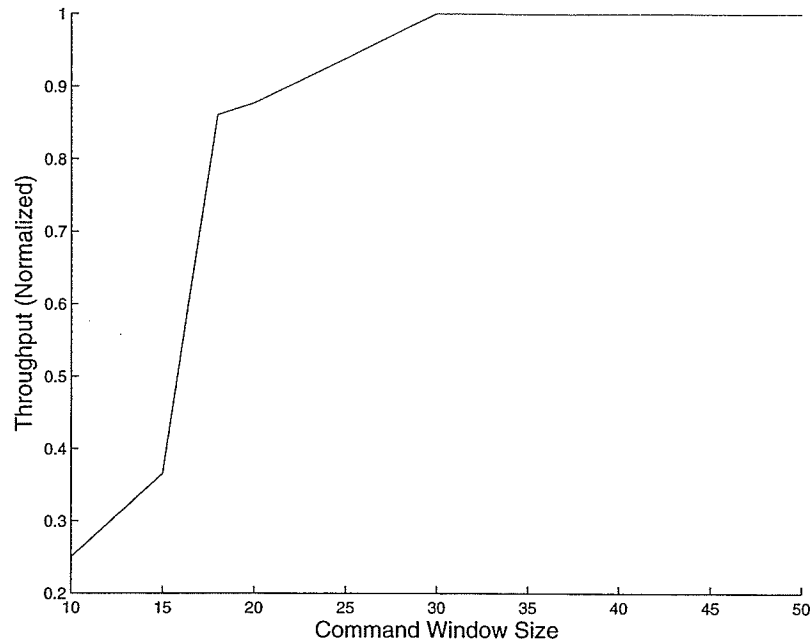


Figure 5.8: Throughput and Command Window Size.

### 5.3 Bit Error Rate and Throughput

Figure 5.9 shows the graph of the throughput variation as the bit error rate increases. It should be noted however that iSCSI error detection is effective in detecting packet errors and recovering from them. Under normal operating conditions with a BER on the order of  $10^{-9}$  the impact on the system performance is minimal.

### 5.4 iSCSI over UDP

The iSCSI over UDP is compared with iSCSI over TCP in error free environment. The packet loss can only occur because of buffer overflow. The experimental details are follows:

- bandwidth=100MB
- propagation delay=10 milliseconds
- queue-limit at the nodes=1000 packets

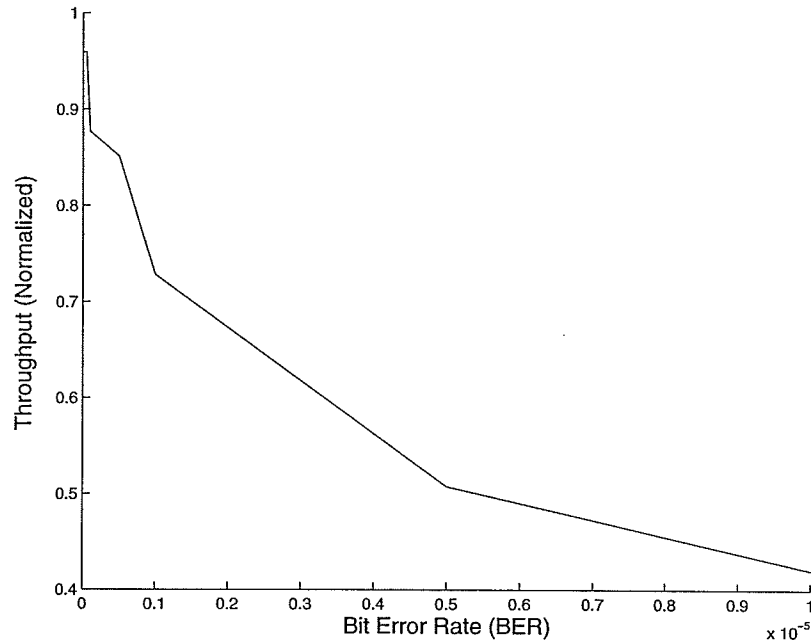


Figure 5.9: Throughput and Bit Error Rate (BER).

- *MaxRecvDataSegmentLength* = 944bytes
- *MaxBurstLength* = 64KB
- *FirstBurstLength* = 64KB.

Figure 5.10 shows that UDP performs better for shorter values of EDTL. However, the performance of UDP was drastically degraded for higher values of EDTL. Figure 5.10 shows a huge jump in UDP response times, from 0.1195 seconds to 20.0445 seconds, when *EDTL* was increased from 1 MB to 2 MB. The huge jump was because the *TransferContext* timeout value was 10 seconds and the iSCSI error recovery mechanism was not activated until the timeout was fired. If the timeout values are not well chosen, the performance can be severely degraded.

Figure 5.11 shows that suitable *TransferContext* timeout values can improve iSCSI performance. The performance crossover point between UDP and TCP is increased to 8

EDTL KB	TCP sec	UDP sec
1	0.07	0.02
10	0.1106	0.0208
100	0.17411	0.04314
1000	0.2776	0.1195
2000	0.3458	20.0445
3000	0.454	30.0493
4000	1.622	40.054
5000	1.86	50.0588
6000	1.928	60.0635
7000	2.036	70.0681
8000	2.104	80.0726
9000	2.212	90.077
10000	2.281	100.081
20000	3.162	200.119
30000	4.0436	310.081

Figure 5.10: Response times for TCP and UDP for different values of EDTL.

MB as compared to 1 MB, which was obtained in Figure 5.10. The iSCSI TransferContext timeout value in case of UDP was 0.1 second but the value was changed to 1 second for *TCP*. If the timeout value of 0.1 seconds is also used for TCP, iSCSI performance will be severely degraded because iSCSI error recovery mechanism will interfere with TCP error recovery, resulting in more redundant retransmissions leading to more buffer overflows.

## 5.5 Summary

Performance evaluation is done for various parameters of the iSCSI protocol. The results of the simulation experiments show that the iSCSI protocol can perform quite well if parameters such as the command window size, the amount of unsolicited data, and timeout values are properly tuned. Recall that data is transmitted in sequences of packets, one sequence for each corresponding *R2T* packet. As expected, sequence-based transmission without proper regard for network conditions can introduce spikes or burstiness in the traffic. Er-

EDTL KB	UDP seconds	TCP seconds
1000	0.1438	0.2561
1500	0.1861	0.3000
2000	0.2484	0.3438
2500	0.2907	0.3878
3000	0.3530	0.4315
4000	0.4576	0.5193
5000	0.5621	0.6070
6000	0.6670	0.6947
7000	0.7712	0.7824
8000	0.8758	0.8701
9000	0.9804	0.9578
10000	1.0849	1.0455
10500	1.1472	1.0893
10600	1.1482	1.0981

Figure 5.11: Response times for TCP and UDP where the TransferContext timeout value for UDP is 0.1 seconds and the TransferContext timeout value for TCP is 1 second.

atic transmission behavior can be avoided by proper pacing of *R2T* packets.

# Chapter 6

## CRC Errors and iSCSI

This chapter addresses potential error control redundancy that may be inherent in running iSCSI over wide area IP networks. The iSCSI standard recommends the use of a 32-bit error control CRC on iSCSI data frames of, typically 8 KBytes. In a similar manner the TCP transport layer provides a TCP 16-bit checksum on each TCP data packet of, typically 1500 bytes, and the Ethernet physical layer provides a 32-bit CRC on each frame, again typically 1500 bytes. In addition to data packets, there are approximately half as many control frames of minimal length, typically 64 bytes in a given iSCSI session. The probability of data and control frames or packets escaping error detection is discussed in this chapter and alternative mechanisms that may be required to ensure high data integrity are suggested.

Although the context of the present discussion is iSCSI, the analysis of the CRC behavior during its transient toward asymptotic behavior applies equally well to all systems where CRCs are deployed.

### 6.1 Introduction

A simplified view of the Internet has packets being shunted between routers using Ethernet as the underlying physical layer protocol. On top of Ethernet, the protocol is IP, which does not provide any error control. TCP, which is on top of IP, is usually responsible for reliable end to end flow and error control. With these assumptions, we are usually interested in



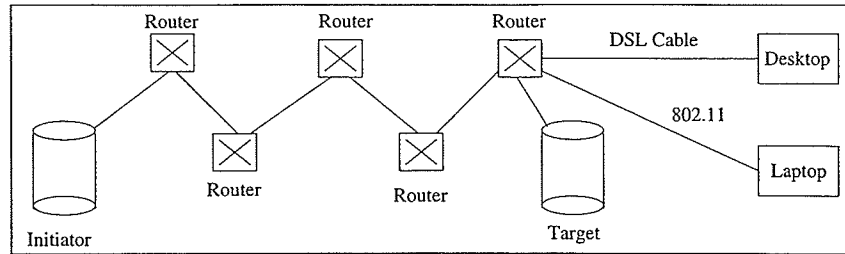


Figure 6.1: Typical Internet scenario showing various nodes and connectivities.

considering certain “what if” scenarios. For example, determining what happens in the cases where errors occur. At the physical layer, a 32-bit CRC error checker is used to provide error detection. On the wired segments, if an error is caught, the packet is dropped; whereas on a wireless (802.11) link, the frame is not acknowledged. The *lost* packet is assumed to have not arrived at its destination and retransmission is attempted some limited number of times. If repeatedly not acknowledged, a time-out will eventually occur at a higher level and end-to-end recovery techniques will be used.

These methods work quite well under the assumption that the error is detected. A problem arises for packets in error that avoid detection. An *aliased* packet is an erroneous packet that is indicated as not an erroneous packet by the error checker. Aliased packets have largely been ignored or never considered, although well documented [SP00]. One of the reasons that they perhaps were ignored was that under most circumstances they are pathological, occurring so seldom that their effect can be ignored. Error coverage from a 32-bit maximal length CRC such as that employed in Ethernet alone is 1 part in  $2^{32}$  (or 4.3 billion). This is, however, an asymptotic limit, which will be discussed later. Aliased packets from a combined iSCSI, TCP, and Ethernet system would be considerably less likely. One can, however, argue that at each level where data is manipulated, buffered, or moved there is a possibility of errors occurring manifesting themselves as aliased packets.

Figure 6.1 shows a scenario illustrating a typical client server architecture (target initiator iSCSI) where users are accessing data over the Internet. The situation of interest here

is the connection between an iSCSI target and iSCSI initiator which is part of a network running various protocols over IP in addition to the iSCSI protocol of this particular session. Of central importance to this discussion is that the connection can be characterized with a bit error rate (BER) and primary error control is provided by a 32-bit CRC at the physical layer, the 16-bit TCP checksum at the transport layer and a 32-bit CRC at the iSCSI application layer.

### 6.1.1 CRCs

CRCs or cyclic redundancy checks are extremely powerful techniques for detecting errors. In almost all literature, the probability of CRC aliasing is taken to be extremely small. In fact most references will cite the probability of an aliased packet escaping detection by the CRC as being  $2^{-32}$  if a 32-bit maximal length CRC checker is employed. This value is well known and results from the observation that the state transition matrix associated with the CRC is doubly stochastic and in the limit all states are equally likely. As such, the problem of aliased packets is rarely if ever addressed. This  $2^{-32}$  limit however is an asymptotic bound. On the Internet, the packet size is basically governed by the maximum frame size of an Ethernet frame, 1500 bytes, or minimal frame sizes of 64 bytes. The question arises as to whether the asymptotic bound is close enough under these circumstances.

The basic model that will be presented here has been around for approximately 15 years in relation to VLSI signature analysis and the probability of signature registers making similar mistakes [WDGS88], [Dav90]. One of the difficulties at the time, as well as in the present, was in computing the probability of errors for large signature analyzers. The difference of the present discussion is that VLSI testing is not constrained in a similar manner as packets on the Internet. As such, re-evaluating the aliasing performance of Ethernet CRCs, and TCP checksum error control warrants further investigation. In addition, there is a class of unexplained aliasing packet that may be contributed by larger than expected aliasing probabilities, as reported in [SP00].

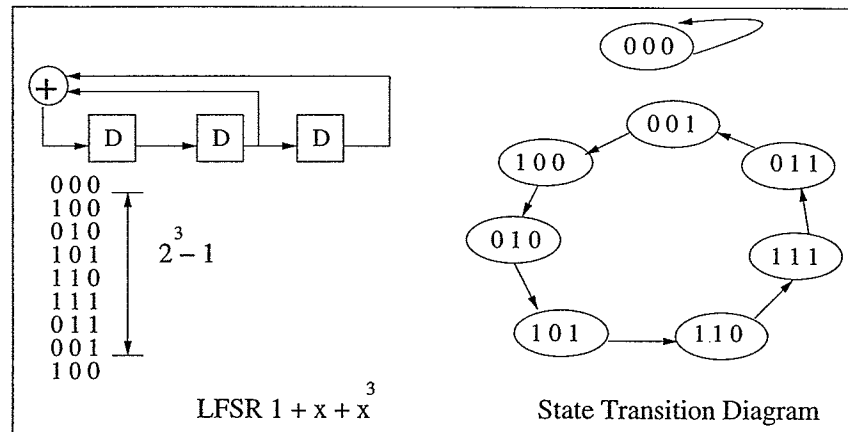


Figure 6.2: An LFSR generating a 3 bit CRC

A similar approach to VLSI signature analysis will be presented here illustrating the usefulness of both the general approach as well as approximation techniques in the context of aliased packets. The type of CRC that is considered here is a maximal length CRC, that is, one whose state transition diagram is of maximal length. That is not to say that all CRCs employed for error control are of maximal length, but it makes the present discussion and coding simpler. A maximal length CRC is essentially equivalent to a linear feedback shift register (LFSR) whose feedback taps are selected in such a manner as to produce a state machine that traverses the entire non-zero state in one cycle, as illustrated in Figure 6.1.1. For a maximal length LFSR, the non-zero state space is one large cycle, plus the all zero state. The LFSR will remain in the all zero state if the incoming data pattern is also all zeros. In what follows, the word “packet” will be used in association with an Ethernet frame or IP packet, TCP segment, or iSCSI frame. The more specific terminology will be clear from the context.

There are alternative hardware implementations for the same generator polynomial as illustrated in Figure 6.3. The main difference is in the orientation of the feedback taps. The corresponding state transition diagrams are isomorphic. The type of CRC illustrated in Figure 6.3 is more commonly implemented in hardware than that of Figure 2 because it

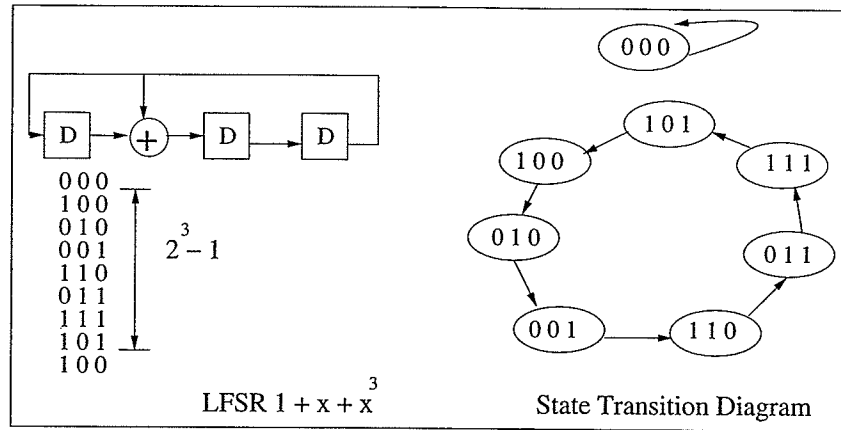


Figure 6.3: An alternative CRC implementation for the same function as shown in Figure 6.1.1

generates the remainder when the input polynomial (data) is divided by the CRC polynomial. This allows for more efficient processing at both the receiver and sender.

Although not considered further here, error performance comparisons have been made between the two implementations for a 4 bit generator represented by the polynomial  $1 + x^3 + x^4$  and the results are identical. For the remainder of the discussion here, the LFSR of Figure 6.1.1 was employed.

## 6.2 Aliased Packets

Aliased packets are those that necessarily have more than one bit in error. For the model here, it suffices to start in the all zero state and have an error occur in the packet and have at least another error occur such that the effect of the first error is masked. Figure 6.4 illustrates this effect on the state transition diagram in the case of an erroneous packet being detected, as well as for an aliased packet.

Figure 6.5 illustrates how a CRC register is implemented in practice. Data arrives serially and is mixed with the LFSR representing the CRC. Upon receipt of the packet, the value remaining in the CRC register is the signature and can be compared with the known

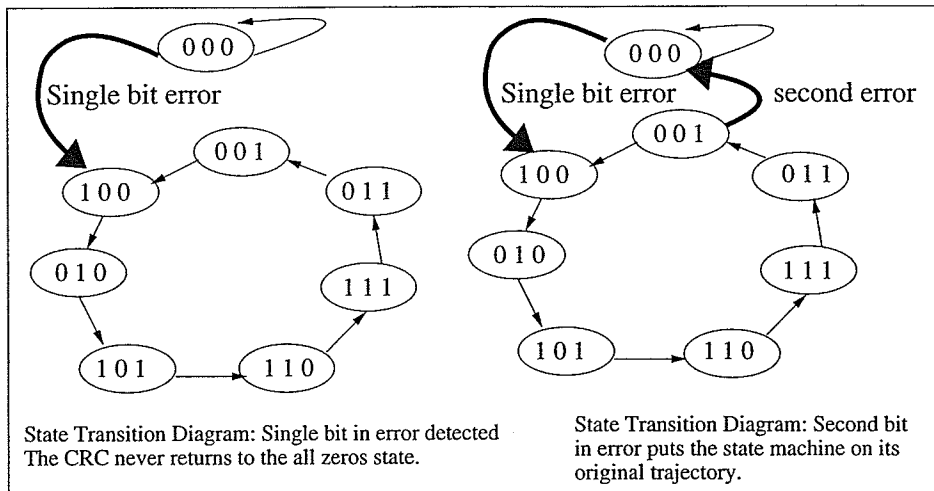


Figure 6.4: State space trajectory of a detected packet in error and an aliased packet.

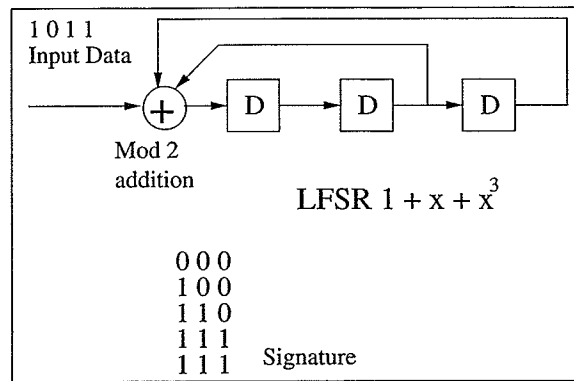


Figure 6.5: Signature for an input data stream.

good signature, which can be made to be the all zero vector. If the two are different, the packet contains errors and is discarded, or not acknowledged. The packet is subsequently retransmitted at either the physical layer or as a result of error control at the transport layer. Similarly, in the case of detection at the iSCSI layer the iSCSI frame would be retransmitted.

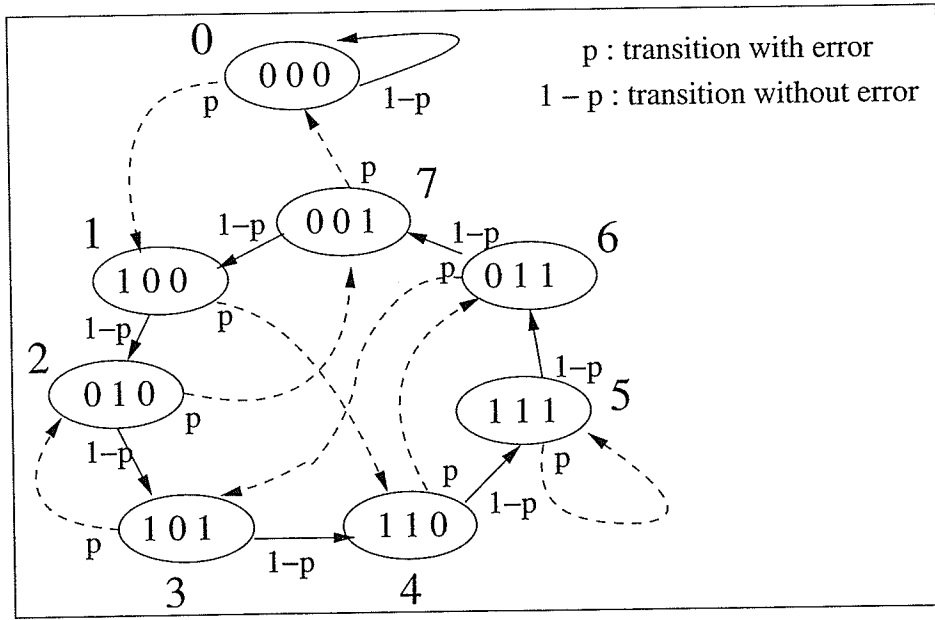


Figure 6.6: Markov model for a 3 bit CRC

### 6.3 Aliased Packet Probabilities

The basic model for calculating aliased packet probabilities is a simple Markov model derived from the state transition diagram in the presence of a medium with a bit error rate. The Markov model for the example 3 bit CRC previously discussed is shown in Figure 6.6.

Mathematically the Markov model can be cast as a probability transition matrix equation relating the probability of being in any given state at time  $t$  to that of being in any state at time  $t + 1$ , as shown in Equation 6.1.

$$\begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \\ P_6 \\ P_7 \end{bmatrix}_{t+1} = \begin{bmatrix} 1-p & 0 & 0 & 0 & 0 & 0 & 0 & p \\ p & 0 & 0 & 0 & 0 & 0 & 0 & 1-p \\ 0 & 1-p & 0 & p & 0 & 0 & 0 & 0 \\ 0 & 0 & 1-p & 0 & 0 & 0 & p & 0 \\ 0 & p & 0 & 1-p & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1-p & p & 0 & 0 \\ 0 & 0 & 0 & 0 & p & 1-p & 0 & 0 \\ 0 & 0 & p & 0 & 0 & 0 & 1-p & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \\ P_6 \\ P_7 \end{bmatrix}_t \quad (6.1)$$

The probability matrix is doubly stochastic and has the well known property of asymptotically converging to the probability of any state being  $\frac{1}{2^n}$  where  $n$  is the number of bits in the CRC. The probability of a packet aliasing through the CRC (i.e. being in error) is modeled as the probability of a bit stream in error generating the error free signature. This model is the same for all packets and as such the model can use an all-zero packet as the error free packet and a packet in error as any packet that contains non zero bits. In effect, for cyclic codes such as these, it is only necessary to analyze the error polynomial to analyze the aliasing behavior. This allows one to start the matrix iteration with  $P_0 = 1$ , with the remaining state probabilities as all-zero. Iteration of the state transition probability matrix allows one to calculate the probability of being in any given state at time equal to the iteration number. Since the probability of aliasing is defined to be the conditional probability, the probability of aliasing is calculated as the probability of being in the P[0] state, given that an error has occurred. The state transition matrix is sparse and hence relatively efficient techniques can be deployed to calculate the state transition probabilities as data bits enter the CRC register. However, the algorithm is still  $O(i2^n)$  where  $i$  is the number of iterations we are interested in and  $n$  is the number of bits in the CRC. Although exact results such as those presented here are primarily for 16-bit maximal length CRCs, results of extensive simulation for 32-bit CRCs are presented subsequently.

Figure 6.7 illustrates the probability of a packet aliasing given the channel has an associated bit error rate (BER) ranging from  $10^{-4}$  to  $10^{-8}$ . These BERs are likely higher than those encountered on reliable networks, but may reflect a cumulative error over a number of routers.

The interesting aspects of Figure 6.7 are as follows. For BERs on the order of  $10^{-4}$  or  $10^{-8}$ , the asymptotic limit of P(aliasing) being approximately  $\frac{1}{2^{16}}$  is not approached until the CRC has evaluated well over 12,000 bits. At 12,000 bits the probability of aliasing is still approximately 5 times higher than predicted by the asymptotic limit. In the figure the

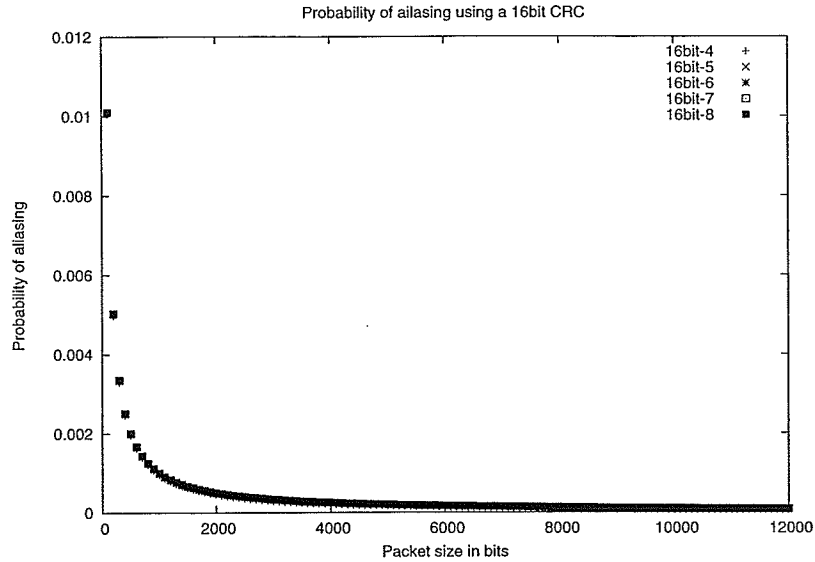


Figure 6.7: Probability of packets aliasing

curves are basically on top of each other. For the 16-bit CRCs analyzed here, the probability of minimum size ethernet frames (64 Bytes) aliasing is on the order of 0.002 or 0.2 percent of packets detected in error are aliased.

For packets on the order of iSCSI data frames (8 KBytes), the probability of aliasing is very close to that predicted by the asymptotic limit (within 1%). Again short control packets such as acknowledgements would be more susceptible to aliasing. Although preliminary, the asymptotic behavior appears to be present even as the size of the CRC register increases. For a BER of  $10^{-7}$ , the behavior of a 20-bit maximal length CRC is almost identical to that of a 16 bit maximal length CRC, as illustrated in Figure 6.8.

## 6.4 TCP Error Control

In addition to CRC aliasing there is also a potential problem when an aliased packet is sent on to the TCP layer where the segment is checksummed.

Checksums are error detection techniques designed for lower computation cost rather



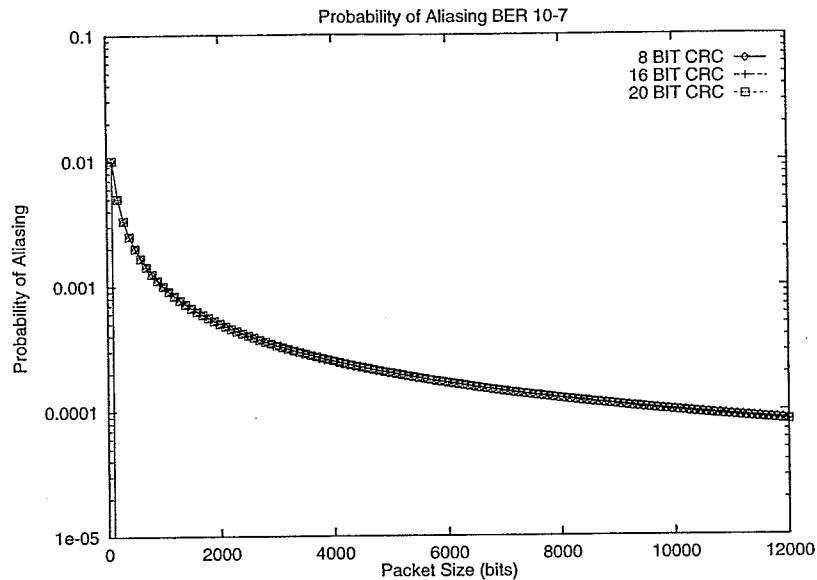


Figure 6.8: Aliasing transient from 20, 16 and 8 bit CRCs

than higher chances of error detection capability. Basically, checksum is a 16-bit ones-complement sum of the data. For uniformly distributed data, the checksum error detection probability is  $2^{-16}$ ; however, for non-uniformly distributed data, the checksum's performance can be significantly worse, such as  $2^{-10}$  [SGPH98].

Unfortunately the Markov model technique used previously can not be used without requiring dense matrix multiplications. A simple rare event simulator was written that indicated that the aliasing probability was approximately 0.0027 (total packets 100,001 with 16,743 packets in error and 45 aliased packets). The error model was a uniform BER stream of approximately  $1.5 \times 10^{-5}$  with a single bit in word error probability of  $\frac{8}{2^{15}}$ . The packet size was 1500 bytes. Increasing the TCP segment size increases the probability of the TCP checksum aliasing.

The simulator was run with various seeds and test packets yielding variations on the results on the order of plus or minus one packet being aliased. For example, assuming there is a 0.00008 probability of aliasing from the CRC at a BER of  $10^{-5}$ , of these packets .0027 (0.27%) of them will pass through the transport layer in error to the application layer,

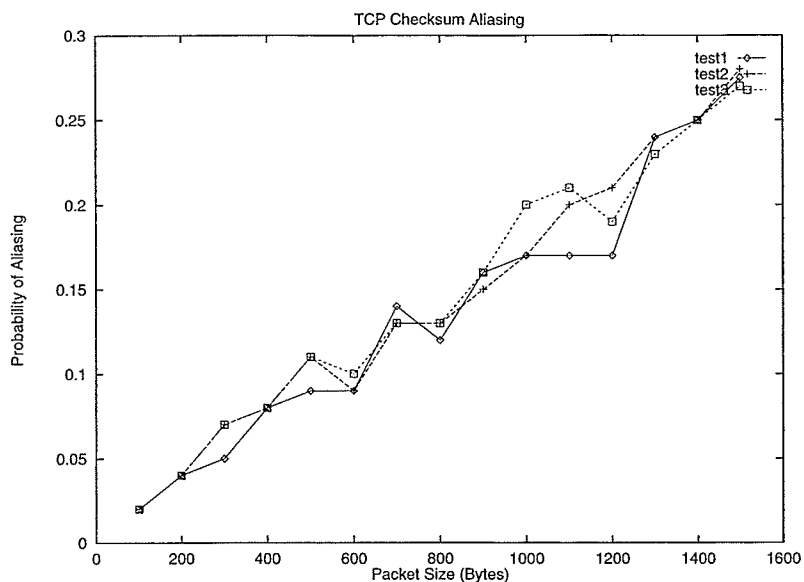


Figure 6.9: Probability of TCP checksum aliasing

which means that there is a probability of 0.00000021 that an erroneous packet will make it to the application layer. The expected number of packets seen before an error occurs would then be 4,800,000, or equivalently a 7GB file. Statistical independence is assumed here between the two detection schemes. In cases where extreme data integrity is required this may be significant, though it should be noted that these results were obtained with a 16-bit CRC and 16-bit TCP checksum. The TCP checksum can not be improved upon, but it is anticipated that a 32-bit CRC would provide many of orders magnitude greater protection than the result mentioned here.

Figure 6.9 illustrates the probability of TCP checksum aliasing as a function of packet size. The checksum calculation in Figure 6.9 is based on the number of packets aliased over the number in error. The simulation used a constant BER of approximately  $1.5 \times 10^{-5}$ .

## 6.5 CRC Simulations

An attempt to predict maximal length CRC aliasing has been initiated through simulation. The coverage provided by the CRC defined by :  $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$  is being simulated. Even with a BER on the order of  $10^{-4}$ , we have not been able to detect an aliased packet as a result of the CRC improperly classifying an erroneous packet. The simulation has exercised 1 billion maximum size packets running for over 280 hours on a Sun Fire 6800, 24 processor machine. The simulation is a batch mode process and utilizes under 5% of the available CPU resources.

### 6.5.1 iSCSI Error Control

Adding an iSCSI CRC to the above scenario would also improve aliasing performance considerably. Unfortunately, the CRC aliasing does not account for the discrepancies seen on real networks with real data. Here erroneous packets fairly routinely escape detection as discussed in [SP00]. A signature attached to the entire file is a potential solution although it may mean retransmitting a very large file once the entire file is checked for integrity, hence an impractical solution. A better solution, and one that perhaps should be included in a reliable iSCSI implementation, would be one that provides signatures on data from iSCSI boundaries of 64 KBytes using an *HMAC*, as well as requiring iSCSI to be implemented within the IPsec authentication framework at the packet level. In the case of control packets, their aliasing behavior warrants further consideration. Although not integral to the actual data they can play a role in how well the application and transport layer respond in terms of error and flow control. In certain circumstances the control information may also be of crucial importance.

## 6.6 Experimental Results

<sup>1</sup> Two experiments were performed to attempt to determine if CRC/TCP aliasing errors could be detected in a manner similar to that in the study by Stone [SP00]. The wide area network environment was created by connecting a wireless LAN and an ADSL connection from a residential host to the University and back.

The first was a file transfer experiment over the wireless LAN. The experiment consisted of FTP over TCP of a 4 MB file. An MD5 hash was performed after every transfer to verify that no aliasing had occurred - i.e. that no error escaped both the CRC check and the TCP checksum. The network topology consisted of a receiving station with an ethernet switch at 100 Mbps; a wireless access point (802.11b at 11 Mbps) connected to a wireless host (running windows XP). The 4MB file was sent 100,082 times without ever failing the MD5 check. In effect approximately 3.2 Terabits of actual data was transmitted without incurring 1 aliased packet.

A second experiment transferred data of various packet types and various sizes from a residential host to the University and back over an ADSL connection. The trials spanned a total of approximately 12 hours, on different days, with 18 hops between the two hosts used. The UDP packets were sent from the residential ADSL host to the UM, where a Linux iptables firewall rule redirected the UDP packets back to the originating IP host. Thus all figures refer to UDP packets that have traveled a total of 36 hops. UDP checksums were disabled (set to 0x0000) both in the incident and reflected packets, thereby allowing the monitoring of packets in error that escaped the physical layer error detection. Although there were several periods of excessive round trip times and packet loss in excess of 20 packets in a row, only one instance of packet errors escaping CRC detection was observed. Out of the 313,985 packets and approximately 235MB sent in tests so far, there have been 21 bytes in error, which occurred within packets of 67 bytes in size. The source of the errors

---

<sup>1</sup>The experimental results were obtained by Mr. M. Laskowski and Mr. J. Berkes and are included here as they directly relate to experimental evidence of aliasing and/or packet corruption.

is still undetermined and not reproducible. This type of intermittent aliasing is perhaps of more concern, as it is a source of errors that is nondeterministic.

## 6.7 Discussion and Summary

The Markov analysis used to model aliased packets that escape CRC or TCP detection does not seem to account for the anomalously high degree of aliased packets seen in real traffic studies [SP00]. Notwithstanding this, if packet aliasing errors appear to be present, a model should be developed to account for aliasing independent of its origin. These aliased packets do in fact influence network performance and potentially data integrity.

The results presented in this chapter lead to the following conclusions:

1. Results of network models and their simulations should take aliasing into account. However, current simulations, such as those based on ns-2, do not take aliasing into account. Based on our studies and those of [SP00], this aliasing model should be parametric, that is value such as  $P_{aliasing} \cong 10^{-8}$  based on empirical measurements such as those of [SP00].
2. Even in the presence of strong error detection techniques, data may be corrupted without the user being aware.
3. If data integrity is of importance, stronger digital signing, for example, using MD5 should be employed.

When designing a system where a CRC is used, the possible limitations of CRC should be considered rather than simply CRC's asymptotic behavior. This is especially true if the size of the packets is small and the integrity of those packets is important.

Based on mounting empirical evidence, all systems should consider aliasing as inherent, whether caused by the mathematics or not, at least in cases where data integrity is important.

# Chapter 7

## Summary and Future Work

### 7.1 Summary

This dissertation concerned itself with performance evaluation of mass data storage and efficient transport of data across a wide area network. Contributions made during the course of the Ph.D. study include:

- The development of an iSCSI module for use within ns-2. This is the first iSCSI protocol implementation for ns-2. As ns-2 is an open source initiative, the implementation can be used by others interested in simulating iSCSI under various network conditions.
- The performance evaluation of the iSCSI protocol was performed for several network conditions. Since iSCSI data transfer occurs in sequences of solicited data transfer, the burstiness of the data can be controlled by appropriate values of iSCSI parameters, such as *FirstBurstLength*, *MaxBurstLength*, and command window size.
- Protocol tuning was done to improve the iSCSI performance. The iSCSI performs better for larger values of MTU and EMSS. The buffer queue size at the target node was seen to have a significant impact on the performance of iSCSI over UDP.
- The performance of iSCSI error control mechanism is analyzed when iSCSI is deployed over unreliable transport protocol such as UDP. The simulation results verify

that in low error environment, the iSCSI error recovery mechanism is sufficient to send SCSI commands over UDP. However, for relatively higher error rates, the iSCSI over UDP performance degrades drastically as compared to iSCSI over TCP.

- In terms of CRC error control, a Markov model analysis, simulation and experimental measurements were done to provide estimates of packet aliasing.

## 7.2 Future Work

The iSCSI protocol has immense potential to address the needs of emerging storage applications. The current research can be extended in at least three directions as described below.

First, the simulation of iSCSI in ns-2 can be improved by the addition of the following features:

- The simulation of kernel queues such as user buffers, transmit queue, and send queue.
- Modeling the processing overhead at the initiator and the target.
- The estimation of optimum timeout values, which are dependent on the network conditions and the available resources. For example, the timeout value to transfer a sequence of data-out PDUs can be a function of *DesiredDataTransferLength*, round-trip-time, and the network congestion.
- The estimation of the optimum values for the data transmission rate, the command window size, and the size of *ExpectedDataTransferLength*.
- Compare the ns-2 simulation results with the results of other benchmarking tools.

Secondly, performance evaluation of the iSCSI protocol can be enhanced as follows:

- More accurate performance evaluation of the iSCSI protocol by incorporating kernel level performance tools.

- Tuning of iSCSI protocol based on the simulation results of ns-2. It seems that iSCSI performance could be improved when successive transmissions of sequences of data PDUs are delayed by a suitable factor, which is a function of the network condition and the available end-to-end resources.
- Analyze the impact of different queuing policies, such as drop tail, random early detection (RED), and adaptive RED, over different transport agents (TCP, UDP) for the iSCSI protocol.

Finally, the iSCSI protocol can improve the performance of various applications that use block I/Os and are dependent on their storage systems. For example, Database Management System (DBMS) performance is highly dependent on the block I/Os of the storage system. The DBMS performance could be improved by the following techniques:

- Since the DBMS works with blocks of data, the disk access is a very significant factor in database tuning and query optimization. The iSCSI protocol could assist in the optimization of the overall block I/Os by associating database block I/Os with SCSI block I/Os.
- Distributed DBMSs have to deal with the challenge of optimum allocation and replication strategies. It would be interesting to investigate the possibility of any association of iSCSI based storage management with the DBMS allocation and replication strategies.

Since the iSCSI protocol has the potential to address several challenges of storage systems, the performance evaluation of various data storage applications should be revisited. Hence, there is a significant demand for future research to assess the impact of the iSCSI protocol over the current computing and storage resources.



# Glossary

The list of acronyms and other related terms are described here.

ACK	Acknowledgement
ADU	Application Data Unit
AHS	Additional Header Segment
ASU	Application Specific Unit
ATA	Advanced Technology Attachment
BHS	Basic Header Segment
CHAP	Challenge Handshake Authentication Protocol
CIFS	Common Internet File System
CmdSN	Command Sequence Number
CRC	Cyclic Redundancy Check
DataSN	Data Sequence Number
DDP	Direct Data Placement
EDTL	Expected Data Transfer Length
EMSS	Effective Maximum Segment Size
ExpCmdSN	Expected Command Sequence Number
ExpDataSN	Expected Data Sequence Number
ExpStatSN	Expected Status Sequence Number
FC	Fibre Channel
FCIP	Fibre Channel over Internet Protocol
FFP	full-Feature Phase
FFPO	Full-Feature Phase Only
FIM	Fix Interval Marker
FPDU	Framing Protocol Data Unit
Gbps	Gigabit per second
HBA	Host Bus Adapter
HDD	Hard Disk Drive
iFCP	Internet Fibre Channel Protocol
IKE	Internet Key Exchange
I/O	Input/Output
IO	Initialize Only
IP	Internet Protocol
ips	IP storage
IPsec/IPSec	Internet Protocol Security
iqn	iSCSI qualified name
iSAN	iSCSI-based Storage Area Network
iSER	iSCSI Extension for RDMA
iSCSI	SCSI over IP
iSNS	Internet Storage Name Service

ISID	Initiator Session ID
IT	Initiator_Target
IT L	Initiator_Target_LUN
ITN	iSCSI Target Node or iSCSI Target Name
ITT	Initiator Task Tag
JBOD	Just a Bunch Of Disks
KRB5	Kerberos version 5
LAN	Local Area Network
LBA	Logical Block Address
LO	leading Only
LONP	Login Operational Negotiation Phase
LU	Logical Unit
LUN	Logical Unit Number
MAN	Metropolitan Area Network
MaxCmdSN	Maximum Command Sequence Number
MC/S	Multiple Connections per Session
MD-5	Message Digest version 5
MIB	Management Information Base
MPA	Markers PDU Aligned
MSS	Maximum Segment Size
MTU	Maximum Transfer Unit
NA	Not Applicable
NAS	Network Attached Storage
NFS	Network File System
NIC	Network Interface Card
NOP	No Operation
NSG	Next Stage
OS	Operating System
PDU	Protocol Data Unit
PKI	Public Key Infrastructure
R2T	Ready to Transfer
R2TSN	Ready to Transfer Sequence Number
RAID	Redundant Array of Independent Disks
RDMA	Remote Direct Memory Access
RFC	Request for Comments
SAM	SCSI Architectural Model
SAM2	SCSI Architectural Model 2
SAN	Storage Area Network
S-ATA	Serial ATA
SCSI	Small Computer Systems Interface
SCTP	Stream Transmission Control Protocol
SLP	Service Location Protocol
SN	Sequence Number

SNACK	Sequence Number Acknowledgement or Selective Negative Acknowledgement
SNMP	Simple Network Management Protocol
SNIA	Storage Networking Industry Association
SoHo	Small office/Home office
SoIP	Storage over IP
SSID	Session ID
StatSN	Status Sequence Number
SW	Session-Wide software
TCB	Task Control Block
TCP	Transmission Control Protocol
TOE	TCP/IP Offload Engineer
TPGT	Target Portal Group Tag
TSID	Target Session ID
TSIH	Target Session Identifying Handle
TUF	TCP Upper-Level-Protocol Framing
TTT	Target Transfer Tag
UDP	User Datagram Protocol
ULP	Upper Level Protocol or Upper Layer Protocol
ULPDU	Upper Layer Protocol Data Unit
URL	Uniform Resource Locator
URN	Uniform Resource Name
UTF	Universal Transformation Format
VPN	Virtual Private Network
WAN	Wide Area Network
WG	Working Group

# Bibliography

- [ADR03] Dave Anderson, Jim Dykes, and Erik Riedel. More than an interface – SCSI vs. ATA. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 245–257, 2003.
- [AGPW03] Stephen Aiken, Dirk Grunwald, Andrew R. Pleszkun, and Jesse Willeke. A performance analysis of the iSCSI protocol. In *Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST 2003)*, pages 123–134, April 2003.
- [BHH<sup>+</sup>04] Mark Bakke, Jim Hafner, John Hufferd, Voruganti Kaladhar, and Marjorie Krueger. iSCSI naming and discovery. *RFC 3721*, April 2004.
- [CGY01] Jeffrey S. Chase, Andrew J. Gallatin, and Kenneth G. Yocum. End system optimizations for high-speed TCP. *IEEE Communications, Special Issue on High-Speed TCP*, 39(4):68–74, 2001.
- [CN] Don Capps and William D. Norcott. *IOzone - Filesystem benchmark tool*. <http://www.iozone.org>.
- [Cok] Russell Coker. *Bonnie++*. <http://www.coker.com.au/bonnie++>.
- [CSE<sup>+</sup>03] Mallikarjun Chadalapaka, Hemal Shah, Uri Elzur, Patricia Thaler, and Michael Ko. A study of iSCSI extensions for RDMA (iSER). In *Proceed-*

- ings of the ACM SIGCOMM workshop on Network-I/O convergence*, pages 209–219. ACM Press, 2003.
- [CT90] David D. Clark and David L. Tennenhouse. Architectural considerations for a new generation of protocols. In *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 200–208, 1990.
- [Dav90] Rene David. Comments on signature analysis for multiple output circuits. *IEEE Transactions on Computers*, 39(2):287–288, February 1990.
- [DSE<sup>+</sup>] Dan B. Dov, Daniel Scheibli, Joe Eiler, Ming Zhang, Richard Riggs, Thayne Harmon, Tony Asleson, and Vedran Degoricija. *Iometer*. <http://www.iometer.org>.
- [FV03] Kevin Fall and Kannan Varadhan. *The ns Manual*. The VINT Project, December 2003.
- [FV04] Kevin Fall and Kannan Varadhan. The network simulator - ns-2. <http://www.isi.edu/nsnam/ns/>, version 2.26, 2004.
- [HY02] Xubin He and Qing Yang. A caching strategy to improve iSCSI performance. In *Proceedings of the 27th Annual IEEE Conference on Local Computer Networks*, pages 278–285, 2002.
- [HY03] Xubin He and Qing Yang. SPEK: A storage performance evaluation kernel module for block level storage systems. In *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems MASCOTS' 03*, 2003.
- [Kat01] Jeffrey Katcher. PostMark: A new file system benchmark. Technical Report TR3022, Network Appliance, [http://www.netapp.com/tech\\_library/3022.html](http://www.netapp.com/tech_library/3022.html), 2001.

- [LD03] Yingping Lu and David H. C. Du. Performance study of iSCSI-based storage subsystems. *IEEE Communications Magazine*, 41(8):76–82, August 2003.
- [LoB02] Michael T. LoBue. Surveying today’s most popular storage interfaces. *Computer*, 35(12):48–55, December 2002.
- [MTJM02] Charles Monia, Franco Travostino, Wayland Jeong, and Edwards Mark. iFCP - a protocol for internet fibre channel networking. *IP Storage Working Group’s Internet Draft*, December 2002.
- [RRW02] Murali Rajagopal, Elizabeth Rodriguez, and Ralph Weber. Fibre channel over TCP/IP (FCIP). *IP Storage Working Group’s Internet Draft*, August 2002.
- [SAM02] SAM-2. SCSI architecture model - 2. *T10/1157D*, September 2002.
- [SGPH98] Jonathan Stone, Michael Greenwald, Craig Partridge, and James Hughes. Performance of checksums and CRCs over real data. *IEEE Transactions on Computers*, 6(5):529–543, October 1998.
- [SP00] Jonathan Stone and Craig Partridge. When the CRC and TCP checksum disagree. In *Proceedings of the ACM SIGCOMM*, pages 309–319, 2000.
- [SPRC04] Hemal Shah, James Pinkerton, Renato Recio, and Paul Culley. Direct data placement over reliable transports. *Remote Direct Data Placement Working Group’s Internet Draft*, February 2004.
- [SSCZ04] Julian Satran, Constantine Sapuntzakis, Mallikarjun Chadalapaka, and Efri Zeidner. Internet small computer systems interface - iSCSI. *RFC 3720*, April 2004.
- [VS01] Kaladhar Voruganti and Prasenjit Sarkar. An analysis of three gigabit networking protocols for storage area network. In *Proceedings of the 20th*

*IEEE International Performance, Computing, and Communications Conference (IPCCC), 2001.*

- [WDGS88] Thomas W. Williams, Wilfried Daehn, Matthias Gruetzner, and Cordt W. Starke. Bounds and analysis of aliasing errors in linear feedback shift registers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 7(1):75–83, January 1988.