PERFORMANCE ANALYSIS AND IMPLEMENTATION OF OBJECT BASED STORAGE

A Thesis by

Ashish Maddi

Bachelor of Engineering, Mahatma Gandhi Institute of Technology, JNTU, India, 2006

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

August 2009

PERFORMANCE ANALYSIS AND IMPLEMENTATION OF OBJECT BASED STORAGE

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

____

Ravi Pendse, Committee Chair

____

M. Edwin Sawan, Committee Member

____

Krishna K. Krishnan, Committee Member

ACKNOWLEDGEMENTS

I would like to express my deep gratitude and sincere thanks to my academic advisor Dr. Ravi Pendse for his generous assistance and invaluable support. I would also like to thank my committee for taking the time to work with me in my research work. I appreciate their support.

I am thankful to Mr. Amarnath Jasti without whose knowledge and assistance this study would not have been successful. Sincere gratitude is also extended to the people who serve in the Department of Electrical Engineering office and the technical leads in Cisco TAC (Technical Assistance Center) at Wichita State University for extending their support and encouragement all through my Master’s Degree.

I would also like to thank my family members, friends and colleagues whose names have not been personally mentioned in making my thesis a success. It would not have been possible without their tremendous support and encouragement.

ABSTRACT

iSCSI (SCSI over IP) is used to implement storage area network (SAN) technologies (ex SCSI) over TCP/IP. In traditional implementations, iSCSI has been implemented using the block level storage. This paper emphasizes on the implementation of iSCSI based on Object-storage instead of block level storage by which the speed and security can be improvised, there by enhancing the performance. This research work also includes implementing the Object-storage based iSCSI, analyzing T10-OSD standard command set. In most of the operating systems, automatic TCP window tuning is implemented, but these available tcp window tuning methods are not optimal for all the available protocols In this paper, using the iSCSI protocol, the author made a careful study to figure out which iSCSI parameters effect the TCP window, and also determines the mathematical model for achieving the optimal TCP window size for iSCSI protocol.

TABLE OF CONTENTS

CHAPTER **PAGE**

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

Tables Page

LIST OF ACRONYMS

AHS	Additional Header Segment
BHS	Basic Header Segment
CDB	Command Descriptor Block
CHAP	Challenge Handshake Authentication Protocol
CPU	Central Processing Unit
EUI	Enterprise Unique Identifier
ExpCmdSN	Expected Command Sequence Number
FC	Fiber Channel
FFP	Full Featured Phase
FQDN	Fully Qualified Domain Name
IO	Input and Output
IP	Internet Protocol
IQN	iSCSI qualified name
iSCSI	Internet Small Computer System Interface
ISID	Initiator Session Identification
iSNS	internet Storage Naming Service
ITT	Initiator Task Tag
LBA	Logical Block Address
LU	Logical Unit
MaxCmdSN	Maximum Command Sequence Number
MTU	Maximum Transmission Unit
NAS	Network Area Storage

LIST OF ACRONYMS (Continued)

OSD	Object based Storage
OSI	Open Systems Interconnection
PDU	Protocol Data Unit
SAN	Storage Area Network
SAM	SCSI Architectural Model
SCSI	Small Computer System Interface
StatSN	Status Sequence Number
SLP	Service Location Protocol
SRP	Secure Remote Password
TCP	Transmission Control Protocol
TPGT	Target Portal Group Tag
TSIH	Target Session Identification Handle
UA	User Agent

CHAPTER 1

INTRODUCTION

The existing Storage Area Network (SAN) is a topology which interconnects storage area network over the internet. Before the SAN era Network Attached Storage (NAS) was the widely sharing mechanism. Storage area network was introduced in order to overcome certain drawbacks of the network attached storage like overhead on the network etc,. The current existing protocols that are used for implementing the storage area network topologies are fiber channel (FC) and iSCSI (Internet Small Computer System Interface). Prior to iSCSI protocol fiber channel was widely used protocol in storage area networks. After iSCSI was introduced, it has been widely adopted and used, because of the minimal implementation and maintenance costs [9]. However this paper does not discuss about fiber channel protocol.

1.1 introduction to iSCSI:

Internet Small Computer System Interface also abbreviated as iSCSI is used to transfer storage protocols like SCSI (small computer system interface) and FC (fiber channel) over existing IP cloud [8]. iSCSI encapsulates storage protocol's CDB (command descriptor blocks) in an IP packet and sends it over to the other end. As iSCSI is implemented over the existing IP cloud, it does not require a new network infrastructure [9], which helps it to maintain a minimum implementation and maintenance costs. Figure 1.1 depicts the architectural model of the iSCSI protocol.

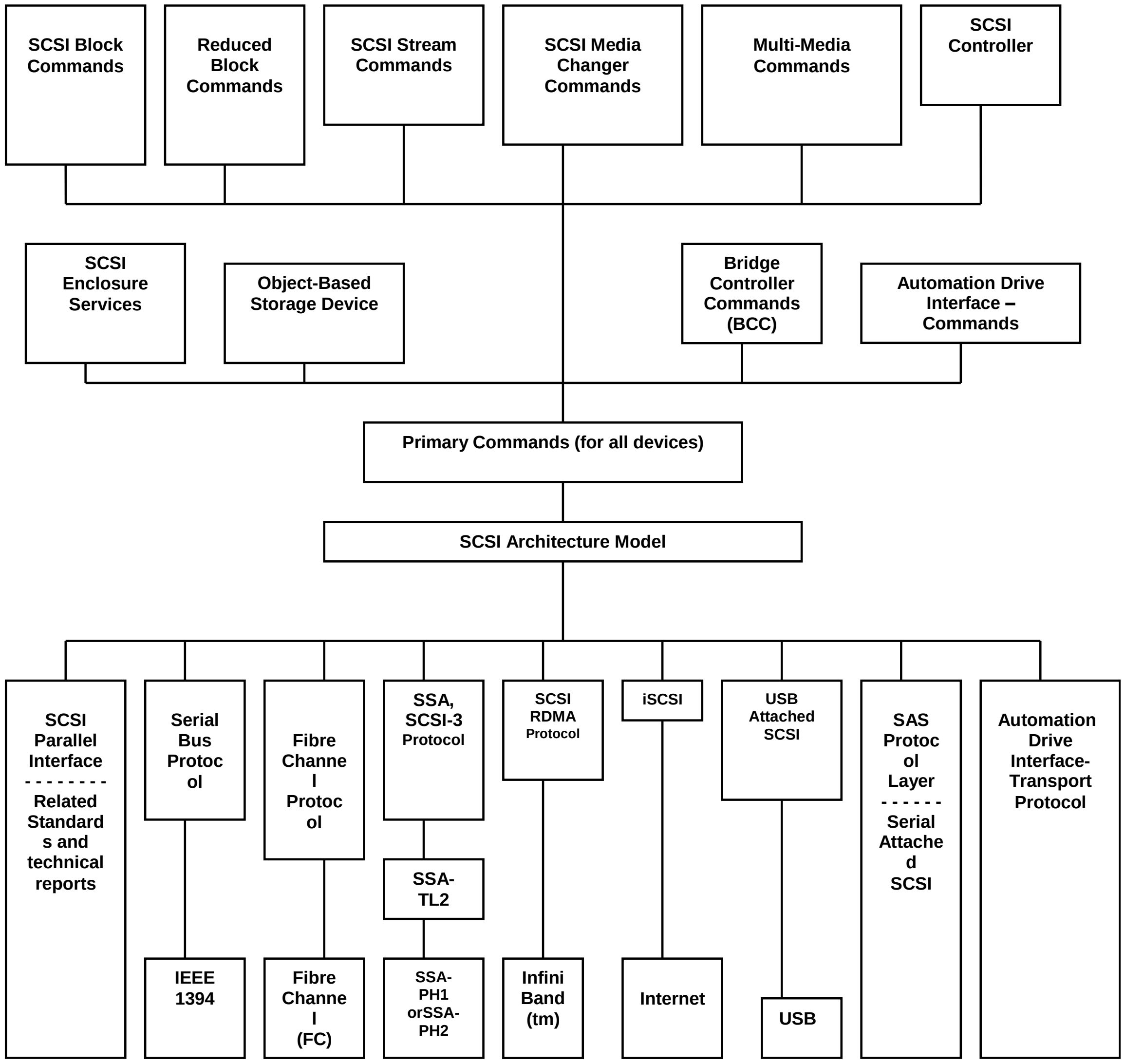


Figure 1.1 : iSCSI Architectural Model

iSCSI protocol is a medium which allows to transmission of different storage protocols, In SAM -3 (SCSI Architectural Model -3) most of the existing storage technologies are compatible with iSCSI.

Although wide varieties of storage techniques are available, which are compatible with the iSCSI, majority of the iSCSI implementations are based on the block level storage.

The performance of the iSCSI implementations can be enhanced further with the use of Object level storage devices instead of block level storage.

1.2 Introduction to Object based Storage:

Object based storage (OSD) is an upcoming technology which is still under development. The device in which the data is accessed in terms of objects and implements a certain standard for organizing the information is called an Object Storage Device. In such devices, 'Object' is defined as an identifier specified by unique set of bytes contained in an OSD. This OSD (Object based storage devices) logical unit (LU) places and assigns the Objects on the media [12]. Metadata is transmitted between layers and is related directly to each data object. Storage device files and the records that do not have long abstractions utilize Metadata, which supports better usage of storage devices [12]. At device level, the storage objects are secured as they are associated with metadata.

1.2.1 Advantages of Object based Storage over block level storage:

As the entire file management tasks are performed by the storage device itself, a unified storage interface mechanism can be achieved, which is compatible with any manufacturer’s devices without having the restriction of proprietary locking and this kind of implementation also off loads a considerable amount of CPU utilization at the initiator end [16].

In object based storage, every object can be assigned with a set of attributes, hence the user will have more control over the data, also the user can set the access rights to each object for higher level of security [12]. It also has the advantage of non-complex data allocation mechanism.

1.3 Problem Description:

As discussed earlier object based storage technology off loads a considerable amount of CPU utilization at the initiator side, there by reducing the processing time taken for completion of a given task, which results in the enhancement of the over all performance. But the delays introduced by the network residing between the initiator and the target are still present. Hence there is a scope to further improvement in the performance by tuning certain network parameters. In this research work, the author has considered the TCP (Transmission Control Protocol) window parameter in order to attain the optimal performance. The author has also made an attempt to come up with a list of iSCSI parameters which may have an impact on the TCP window and also to find out a mathematical model for determining the optimal TCP window.

1.4 Significance of this Thesis:

Currently there are very few implementations being performed on object based iSCSI. Therefore as a part of this research work, the author has implemented object based storage technology, in order to study and analyze the improvement in the performance of object based storage over the block level iSCSI and has also compared both object based storage and block level storage. The result of this study also focuses on the accomplishment of optimal TCP window size with the help of mathematical model.

1.5 Thesis Organization:

Going further, Chapter 2 introduces the iSCSI protocol and also gives an in depth idea on the error recovery mechanisms implemented in the iSCSI protocol

Chapter 3 introduces the Object based storage and also explains the various attributes and the basic command descriptor block (CDB) fields.

Chapter 4 gives an idea on the literature review that has been evaluated for this research study. Chapter 5 explains the test bed configurations, results and analysis andChapter 6 concludes with the result of this study.

CHAPTER 2

INTRODUCTION TO iSCSI

2.1 Introduction to iSCSI Protocol:

iSCSI (internet Small Computer System Interface) is a transport media for storage area protocols like SCSI over IP network. In the OSI model [8], iSCSI lies above the transport layer. Hence, the underneath layer (i.e. TCP/IP layer) will take care of the end to end connectivity and the reliability of the data [15]. An iSCSI entity comprises of an initiator and target, in generic aspect this can be considered as a client-server implementation.

When an application sends a READ/WRITE request to the SCSI (Small Computer System Interface) layer, the SCSI layer will generates an appropriate CDB (Command Descriptor Block) for the request and sends that to the iSCSI layer, where the CDB is encapsulated in to the iSCSI header, and forms an iSCSI PDU. Then this PDU (Protocol Data Unit) will be transferred to the other end over an iSCSI Session, which consists a TCP (Transmission Control Protocol) connection, There can be more that one TCP connection per an iSCSI session between an iSCSI initiator and a Target (Current implementations can have up to 4 TCP connections).

Figure 2.1 depicts the data flow using iSCSI protocols with reference to the OSI (Open Systems Interconnection) model.

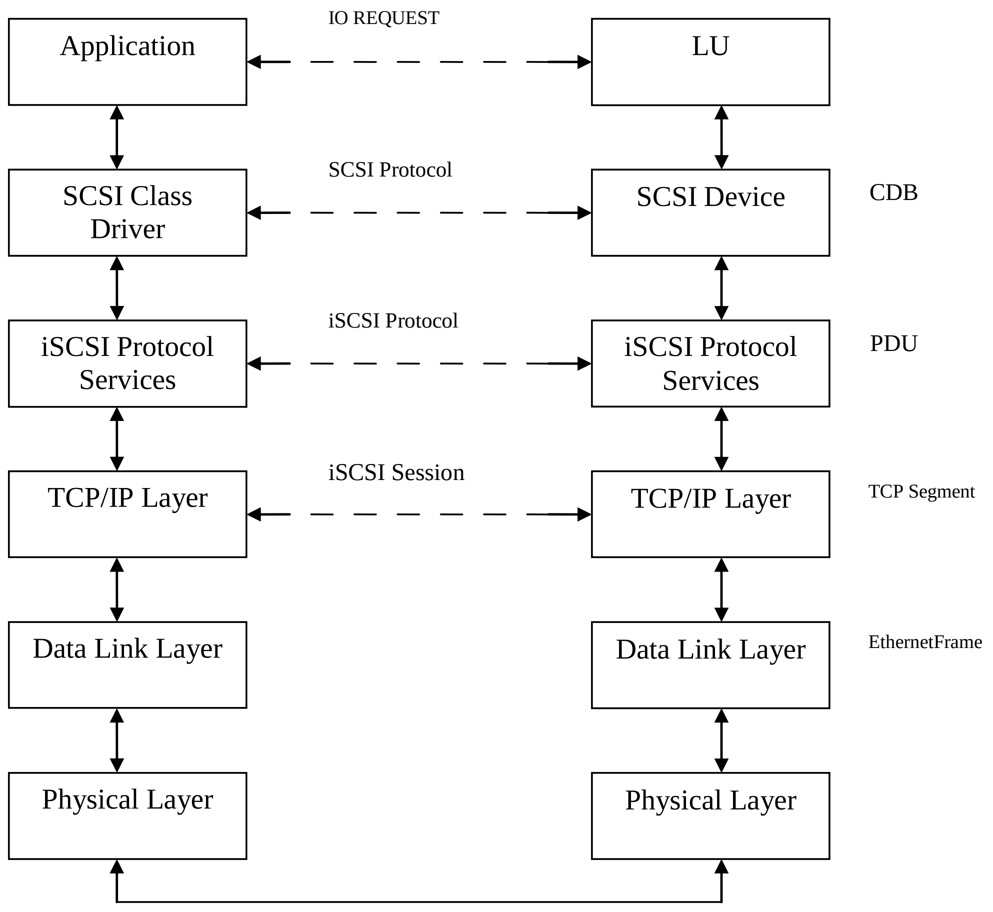


Figure 2.1:iSCSI in OSI model

2.2 Naming AND Addressing:

Naming standards for iSCSI entities are designed in such a way that, theoretically no two iSCSI devices have same iSCSI name [8], this eases the burden for the administrators. iSCSI naming convention is designed to be compatible with the Fiber channel and Serial Attached SCSI technologies. iSCSI devices can have any one of the following two naming conventions.

2.2.1 EUI (Enterprise Unique Identifier):

This naming convention is acquired from an IEEE standard Extended Unique Identifier (EUI). This is a 64 bit name, using which an iSCSI device can be uniquely identified. EUI consists of the following two fields.

OUI (Organization Unique Identifier): This is the most significant 24 bits. The least 48 bits can be chosen by the manufacturer. OUI is the ID which is assigned to the manufacturer, and the least 48 bits can be of manufacturer's choice. An iSCSI device name, which uses EUI standard, starts with eui. Followed by the 64 bit iSCSI name [8,10]. The following iSCSI name gives an example for the EUI standard iSCSI name.

Example: eui.anrc20080601anrc

From the above example anrc20 is the identifier which is assigned to the manufacturer, and the remaining part is the name which is assigned by the manufacturer.

2.2.2 iqn (iSCSI qualified names):

The iqn name starts with iqn. And this type of naming convention consists of more set of policies. The following iSCSI name gives an example of iqn naming convention.

Example: iqn.2008-06.com.anrc.wichita.edu:osd

From the above example, 2008-06 represents the device manufacturing year and the month, followed by FQDN (fully qualified domain name) [8, 9, 10], which is the reversed domain name of the manufacturer. The name after the (:) symbol is called iSCSI string, using which duplicate iSCSI names can be avoided. And the (:) symbol is used as the demarcating symbol, which differentiates the FQDN and the iSCSI string.

2.3 iSCSI Discovery Process:

iSCSI nodes undergo discovery process to find accessible iSCSI peers. This prevents if any unauthorized initiators to login to the target. iSCSI discovery process can be done in below two techniques:

I. Static Discovery Process

II. Dynamic Discovery Process

In static discovery process access permissions for iSCSI targets are configured manually, where as dynamic discovery process dynamically learns the mapping table. These two can be further categorized as follows:

I. Administrative Specifications

II. Using SendTargets

III. Using SLP

IV. Using iSNS

2.3.1 Administrative Specifications:

This is the basic discovery mechanism used in iSCSI. This technique involves manual configuration, administrator has to manually long in to iSCSI nodes and set permissions for each and every device. This technique can be used in small implementations. This will be a big hassle for administrators in complex iSCSI implementations.

2.3.2 Using SendTargets:

In this technique the initiator initiates a discovery session with the target device, and the target will respond to the initiator [8], if it is allowed to establish a connection with the initiator. Then the initiator sends a SendTargets command to the target, to which the target has to respond with the list of target nodes. In this the administrator has to manually configure the initiators with the list of targets which can be accessed.

This type of technique can also be implemented in small iSCSI implementations, because of the manual configurations that are required at the initiator end.

2.3.3 Using SLP:

SLP (Service Location Protocol) is used to locate the target nodes. This implementation requires the following three components.

I. User Agent

II. Service Agent

III. Directory Agent

1. User Agent also known as **UA,** which is a software component and implemented at the initiator end. Before sending a login request to a target the initiator sends a

request to the service agent in order to get the information regarding the target devices. Initiator can request for access only if it gets authorization from Service agent.

2. SA (Service Agent) contains the capabilities and permissions for that particular iSCSI target device in the topology. The SA advertises the target device capabilities either to the Directory Agent or directly to the User Agent, using which the administrator can restrict the access to the target devices.
3. DA (Directory Agent) which stores the capabilities of all the iSCSI target devices, and gives the information to the User Agents when requested. In this kind of implementations, the administrator has to configure the DHCP address, and also he has to configure the Service Agent access permissions. This can be used in complex iSCSI deployments.

2.3.4 Using iSNS:

This is the optimal solution for the iSCSI discovery process [8, 9]; this is also called as Internet Storage Name Service. This technique allows all the targets to register with the iSNS (internet Storage Name Service) server. Each time when the initiator wants to establish a connection, it has to go through the iSNS server, which checks the permissions set for the given initiator and target, and this will advertise the same to the initiator.

2.4 Session Establishment:

iSCSI session establishment process includes the TCP/IP connection, verifying the authentication, and also exchanging the operational parameters [10]. One or more TCP/IP connections between an initiator and target pair are called as iSCSI session [8]. Each TCP connection can be established based on its own operational parameters.

The Session Establishment can be of two types:

I. Normal

II. Discovery

Normal Login operation and Discovery Login operation are similar to each other; the only difference in the Discovery login operation is the target device information is not included in the login request PDU [8].

Figure 2.2 depicts different states in the login phase.

iSCSI session establishment process starts with the TCP/IP connection establishment, followed by iSCSI login phase, during which the iSCSI entities exchanges the operational parameters.

The iSCSI login phase consists of three phases [8, 9]: Security Negotiation Phase, Login Operational Negotiation Phase and Full Featured Phase.

Normal iSCSI traffic cannot be transferred unless both iSCSI target and initiators are in full featured phase.

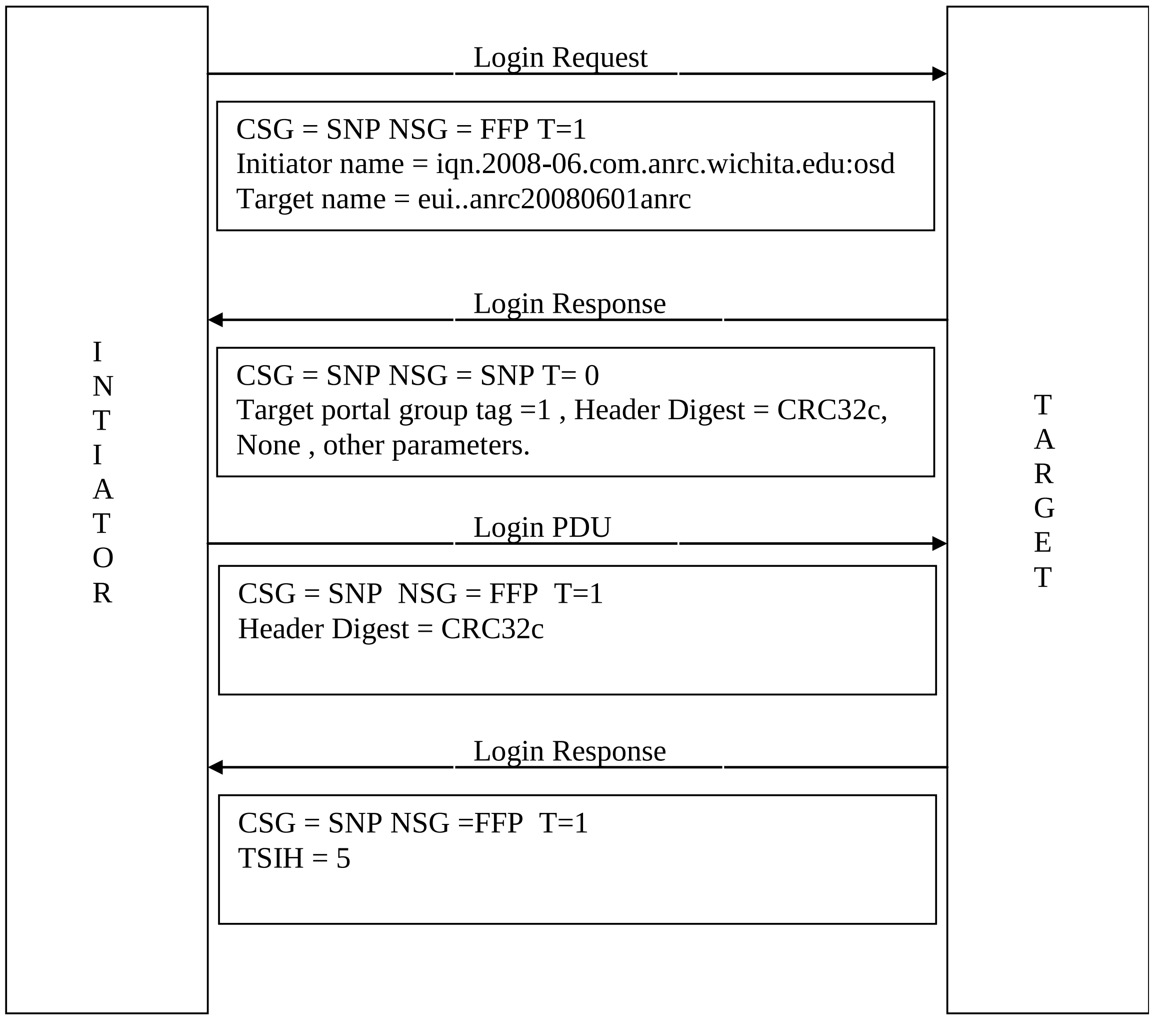


Figure2.2:iSCSI login Phase

2.4.1 Security Negotiation Phase:

In this phase both iSCSI initiator and target authenticates each other, this is the very first phase in the login operation [10]. The authentication has to be verified in order to proceed further with the login phase, if this phase fails the connection between the initiator and target will not occur.

iSCSI implementations support the following security mechanisms.

I. CHAP - Challenge Handshake Authentication Protocol

II. IPSec – IP Security

III. KRB5 - Kerberos V5

IV. SPKM1 - Simple Public-Key GSS-API Mechanism

V. SPKM2 - Simple Public-Key GSS-API Mechanism

VI. SRP - Secure Remote Password

VII. None

These authentication methods can be exchanged using the key=value pair *authmethod* [8, 10]. Once both iSCSI target and initiator decides on a security mechanism, the security mechanism specific authentication keys will be exchanged. Both the iSCSI devices has to negotiate on a security mechanism before proceeding further. By default all the iSCSI devices support none, which means no security mechanism is required.

2.4.2 Login Operation Negotiation Phase:

During this phase both the initiator and target exchanges and agrees on required operational parameters. These parameters are exchanged as key=value pair.

Example: FirstBurstLength=value

From the above example FirstBurstLength is the operational parameter, where as value indicates the max supported FirstBurstLength for that particular device.

Below are the different Operational parameters that are exchanged during this phase.

Header Digest: Both initiator and target decides on whether to use CRC-32C header digest mechanism or none. Default value for this is none. This can be used to find errors in PDU headers. In order recovery Header digest errors, the iSCSI initiator and target both should support Error Recovery level 1 or Error recovery level 2.

FirstBurstLength: This key=value pair decides how much data can be sent to the target with the command, which means the amount of data that can be sent to the target without any acknowledgement, this is also called as unsolicited data. This parameter must be less than are equal to the MaxBurstLength.

Error Recovery Level: Using this key=value pair both initiator and target chooses an error recovery level among the three available recovery levels. All the iSCSI implementation has to support one of the three error recovery levels. Default value for this is Error Recovery Level 0.

DefaultTime2Wait: This key=value pair decides, how much time the initiator has to wait before re issuing the failed commands. This time allows the target to clean up the existing buffers, by default this value is set to 2.

Immediate Data: This tells whether the immediate data is supported by both the ends, default value for this parameter is YES [8].

Initiator Name: This parameter is sent by the initiator during the first login request PDU. This can be either iqn name or the eui name.

MaxBurstLength: This is the amount of data that an iSCSI device can transfer with out waiting for an acknowledgement. By default this value is set to 256K bytes.

SendTargets: This parameter tells whether the iSCSI login phase is a Normal login phase or a Discovery login phase. A normal login phase has the SendTarget key=value pair with the value of the target node name. If it is a Discovery Login Phase, the Value for this parameter will be ALL. And also the initiator has to send the SessionType=Discovery key=value pair.

2.4.3 Full Featured Phase (FFP):

In order to transfer iSCSI data between any two iSCSI initiator and targets, both the ends has to be in full featured phase, which means they have to agree on some security mechanism, and have to negotiate all the required operational parameters.

2.5 iSCSI PDU (Protocol Data Unit) structure:

Protocol Data Units are used for communication between iSCSI nodes (target and initiator). This PDU contains iSCSI parameters and higher level CDB (command descriptor block), which is built by SCSI layer. PDU will be encapsulated with TCP header, which carries the end-to-end network information.

Figure 2.3 depicts the general iSCSI PDU (protocol Data Unit)

2.6 Session Management:

iSCSI Session starts with iSCSI login phase [9], during which iSCSI initiator sends CID (connection ID) and Initiator Session ID (ISID) values to the initiator. Along with the Connection ID the initiator also sends ISID (Initiator session ID), which is a 48-bit field, it consists of vendor ID (48-bit) and a vendor assigned ID. ISID field can be incorporated into the login PDU in three different ways.

If the vendor type value is 0, Organizational Unique Identifier will be placed in the lower 22 bits, followed by unique identifier. If the vendor type is 1, first 6 bits will be zeros, next three bytes will be IANA EN (Enterprise number). If the vendor type is 2, again the first 6 bits will be zeros, followed by 3 bytes of random numbers, and 2 bytes of unique qualifier. And the vendor type value 3 is reserved.

Byte	0	1	2	3
00 to 47	Basic Header Segment (BHS)			
48 to xx	Additional Header Segment (AHS) (optional)			
xx + 1 to yy	More AHS (optional)			
yy + 1 to yy + 4	Header Digest (optional)			
yy + 5 to zz	Data Segment (optional)			
zz + 1 to zz + 4	Data Digest (optional)			

Figure 2.3 : iSCSI PDU

ISID and the iSCSI initiator node name together form the iSCSI initiator port. This makes it an unique name in the world.

Figure 2.4 shows the general Additional field header for iSCSI.

Byte	0	1	2	3
Bit	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
0 to 3	.|i|. Opcode	Opcode Specific field		
4 to 7	Total AHS Length	Data Segment length		
8 to 15	Logical Unit number (LUN)			
16 to 19	Initiator Task Tag (ITT)			
20 to 31	Opcode Specific Field			
32 to 47	Command Descriptor Block (CDB)			

Figure 2.4 : Additional Header Segment Field

Byte in PDU	0	1	2	3
Bit	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
8 to 13	ISID			

Figure 2.5: Initiator Session ID

2.7 Connection Establishment:

In order to establish an iSCSI session, the initiator sends a login request PDU to the target, which includes Connection ID, Initiator Session ID, iSCSI initiator name, iSCSI target name etc,. Target responds with a login response PDU. As explained in [2.4.3], a successful login phase ends with a reply from the target, which includes a non-zero TSIH (target session identifier handler) value. Each iSCSI session between an iSCSI initiator and a target is represented by an iSCSI Session ID (SSID), which is unique for a initiator and target pair [8]. An iSCSI session can contain any no of connections, if an initiator is trying to establish a new connection with a SSID same as previously established connection and with a non-zero TSIH value, target assumes that the initiator is trying to establish a new connection in the same iSCSI session.

Initiator can transfer the commands in any one of the existing connection in the current session. In case of error occurrence, if Error Recovery Level 2 is implemented, the initiator uses the CID (Connection ID) to re-establish the connection [23], when ever the target sees a login request with an already existing CID, target terminates the existing connection, and tries to establish a new connection.

2.8 Sequencing:

iSCSI implementation has incorporated sequence numbers, in order to keep track of the errors in the iSCSI PDU's. All the sequence numbers start from zero and increments with the commands [9, 20]. Hence the iSCSI end nodes can easily find the errors in the iSCSI PDU's. By incorporating the sequence numbers in the PDU's, iSCSI does not require external acknowledgements or sync bits. Error Recovery mechanisms will be discussed in the following section. Below are some of the sequence numbers,

I. StatSN

II. ExpStatSN

III. CmdSN

IV. ExpCmdSN

V. MaxCmdSN

VI. DataSN

VII. ExpDataSN

StatSN: StatSN (Status Sequence Number) used to keep track of the status PDU's that are generated in a particular session. This sequence number is used to find any holes in status messages. Scope of the StatSN is a session. If iSCSI initiator finds a hole in StatSN, it sends a SNACK PDU to the target with the missing StatSN. StatSN and

ExpStatSN together make sure the delivery of all the status messages. StatSN is always less that the ExpStatSN.

ExpStatSN: ExpStatSN (Expected Status Sequence Number) is a counter which represents the expected StatSN (status sequence number) of the following command. This is used to detect the command PDU drops. ExpStatSN is greater than the StatSN.

CmdSN: Command sequence (CmdSN) number is associated with a command, which counts the number of commands that the initiator has sent. This counter is valid throughout the session. As an iSCSI session may have multiple TCP connections [20], and the initiator can choose any of the existing TCP connection to send the commands, Hence at the other target the commands may receive out of order. The target uses CmdSN to send the commands to the SCSI layer in order. This is the only counter which is valid through out a session.

ExpCmdSN: ExpCmdSN (Expected Command Sequence number) counter represents the expected CmdSN for the next command. Which is also helps in identifying the command PDU drops. iSCSI Response PDU contains the existing Status Sequence number and the ExpCmdSN.

MaxCmdSN: MaxCmdSN (maximum command sequence number) counter tells the maximum number of commands the target can handle at a particular time, this is used to implement the windowing technique in the iSCSI implementation. If the MaxCmdSN plus 1 is equal to the ExpCmdSN, the window is closed, which means the target can not accept new commands.

In addition to the command sequence numbers, which are used to keep track of the command numbering, iSCSI implementations do have data sequence numbers, which are used to count the number of the data PDU. Every Data PDU contains DataSN and ExpDataSN.

DataSN: DataSN (Data sequence number) counter increments with the data PDU's, which is included in every data PDU. If there is a hole in the DataSN, iSCSI end node knows that there is a lost Data PDU.

ExpDataSN: ExpDataSN (Expected Data Sequence number) counter contains the expected DataSN of the next Data PDU.

2.9 iSCSI Data Transfer:

In order to transfer the actual data the iSCSI entities have to be in full featured phase [20]. Once the target and initiator are in the full featured phase SCSI layer at the initiator issues commands to Read or Write the data. Further these SCSI commands are encapsulated in iSCSI header at iSCSI layer. TCP layer takes care of the end-to-end connectivity. TCP layer resends a packet, if it finds any packet drops. In multiple TCP connections scenario iSCSI data can be transferred on different TCP connections, and the initiator uses status sequence numbers to put the data together.

2.10 iSCSI Error Recovery Mechanisms:

In this section we will discuss different error recovery level mechanisms implemented in iSCSI. iSCSI has three error recovery techniques, among these three levels of error recovery techniques, target and initiator negotiate on which technique to use during the login operational phase.

iSCSI error recovery techniques are classified as below.

I. Error recovery level 0

II. Error recovery level 1

III. Error recovery level 2

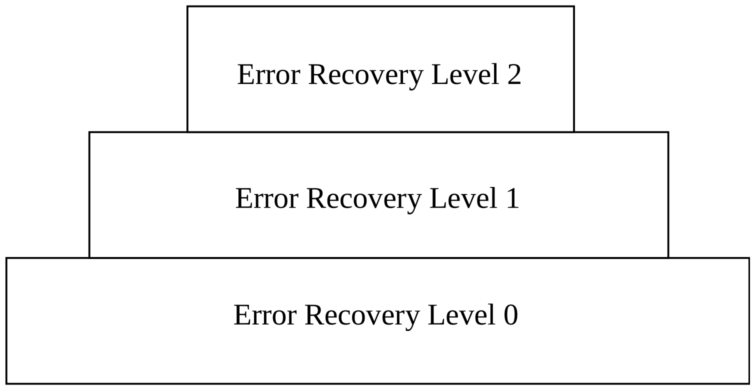


Figure 2.6 : iSCSI Error Recovery Levels 1

Error recovery level 0 recovers failures by restarting the session. Error recovery level 1 recovers by explicitly requesting the initiator to retransmit the data, Error recovery level 2 starts a new TCP session and tries to move all the existing connections to the new session. These three Levels of error recovery can be depicted as a hierarchical implementation,

Error Recovery Level 0 is the basic technique which just restarts the session for all the failures, and every iSCSI implementation has to support this Error Recovery mechanism. Recovery Level 1 can recover certain failures without restarting the session, where as Recovery Level 2 implements the optimal way of implementing the error recovery. And iSCSI implementations, which support Error Recovery techniques greater than level 0, have to support the error recovery levels below also. Therefore iSCSI implementations, which support Error Recovery level 2, has to support Error Recovery level 1 and Error Recovery level 0, and the implementations which support Recovery level 1 has to support Recovery level 0 also. During the login operational phase, if the target supports Error Recovery level 1(which means it also supports Recovery level 0), and the initiator supports Error Recovery level 0, both the initiator and target will use the Recovery level 0 for that particular iSCSI session, as Recovery level 0 is supported by the target and initiator also.

2.10.1 Error Recovery Level 0:

Error Recovery level 0 is implemented only when the iSCSI implementation is failed to implement the other two error recovery mechanisms. If this recovery mechanism is implemented, whenever an error occurs, iSCSI initiator aborts all the tasks and starts a new session [20], then tries to retransmit the data, and all the SCSI layer tasks which are being processed will get a simulated error messages from the iSCSI layer. And at the initiator side, once all the TCP connections are closed and everything is cleaned up, the iSCSI initiator has to again request for a login as the initiator did the first time.

The initiator has the privilege to decide whether to tear down the existing TCP connections. If an error occurs the initiator sends a logout request to the target. Once the target receives the logout PDU, it starts simulating the existing processes, and sends appropriate error messages to the SCSI layer, then clears all the existing tasks and sends a logout response to the initiator, then the initiator starts tearing down all the existing TCP connections, and clears all the tasks.

At the initiator end, once the logout response is received, initiator sends a pseudo error messages to the SCSI layer, and then starts cleaning up all the existing tasks. And the initiator will try to establish a new session as it did the first time. Number of connections can be more or fewer.

In some situation, the initiator has to drop all the connections without sending a logout PDU (Protocol Data Unit) to the target. If this happens, the initiator has to wait a for default time period specified by **DefaultTime2Wait** parameter, this key value pair is exchanged during the login operational phase. This time allows both the initiator and target to send the simulated errors to the respective SCSI layers, and also to clean up all the existing tasks.

Error recovery level 0 may also implement another technique for aborting and re establishing the TCP connections. In which the initiator will send a login PDU with a zero TSIH (Target Session Identifying Handle) and ISID (Initiator Session ID) of the connection that needs to re established, the initiator node name target node name and the target portal group remain the same, this type of login PDU also lets the target and initiator to send simulated outputs to the corresponding SCSI layer and clean up all the existing tasks.

During the process of aborting and restarting the existing TCP connections, the initiator node name, initiator session id (ISID), the target node name, and the target portal group tag (TPGT) remain the same.

2.10.2 Error Recovery Level 1:

As it is mentioned, all the iSCSI implementations which use Error Recovery Level 1 needs to support Error recovery level 0 also. Hence Error Recovery Level 1 can restart the session if an error occurs, other than this it can also recover most CRC errors with out restarting the session.

Error Recovery level 1 can handle the following errors:

I. Header digest error

    a. At the target side

    b. At the initiator side

II. Data digest error

If a header digest error occurs, the PDU will be thrown away silently, and the other end will detect the packet loss (as the iSCSI devices keep track of the sequence numbers), and retransmits the packet. If a Data digest error occurs at the initiator side, initiator responds with a SNACK PDU (selective negative acknowledgement PDU), which let the target know about the error. If a Data digest error occurs at the target side, the target will send a Reject PDU to the initiator.

2.10.2.1 Header Digest Error at the Initiator Side:

Assuming the both the initiator and target are implementing some sort of FIM (Fixed Interval Marking), which places the markers in the TCP stream, this helps in identifying the borders for the iSCSI header. If no marking technique is used, in case of the header digest error, it is hard to figure out the exact boundaries for the iSCSI PDU. In this kind of scenarios, we may not determine the exact error and the iSCSI implementation has to lower the Error recovery mechanism to level 0, and recover the error by restarting the session.

At the initiator side, if an error occurs in the command or request PDU, this can be detected at the initiator side by checking the expected command sequence number (ExpCmdSN), the initiator checks the ExpCmdSN (Expected command sequence number) of the response PDU from the target, whether it is in synchronization with the ExpCmdSN of the command PDU. This kind of error detection is possible, because in the iSCSI implementations, each command or request PDU has a command sequence number (CmdSN) and an ExpCmdSN, which is the expected command sequence number for the next command or request PDU. If the CmdSN number is greater than the ExpCmdSN in the last PDU, it is obvious that there is a drop or an error occurred.

If the initiator finds a difference in the ExpCmdSN and the CmdSN, the initiator tries to resend the PDU with the missing CmdSN. If the initiator receives the response from the target after retransmitting the PDU, the initiator simply does not do any thing, and at the target end, it is a rule that if the target receives any duplicate PDUs, the target has to silently drop the PDU.

If there is any Data-out PDU missing, this can be detected at the target end, by checking the DataSN (data sequence number). If there is any missing DataSN, target assumes still some more information is yet to come, then the target sends an explicit R2T (Ready to transfer) for the missing DataSN, then the initiator retransmits the data.

In some cases, the target assumes that initiator has received all the Data-out PDUs, and sends a final response PDU to the initiator, but if the initiator is still waiting for some Data-out PDUs, as it receives the final Data-out PDU, by using the DataSN and ExpDataSN (expected data sequence number), initiator will figure out the missing Data-out PDUs, and will explicitly request for the missing PDUs by using SNACK PDUs. And the initiator uses StatSN (Status sequence number) and ExpStatSN (expected status sequence number), to figure out if there is any error occurred to the status PDUs.

2.10.2.2 Header Digest Error at the Target Side:

At the target if any header digest error occurs, it simply drops the PDU, by using the ExpCmdSN the initiator will notice the hole in the PDUs sequence, and retransmits the missing PDUs.

If there is an error occurred in the header of the command PDU, the target discards the packet, and will not increase the ExpCmdSN (Expected Command Sequence Number), which indicates the initiator that there is an error occurred, and resends the packet. Same mechanism is implemented for Text Requests, Task Management requests, and logout requests.

And if there is a header digest error in the Data-out PDU, the target can keep track of the DataSN (Data Sequence Number) number, which tells the target if there is any error, then the target can explicitly request for the missing PDU by sending a R2T [8, 20]. If the last R2T is missing, initiator will not be able to detect the error; in this case the target has to resend the R2T, if it does not receive the data in time. And the initiator discards all the duplicate R2Ts.

The target can also send positive SNACK PDUs, to notify that a particular PDU has been successfully processed, if this positive SNACK PDU is missing, As the initiator does not wait for a reply ,there is no way the initiator can identify the error, for this kind of errors the target has to compensate at its own end.

2.10.2.3 Data Digest Error:

In Data Digest errors, there is an error in the data field, which means we can still extract all the information from the header. Hence this can be easily handled. Just by looking at the sequence numbers the iSCSI entity can determine where the error has occurred, and requests for retransmission.

If the initiator detects there is a data digest error in the PDU, initiator will simply looks at the sequence number and determines, which PDU has the error, sends a SNACK PDU for requesting the PDU retransmission, and drops the PDU with the data digest error. If the error is detected at the target side, the target sends a reject PDU with the error code as Data Digest Error, and discards the PDU. Then the traget again requests from the retransmission explicitly by sending a R2T.

In some implementations, all the digest errors are handled just by dropping the PDU. And in order to reduce the wait times, all the implementations are required to send NOP PDUs (No operation PDU), when any transaction has been finished, with this the iSCSI entities can immediately identify if there is an error in the last PDU.

2.10.3 Error Recovery Level 2:

Error recovery level 2 is the most optimal error recovery mechanism, this level of error recovery can recover the session and the outstanding tasks in the session as well. At the initiator end, if it detects an error due to the TCP connection failure, or by receiving an Asynchronous Message PDU, which indicates there is an error occurred in the TCP connection from the target. Asynchronous PDUs are generally sent to request for logging out the current connection.

Target or Initiator may detect an error, if there is any gap in the sequence numbers, or if the NOP PDU is not getting any reply. In iSCSI implementations that support Error recovery level 2, and if there are any other active TCP connections between the same I-T-L (initiator - target - LUN number) nexus, the initiator tries to send a logout PDU to the target, this logout PDUs have a reason code, which notifies the target that this connection has been removed for recovery. Then the target shuts down the TCP connections and prepares all the existing tasks to be able move to the new connections.

After a successful logout, the initiator can start a new connection or it can also use other existing connections in order to resume the failed tasks. In order to recover the failed tasks, the initiator sends couple of task management PDUs, which tells the target to move the old tasks to the new connection. This process has to move one task at a time, after moving all the tasks, they will resume as they did before. Immediate commands can not be recovered using the Error Recovery Level 2. If the initiator decides to use the existing connections, rather than starting a new connection, in order to have the load balancing, the moves the failed tasks to all the existing connections. If the initiator fails to send a logout PDU on other existing connections, or if there is only one existing connection in a session, the initiator opens a new connection, and sends a new login command directly without logging out, In this case a new TCP connection has been established, just to move the failed connections.

During the login operation phase, initiator and target exchange DefaultTime2Wait, and DefaultTime2Retain. DefaultTime2Wait is the default time the initiator has to wait before reconnecting. And the DefaultTime2Retain is the default time the initiator has to wait before re establishing the connection. These two key=value pairs are by default set to 3 seconds. If the target identifies that there is an error in the connection, then the target has to send an asynchronous PDU to the initiator, for requesting a logout.

CHAPTER 3

INTRODUCTION TO OBJECT BASED STORAGE

3.1 SCSI OSD implementation:

OSD based SCSI implementations have below five main components,

I. Object based Storage Device (OBSD)

II. Service delivery subsystem

III. Initiator

IV. Security Manager

V. Policy Manager

The figure 3.1 depicts the functional diagram of the object based storage SCSI implementation.

3.1.1 Object based Storage Devices (OBSD):

Object based storage devices (OBSD): are the repositories for the objects, and which are having any of the existing object based file system, hence these storage devices can internally map the block level data to the objects [2, 3]. These object based storage devices act as the targets for the SCSI implementation. This component may have couple of internal sub components, which includes a firmware which handles the OSD commands.

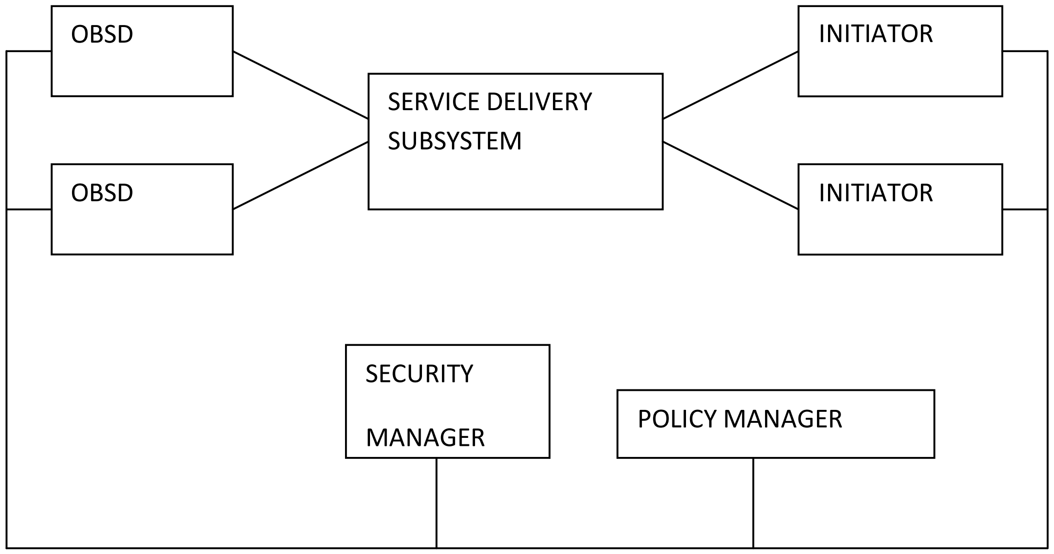


Figure 3.1:OSD based iSCSI configuration 1

3.1.2 Service delivery subsystem:

Service delivery subsystem has a great significance in OSD based SCSI configuration. The functionality of this component includes sending the Object based commands to the Object based storage devices [2, 3]. This might be a software component, which is residing on the storage devices, or a separate component. In most of the implementations this component is used as the mediator between the iSCSI initiator and the object based storage devices.

3.1.3 Initiator:

This is a software component which accesses the data on the object based storage devices via the service delivery subsystem [3]. This component has the same functionality as the block level iSCSI initiator component, except the OSD based initiator generated OSD based commands.

3.1.4 Security Manager:

This component is an optional component in the OSD based SCSI configurations, if configured this component will take care of the access permissions for the object based storage devices and the initiator devices. Using this component the user can define the access permissions for each and every object present in the object based storage, which pushes the level of security to a higher level.

3.1.5 Policy Manager:

Policy Manager is also an optional component like the security manger. This component will monitor the access constraints between the object based SCSI initiator and the object based storage device. Both Security Manager and Policy Manager Components use the service delivery subsystem to communicate between the OSD initiator and the object based storage devices [1, 2].

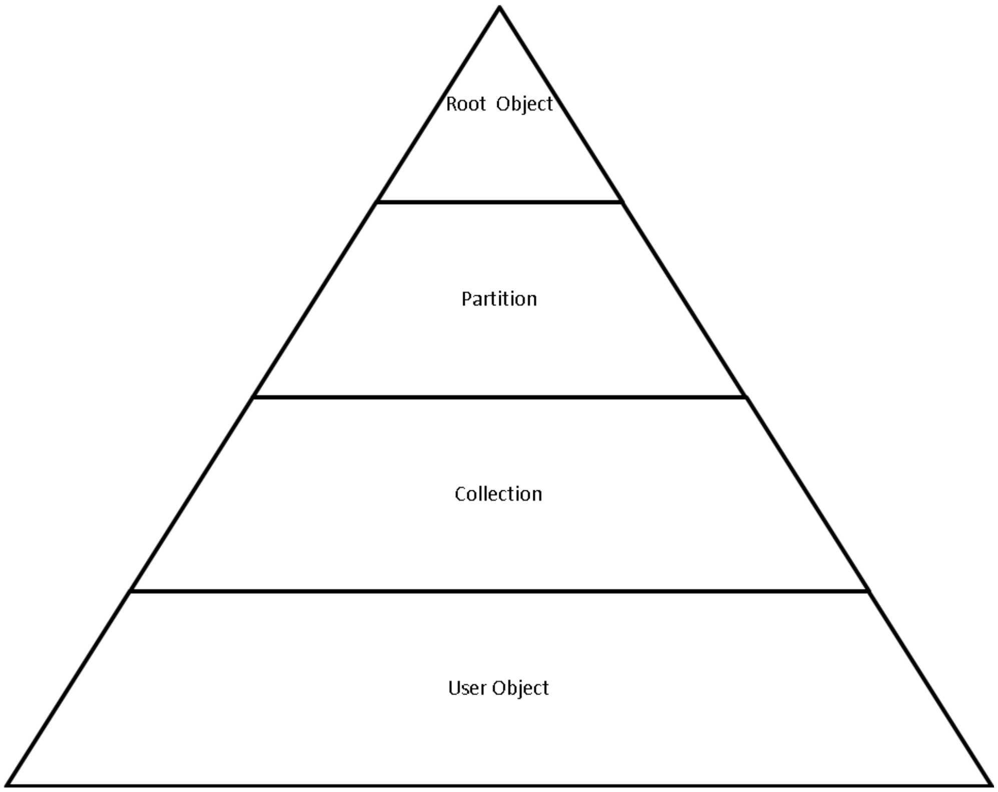


Figure 3.2 : Different types of Objects 1

3.2 Architecture of Object based Storage:

In object based storage devices data is saved in the form of objects, these objects are created by the logical unit (LU) in the object based storage device [7]. Objects can not be represented by the logical block Addresses (LBA). Each object is assigned with a unique ID [2], which is called as the object ID, this ID is used by the initiator to access the data instead of the logical block addresses (LBA).

Object based storage devices implements a hierarchical approach for storing the end user data on the disks. The figure 3.2 shows the hierarchy of different components in the object based storage devices.

3.2.1 Root Object:

Each Logical Unit can have only one Root Object, theoretically logical units itself is represented as the root object. Root object is assigned with global attributes, which are same for all the user objects present in a root object. This also maintains the list of partitions present in the root object. Root object can be addressed by setting the partition ID and user object ID values to zero. And all the READ/WRITE commands that are generated with the Root object address are noted as illegal operations, hence it does not contain any storage space for the user data.

3.2.2 Partition:

Partition contains all the user objects with same security attribute and same capacity quota etc., this can be created by the application client at the initiator side. Partitions may also contain Collection Objects; there is no limitation for the number of partitions that are existed in a root object [3]. Partitions can be represented using a Partition ID, and can be addressed by putting the User Object ID as zero. OSD based initiator can create a partition using the command CREATE PARTITION, and initiator can also request for list of all existing partitions using the LIST command. Partitions will inherit some of the default attributes from the root object attributes.

3.2.3 Collection:

Collection is a group of user objects which have similar attributes; this kind of approach is introduced in the object based storage devices for increasing the access speeds. There could be any number of collections present in a partition. Collection can be addressed by using the Collection ID and the list of User Object IDs. All the user objects, which are sharing the same collection ID, will inherit the default attribute values from the partition attributes. List of all the User Objects in a collection can be retrieved using the command LIST collection.

Collections can be classified as the following three collections.

I. Linked

II. Tracking

III. Spontaneous

Linked collections: Linked collection lists the entire user object IDs which are linked to each other, if an object in the linked collection is removed, reference for the deleted object will be removed from all the objects. By setting the Object attribute, an object can be added to the LINKED collections.

Tracking Collections: Tracking collection objects does not maintain any linkage to the user objects. The only purpose of these collection objects is to track the status of the command issued on an user objects.

Spontaneous Collections: All well known collections are known as spontaneous collections. Every time a spontaneous collection retrieved the membership is recomputed. Memory management hierarchy of Object storage device is well designed to make the management easy.

3.2.4 User Objects:

User Objects contains the actual end user data, which resides inside partitions. Each user object has its own attributes. Object can be of any size. In object based storage one file cannot be saved on to multiple user objects, hence the management will be minimal. User objects consist the actual user data, Attributes and User defined Attributes. Object attributes stores the metadata (date of creation, owner ect.,). User defined attributes can be used to define object based policies.

3.3 Object Attributes:

Object based storage allows assigning attributes and Meta data to all types of objects, attributes describes characteristics of an Object. Attributes can be retrieved by using the command GET ATTRIBUTES and SET ATTRIBUTES command stores the attributes [2, 3]. Attributes of same kind of objects are grouped and saved in the form of pages, and each attribute can be addressed using the attribute ID. Attributes with a non-zero value are called defined attributes, and attributes with a zero value are called non-defined attributes.

Retrieving Attributes Length: Each attribute is associated with the attribute length field, which is used to determine whether an attribute is a defined attribute or non-defined attribute. If the attribute length filed is a non-zero value, the associated attribute is called as defined attribute, if the attribute length field is zero, then the associated attribute is called as non-defined attribute [3].

Setting the Attribute Length: Attribute length for an attribute can be set using the set attribute field in the CDB (command descriptor block). Depending upon the attribute type (defined/non-defined) setting the attribute length field is explained below.

Defined Attributes: If the task is to set the attribute length of a defined attributes to set to zero, this changes a defined attribute to non-defined attribute. If the set attribute length field is setting a non-zero value to an attribute, this will simply changes the existing attribute length field of the attribute.

Non-Defined Attributes: To set the attribute length field of a non-defined attribute to a zero value, the client has to leave the attribute length field as it is, no errors will generate with this operation. In order to set the attribute length field of a non-defined attribute to a non-zero value, client has to replace the null value in the attribute length field with a non-zero value.

3.4 Command ordering for retrieving or setting an Attribute:

Application client can set or get the attributes by processing a separate command, or by incorporating the relevant get command with an existing task. Application client can a single command in order to set or get the attributes of a set of attributes, by using an appropriate collection list. If the application client requests for getting /setting an attribute, the order of the process completion will be done in the same order, how the objects are arranged.

Example: A Read/Write command may set/get an attribute.

If the application client incorporates the set / get functionality with a different command (i.e., other than commands like GET ATTRIBUTES, SET ATTRIBUTES etc,), the process flow will be in the following order.

Example: Both SET and GET attribute fields are present in a WRITE command.

If the application client issues a GET /SET ATTRIBUTE command in order to get / set the attributes, the command ordering will be as follows [2,3].

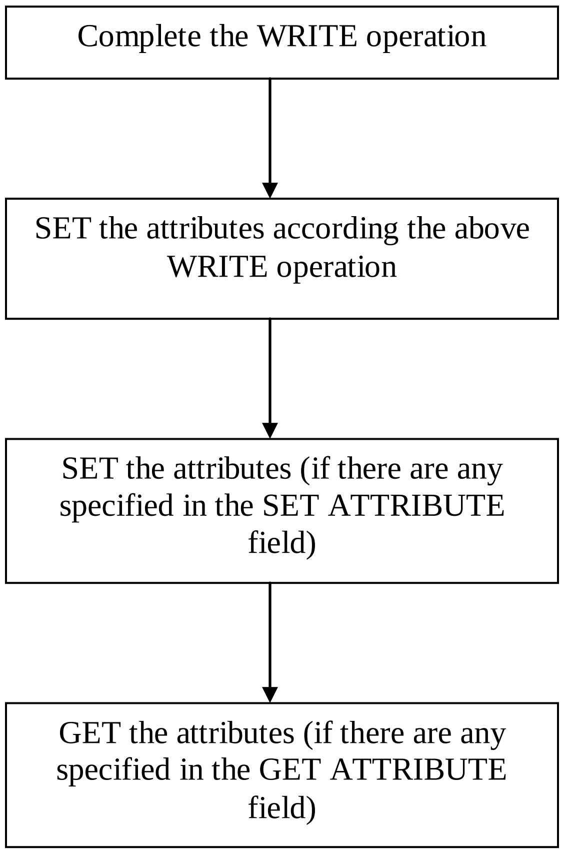

Complete the WRITE operation
SET the attributes according the above WRITE operation
SET the attributes (if there are any specified in the SET ATTRIBUTE field)
GET the attributes (if there are any specified in the GET ATTRIBUTE field)

Figure 3.3 : Command execution flow

1. Set the attributes if there are any changes made due to the command.

2. Get or Set the attributes according to the get or set fields in the command descriptor block in the GET or SET ATTRIBUTE commands respectively.
3. Process any Set or Get functions specified in the GET or SET commands respectively.

3.5 Pages:

Each page contains same type of attributes information. Pages are assigned with an unique page numbers ranging from 0 to FFFF FFFF. If FFFF FFFF page number is used to retrieve the page information, the device will send the information of all the pages that are existing in the logical unit (LU) [3].

3.6 Attributes:

Attributes are assigned with an unique attribute number in a page between 0 to FFFF FFFF. Attribute number 0h is used, if the client has to retrieve the page information. And FFFF FFFF attribute number can not be set to any attribute, if performed this will return an error.

3.7 OSD Command Descriptor Block:

OSD command descriptor block has a 10 byte header [2]. Following are the main fields in the command descriptor block.

I. OPERATION CODE

II. CONTROL

III. Additional CDB length

IV. Service Action

V. Service Action Specific fields

The figure 3.4 depicts the based command descriptor block of the object based storage.

OPCODE (Operation code) is the very first byte in the command descriptor block, this field contains the value of 7Fh. Second byte of the CDB is control. As the OSD command descriptor block has variable length fields, Additional CDB length field specifies the length of the additional fields in the command descriptor block.

Service Action field contains the actual command description [3], and the service action specific field contains the additional information that is required by the command which is mentioned in the service action field.

Example: If the Service action field contains a READ command, service action specific field contains all the information required in order to perform the READ operation like the object ID, required attributes etc.,

3.7.1 Basic Service Action Specific field:

Service Action Specific field contains the actual command related information in the command descriptor block [3]. Figure depicts the Basic Service Action Specific field. Following are some of the important fields in the service action specific field.

I. Allocation Length

II. Caching control bits

III. Capability

IV. CDB continuation length

V. Get and Set attributes parameters

VI. Immediate bit for TRACKING collections

VII. Isolation

VIII. Length

IX. Partition ID

X. Security Parameters

XI. Starting Byte address

Allocation Length filed: Allocation Length is sent by the application client, this informs the target how much space (in bytes) is available at the application client end for storing the incoming data. If this field is set to zero, this means there is no space available at the application client end for the data.

Caching control bits: From the figure the Disable Page out (DPO) and Force Unit Access (FUA) bits are used to control the cache [4]. The Disable Page Out bit is used to minimize the use of volatile cache. If the Disable Page Out bit is set to 1, the data transferred with this command will not be placed in the volatile cache.

Bit / Byte	7	6	5	4	3	2	1	0
1	OPCODE							
2	CONTROL							
3	Reserved							
4	Reserved							
5	Reserved							
6	Reserved							
7	Additional CDB Length							
8 to 9	Service Action							
10 to 235	Service Action specific fields							

Figure 3.4 : Command Descriptor Block

Capability: Capability field contains the information regarding the security mechanism implemented.

CDB continuation length: This field contains the size of the CDB continuation segment field.

GET/SET CDBFMT: Get and Set attributes CDB format (GET/SET CDBFMT) field tells the format to get or set operation in the command. Following are the different values of this field.

I. 00b – Reserved

II. 01b – Set one attribute using CDB fields

III. 10b – Set an attribute value and Get an attributes page

IV. 11b – Set and get attributes using lists

Following CDB fields show the important fields in a set attribute command.

Attribute Page: Page number of which the attribute needs to be set is represented by Attribute page field.

Bit Byte	7	6	5	4	3	2	1	0
48 to 51	ATTRIBUTE PAGE							
52 to 55	ATTRIBUTE NUMBER							
56 to 57	ATTRIBUTE LENGTH							
58 to 75	ATTRIBUTE VALUE							

Figure 3.5: Get or Set attribute field

Attribute Number: This field specifies the attribute number in a page

Both of the above fields can be any value between 0 to FFFF FFFF (can't be FFFF FFFF), if any of the above field is set to FFFF FFFF and error will occur [2].

Immediate bit for TRACKING collections: IMMED_TR (immediate bit for TRACKING collection) bit can be either 0 or 1. If this bit is set to 1, before performing the operation on all the objects in TRACKING collection, the application client will send a response with GOOD status, if this bit is set to 0, the application client has to wait till the operation has been performed on all the objects.

Isolation: This bit is used to inform the application client to perform the command in isolation. This field can be set to the following values [2, 3].

I. 1h – NONE

II. 2h – Strict isolation

III. 3h – Reserved

IV. 4h – Range isolation

V. 5h – Functional isolation

VI. 6h – Reserved

VII. 7h – Vendor Specific

Length: Length field is used to specify the maximum number of data can be transferred to the application client.

Partition ID: This field is used to specify the partition ID of which the task is intended to perform. If this field has a partition ID that is not a valid ID or an ID which does not exist in the logical unit, an error will occurred, and a check condition will be sent back to the initiator.

Security Parameters: Security parameters field contains the information related to the security mechanism implemented. Figure shows the basic security fields in the command descriptor block.

Bit / Byte	7	6	5	4	3	2	1	0
184 to 215	Request Integrity Check Value							
216 to 227	Request Invoice							
228 to 231	Data-In Integrity check value offset							
232 to 235	Data-Out Integrity check value offset							

Figure 3.6: Security Parameter Field

Starting Byte address: This field is used to specify the starting address for a READ or WRITE operation.

3.8 OSD Commands:

Below list shows some of the important OSD commands and their service action field values.

I. CREATE – 8882h

II. CREATE PARTITION – 888Bh

III. GET ATTRIBUTES – 888Eh

IV. SET ATTRIBUTES – 888Fh

V. READ – 8885h

VI. WRITE – 8886h

CREATE: Create command is used to create user objects on the object storage device.

CREATE PARTITION: This command is used to create or initialize a partition in the logical unit (LU).

GET ATTRIBUTES: This command is used to get the attribute values of an existing objects or partitions in a storage device.

SET ATTRIBUTES: Set Attributes command is used to set the attributes of an existing object or a partition or global attributes in a storage device.

Read: Read command is used to read data from the storage device, this command incorporates certain fields like get and set attributes field, in order to retrieve attributes of an object.

Write: This command is used to write data to the object based storage device, like Read command, write command can also be used to get or set the attributes of an object.

CHAPTER 4

LITERATURE REVIEW

This chapter provides a brief overview of relevant research efforts to improve the performance of block level iSCSI (Small Comuputer System Interface over internet) implementations, TCP tuning mechanisms, Ongoing research on Object-based storage (OBS) technology, and efforts to incorporate the Object-based storage with storage area network technologies (iSCSI)

Design considerations and functional blocks of iSCSI are explained in [8], author John L. Hufferd, has explained different constraints considered for design of iSCSI protocol, how iSCSI protocol utilizes the existing TCP connection for the traffic flow without using any additional infrastructures, and also explains the benefits and efficiency of error detection and recovery mechanisms, which are incorporated with iSCSI protocol. In [9], Kalman Z. Meth and Julian Satran explains the use of existing TCP connection for sending the traffic, implantation of sending traffic over multiple TCP connections. And also explains the complexity of mechanisms used for error recovery. In [23], author has explained different Digest errors that can be occurred during the traffic flow and different error recovery mechanisms used for each digest error.

TCP tuning mechanisms are discussed in [18,19], authors have explained how the TCP uses dynamic sliding window mechanism for improved performance in turn minimizing the error occurrence probability. And how this TCP window mechanism is implemented in different operating systems. These two papers also discuss different mechanisms of tuning the TCP window size, and also explains the impact of TCP window buffer size on overall performance.

Concepts of Object based Storage (OSD) and how this could improve the performance of iSCSI are explained in [1], Michael Factor, Kalman Meth, Dalit Naor, Ohad Rodeh and Julian Satran have given a brief overview of evolution of Object based storage device, and research prototypes for adopting OSD. This paper also discusses about different standards that are currently available. In [7], author explains the need for change in the interface, according to author, the device interface, which is holding the designer to come up with better storage solutions. In this paper author discusses different issues with the previous device interfaces, and how Object based storage device interface handle these issues in a better way, author has considered security, data sharing and device intelligence in this paper.

Object Autonomy and Data Sharing are explained in [16], Eric Riedel described the functionality of object based file systems, how the objects are placed and retrieved from the device, author has shown the implementation of security and naming conventions in comparison with existing mechanisms. In [6], author has come up with an intelligent buffering mechanism which maximizes the network attached tape devices. Implementation and performance evaluation is explained in [12], Shuibing He and Dan Feng have proposed a new switch fabric based hardware architecture, which supports parallel data transfer. And also an Object based file system. Using this new object based file system; authors have considerably increased the performance of the object based storage device.

CHAPTER 5

Results and Analysis

5.1 Experimental Setup:

The objective of this test setup is to compare the transaction time taken by the block level iSCSI and Object-storage based iSCSI for a write operation. Therefore this study requires two test beds, one with block level iSCSI implementation, and the other one with object-storage based iSCSI. The figure 5.1 depicts the Test Bed implemented for this study.

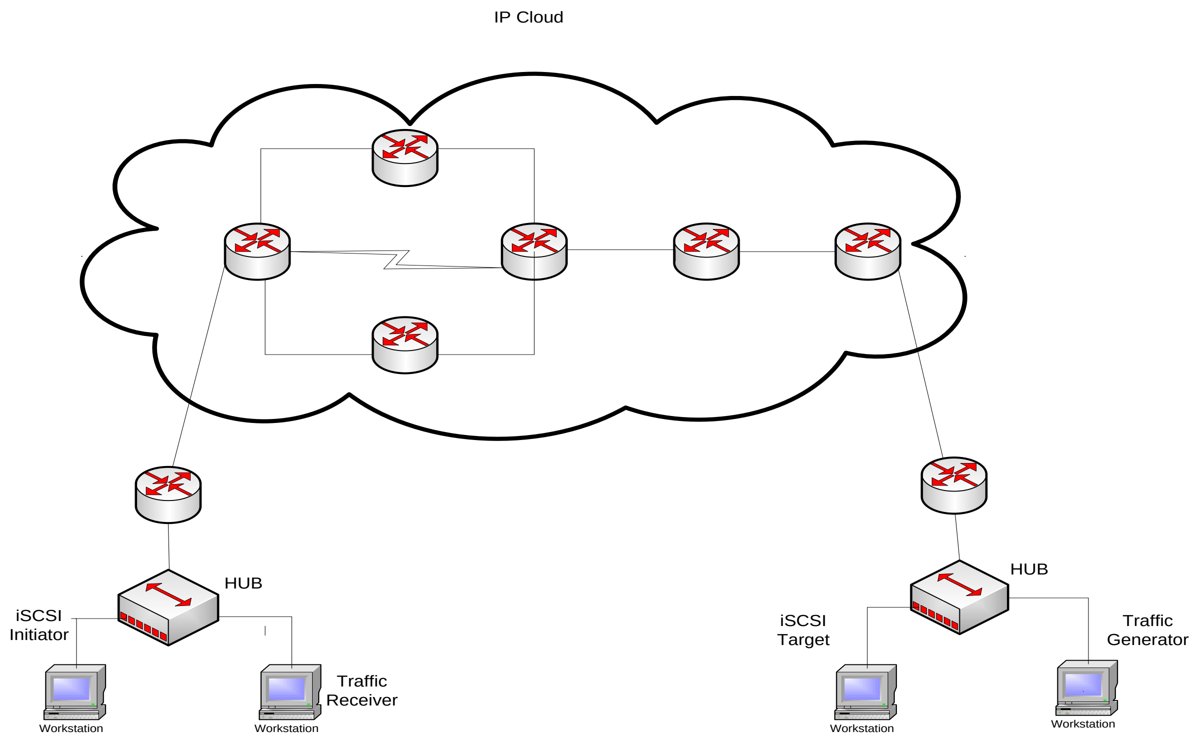
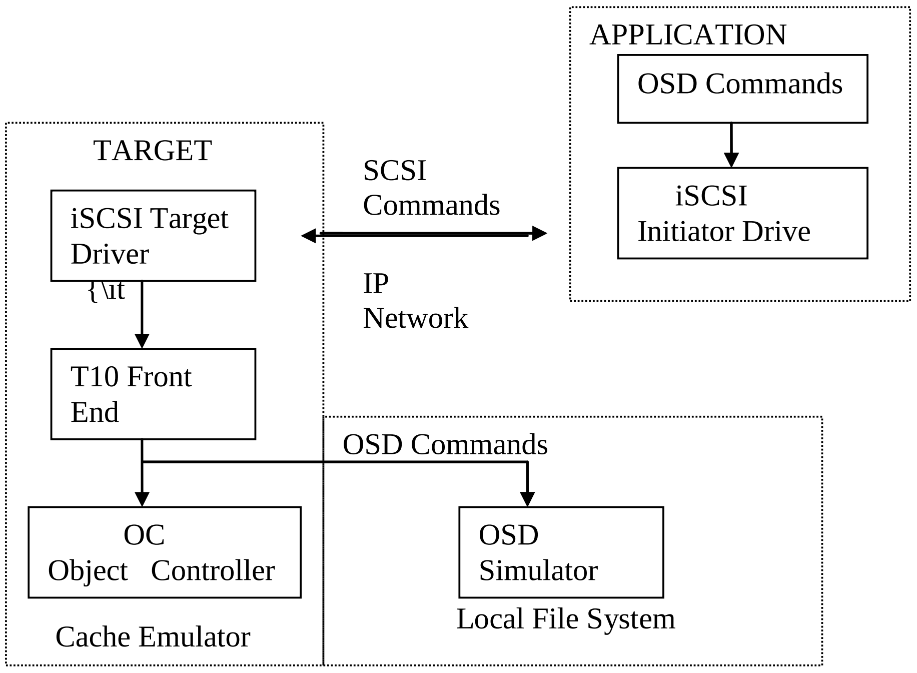


Figure 5.1: Test Setup

As it is shown in the above Figure, the iSCSI initiator and the target are connected across an IP cloud. And in order to have the real time traffic passing through, two more hosts are connected at each end, in which one host acts as http server and the other host is working as http client.

5.1.2 iSCSI components:

Both initiator and target are Linux based systems, running Linux kernel 2.6.20 (Fedora core 8) and 2.6.25 (Fedora Core 9) kernels respectively. The IBM OSD simulator is used as the target [4], and the open-OSD as the iSCSI initiator [5].



5.2 IBM OSD initiator

And Table 5.1 shows the hardware specifications of the target device.

I. layer 1 : iSCSI target

II. layer 2 : SCSI-OSD

III. layer 3 : OSD simulator

5.1.2.1 Target:

The IBM OSD simulator software uses the Linux local file system to store the objects, and it utilized GDBM to store attributes [4]. It consists of three layers:
iSCSI target layer supports extended SCSI CDB's (Command descriptor block), and bi-directional transfers [4]. The SCSI-OSD layer implements the T10-OSD standards. And the OSD Simulator layer physically stores the objects to local file system, and manages the GDBM database. Figure shows the functional block diagram of the IBM OSD simulator [5], which depicts the command flow from the iSCSI initiator to the OSD simulator layer.

CPU	2 x Intel Core 2 CPU 2.13GHz
Memory	DDR2/ 2GB
Disk	ATA SAMSUNG S2504C / 7200 RPM / 180 GB/ 300 Mbits/sec

Table 5.1 : System Specifications

5.1.2.2 Initiator:

Open-iSCSI is an open-source iSCSI initiator available for Linux environment, but the latest available version of Open-iSCSI [11] does not support the OSD (object storage device) command set. In order this needs to be patched with open-OSD [14] patch, which allows the initiator to communicate with the OSD based target and it also communicates with the SCSI layer using T10-OSD command set.

With the same specifications, one more test bed has been created on 2.6.23 kernel for block based iSCSI implementation. And for transferring the data and also for evaluating the performance, a fedora inbuilt command "dd" (which is by default streamed input and streamed output) has been used, which is generally used for copying or replicating any data.

With above setup the following test scenarios have been performed.

A write operation has been performed on the block level iSCSI implementation, in order to evaluate the performance, this test has been repeated for different data sizes. (ex 41MB, 205MB, and 410MB). And the same test has been performed on the object-storage based iSCSI using the dd command.

And in order to figure out the optimal TCP window size for the object-storage based iSCSI, the write operation has been performed several times, by changing the TCP window size at different data sizes. Example the time taken for writing 41MB of data is taken with different TCP window sizes (and the same steps have been repeated for 205MB and 410MB). And for changing the TCP window size in Linux environment is performed by parsing the required TCP window value to **[wmax]** parameter in the **wmem_max** file (which is located at /proc/sys/net/core/wmax_max), this will overwrite the default TCP window size to the optimal TCP window size for the iSCSI protocol [18]. All the values have been taken while real time traffic is going through the network. And the obtained Results have been explained in the Results and Analysis chapter.

5.2 Results and Analysis:

As part of this research work, In order to come up with a modeling for the optimal TCP window size, we have gone through the iSCSI draft and to understand the iSCSI operations. And for simulating the Write operations at different TCP window sizes, the above mentioned experimental setup has been used. And this work also includes comparing the block level vs OSD based iSCSI implementations.

5.2.1 OSD vs Block level iSCSI:

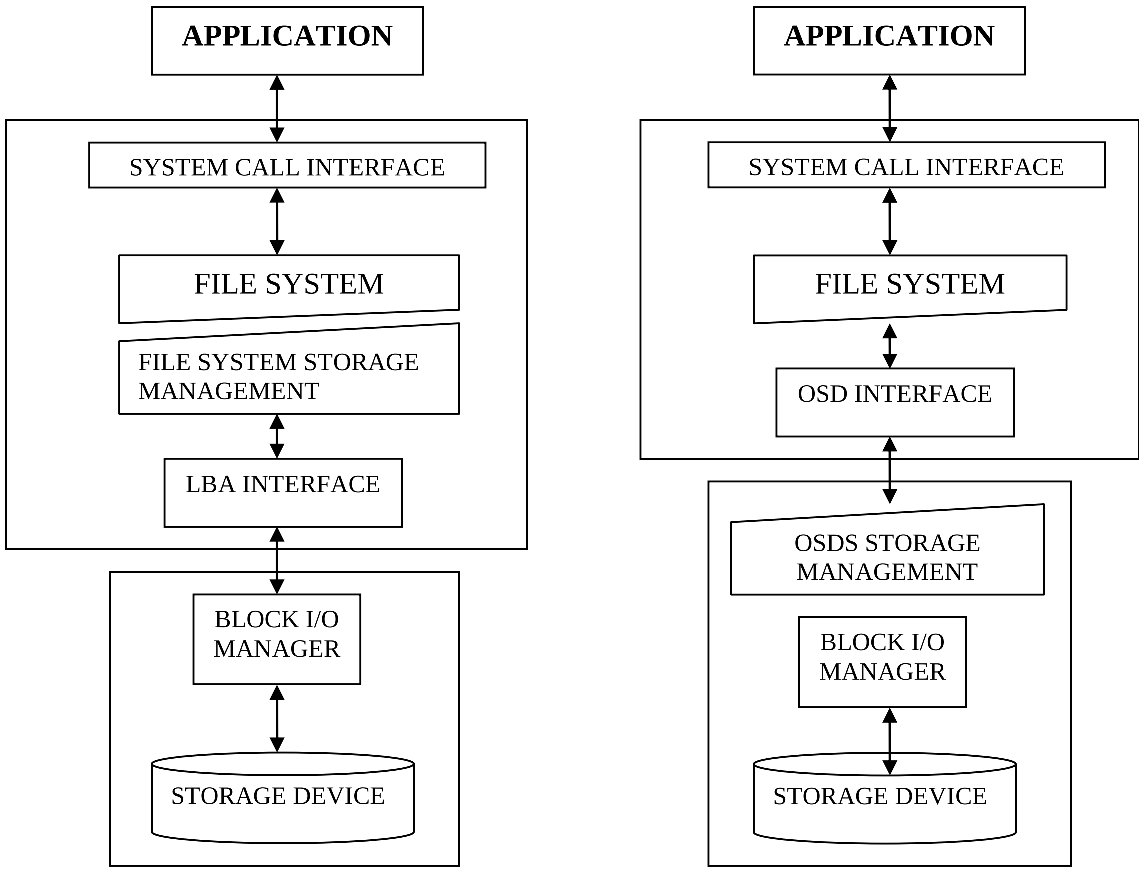


Figure 5.3 : OSD vs Block level iSCSI 1

In object based storage implementations, the Storage Management component of the file system has been moved to the object based target devices, hence all the file management tasks have been performed by the target device it self. Where as in Block level iSCSI all the file management tasks have been performed at the initiator end, hence increasing the overhead on the initiator device. And as explained in [Chapter 3], object based storage devices store data in the form of object, which can store user data and the Meta data as well. Meta data contains different attributes, which holds the information related to the user object. By using the attributes, file handling and management process utilizations and access times are reduced considerably.

Figure shows the layered model of both the OSD and block level iSCSI architectures.

i) In order to observe the difference between the iSCSI block level and the OSD based iSCSI, author has done the simulations for write operations on both the block level and OSD based iSCSI setups with different Data Sizes. Figure 5.4 shows the experimental values for both the tests. From these experimental values we can observe the performance improvement, while using the OSD based iSCSI. This is because of the reason, In OSD based iSCSI we are off-loading almost 90% of the File system operations from the initiator side.

size (MB)	41	76.5	123	164	205	246	287
Time (sec)	0.049953	1.07118	6.23085	9.73119	13.9553	17.6189	22.0577

Table 5.2: Write Times for block level iSCSI

Tables 5.2 and 5.3 show the Time taken for the Write operations for different Data Sizes with both the test setups.

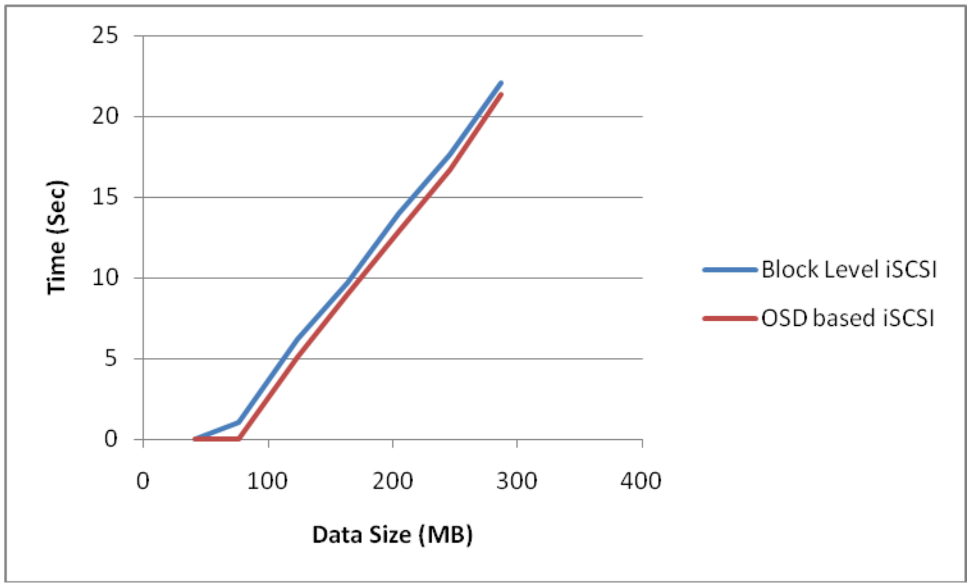


Figure 5.4 : Block level vs OSD

size (MB)	41	76.5	123	164	205	246	287
Time (Sec)	0.0346963	0.0716822	5.11576	8.99135	12.8215	16.63	21.3289

Table 5.3 : Write Times for osd based iSCSI

From the above results, by implementing the Object based storage iSCSI, the performance can be improved.

5.2.2 Varying TCP Window Size:

We have taken the values from the OSD based iSCSI implementation, by changing the TCP window size. Figures 5.5, 5.6 and 5.7 show the Time taken for transferring 41MB, 205MB, and 410MB respectively. From the figures 5.5, 5.6 and 5.7, we can observe that by varying the TCP window size at some point we can get the optimal performance.

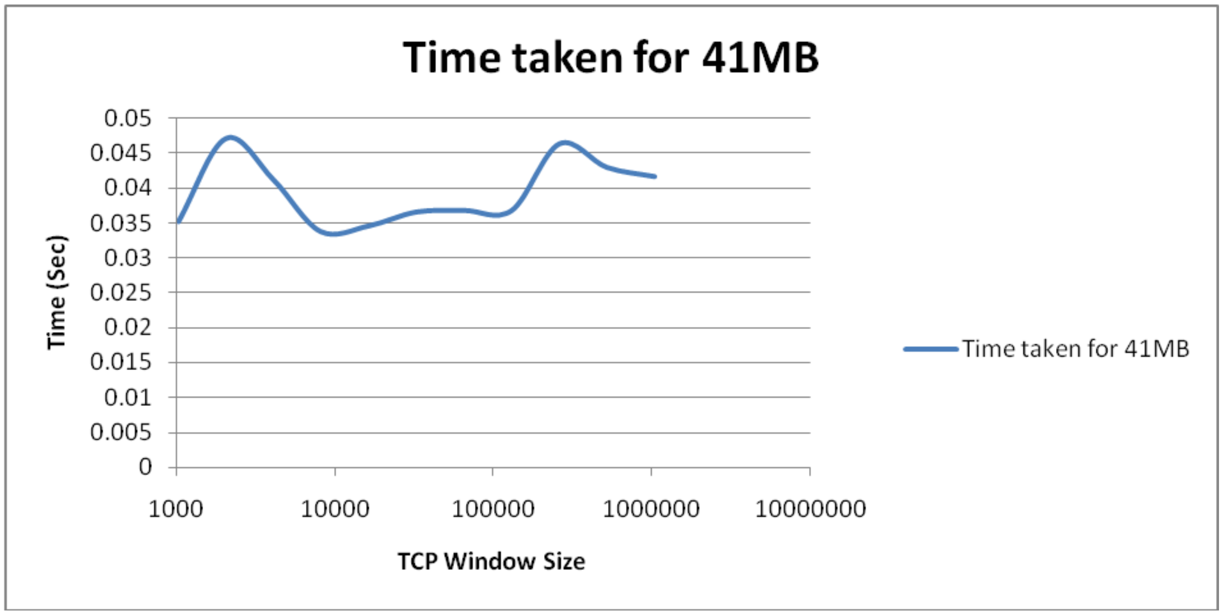


Figure 5.5 : Time taken for 41MB of data

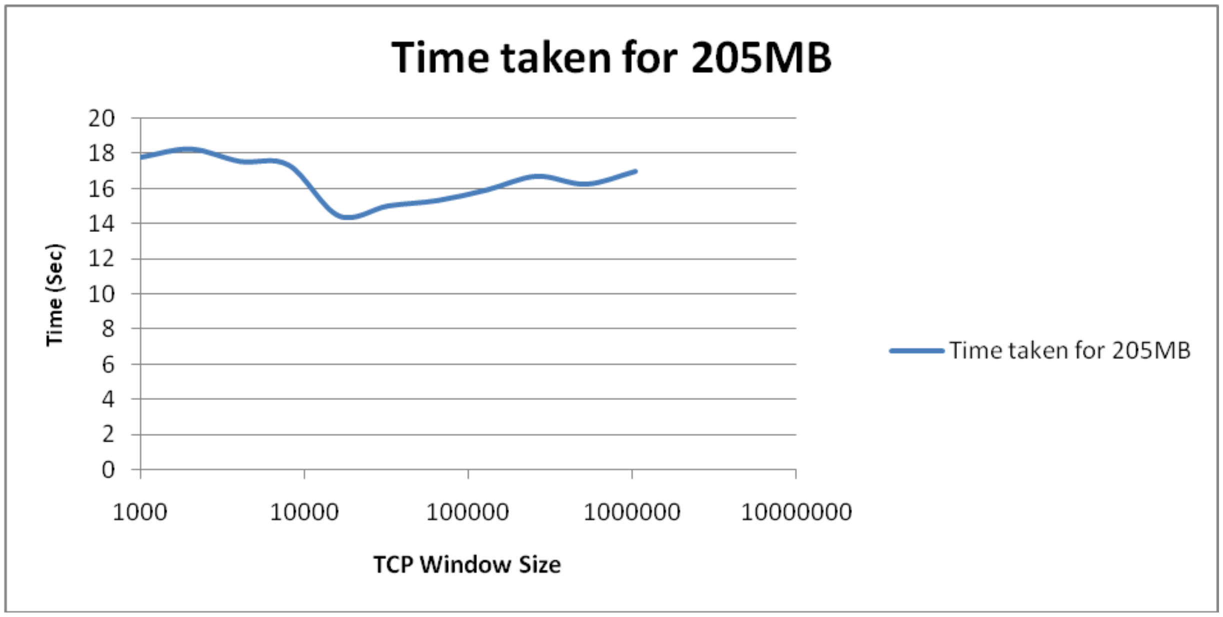


Figure 5.6 : Time taken for 205MB of data

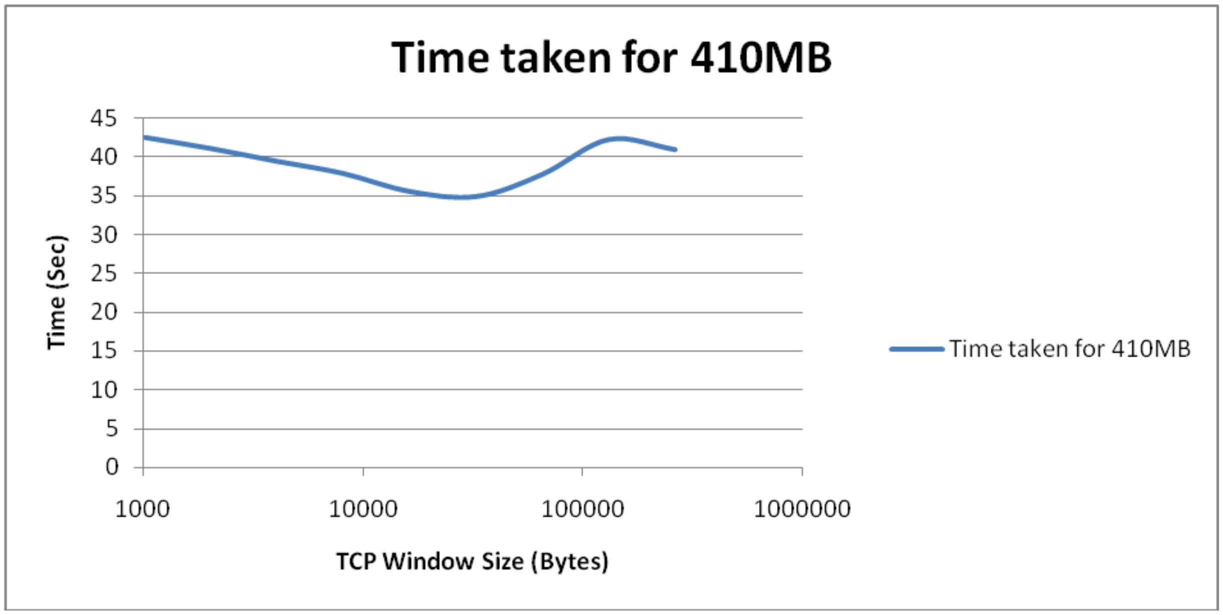


Figure 5.7: Time taken for 410MB of data

From the figures 5.5, 5.6 and 5.7, we can observe that the time taken for the write operation is varying with the TCP window, and as the data size is increasing the optimal TCP window value is increasing.

5.3 Mathematical Modeling:

The optimal Time taken for a given file size can be found by using the following math model. The total time taken for a write operation would be the summation of the time taken for the data transfer and the time taken due to the errors occurred in the iSCSI layer. The error recovery time may vary depending upon the level of error recovery level that has been selected.

For this modeling, $\boldsymbol{P_2}$ is considered as the probability of the error occurrence. And as it is mentioned in [error recovery level], if the iSCSI implementation supports error recovery level 1, it has to support error recovery level 0 also, but iSCSI implementations with error recovery level 0 do not support error recovery level 1. Here, we are assuming that if the iSCSI implementation supports error recovery level 1, both $\boldsymbol{E_0}$ and $\boldsymbol{E_1}$ as 0.5. if the iSCSI implementation supports error recovery level 0, $\boldsymbol{E_1}$ is considered as 0, and $\boldsymbol{E_1}$ will be 1. With the Error recovery level 0 implementation, if an error occurs the operation must be terminated, then it waits for a **DefaultTime2Wait** before re establishing the TCP connection, and re transmits the data. Hence if an error occurs it takes sum of $\boldsymbol{\Delta_{wait}}$ which is the **DefaultTime2Retain**, $\boldsymbol{\Delta_{login}}$ which is the time taken for login, and $\boldsymbol{\Delta_{trans}}$ which is the time taken for retransmission.

Optimal Buffer Size = 2 * bandwidth * delay

$T_{opt} = \min [\, P_1 * [\Delta_{Trans}] + P_2 * [E_0 * (\, \Delta_{tans} + \Delta_{login} + \Delta_{wait}) + E_1 * (\Delta_{tans} + \Delta_{proc} + RTT)]]$

(1)

in which

P_2 = probability of the iSCSI level error occurrence.

$P_1 = (1 – P_2)$

$E_0 = 1$ (if error recovery level is 0)

$E_1 = 1$ (if error recovery level is 1)

Δ_{Trans} = [(Data Size – First Burst Length)/Max Burst Length] * Δ_{trans}

$\Delta_{trans\,=}$ [Max Burst Length / MSS] * Δ_{tcp}

Δ_{wait} = Default time to wait

Δ_{tcp} = TCP layer parameters.

Δ_{trans} is the time taken to process one Maximum Burst Length of data, in order to send one Maximum Burst length, it involves the TCP delays

Δ_{trans} = [Max Burst Length * (1+S)] * Δ_{tcp} + Δ_{R2T} + Δ_{proc} (2)

Δ_{R2T} = Δ_{proc} + (RTT/2) (3)

S = Throughput of the iSCSI layer.

RTT = Round Trip Time

TCP layer modeling:

At the TCP layer the delays introduced are represented by RTT, which is the round trip time.

Δ_{tcp} = (RTT/ MSS) + (RTT/2 * T_{window}) (4)

= RTT (1/MSS + ½ * T_{window})

T_{window} = TCP Window Size (varying from 1024 to 1048576)

MSS = Maximum Segment Size

T_{window_opt} = T_{window} (where the minimum transaction time is obtained). (5)

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion:

In this research work, both the Block level storage and Object level storage have been carefully studied and implemented. The author has also compared these two levels of storage mechanisms and after a careful review of all these techniques, the author has found a significant improvement in the performance, the reason being, the non-complex storage techniques and off loading the file management tasks from the application client side.

From this study, the author has also discovered that the selection of the optimal TCP window size depends on the iSCSI parameters like DataSize, DefaultTime2Wait, DefaultTime2Retain and MaxBurstLength and also on the type of the Error recovery level mechanisms being implemented. From the simulation values and the theoretical study, the author has concluded that the optimal TCP window size is directly proportional to the DataSize and inversely proportional to the MaxBurstLength.

6.2 Future Work:

This proposal can be extended further by programming the Middleware Driver in such a way that it utilizes the proposed Mathematical model to calculate optimal TCP window size and to then send these values to the kernel in order to set the optimal TCP window size for the existing iSCSI session. This entire study is confined to the object based storage technique and hence can be extended by incorporating with iSER. Using object attributes, access permissions and policies can be assigned to each object.

REFERENCES

LIST OF REFERENCES

[1] Michael Factor, Kalman Meth, Dalit Naor, Ohad Rodeh, Julian Satran “The future building block for storage systems”, IBM Corp., Object storage.

[2] Dingshan He, Nagapramod Mandagere, David Du, " Design and Impelementation of a Network Aware Object-based Tape Device" , 24th IEEE conference on Mass Storage Systems and Technology, 2007.

[3] “Object-Based Storage Device Commands (T10 draft)”, ANSI INCITS 400-2004, Rev 10, Date: July 2004.

[4] “Object-Based Storage Devices - 2 (OSD-2)”, Project: 1729-D, Rev 04, July 2008.

[5] Allon Shafrir and Petra Reshef, “IBM Object Storage Device Simulator for Linux” IBM Haifa Research Laboratory, October 2005.

[6] “OSD initiator”, IBM Corp., Aug 2005.

[7] Mike Mesnier, Gregory R. Ganger, Erik Riedel, " Object-Based storage" IEEE Communications Magazine, 2003.

[8] John L. Hufferd, "iSCSI : The Universal Storage Connection", ISBN : 0-201-78419, November 2002.

[9] Kalman Z. Meth, Julian Satran, "Design of iSCSI Protocol". IBM Haifa Research Laboratory.

[10] "Internet Small Computer Systems Interface". RFC 3720.

[11] Alex Aizman and Dmitry Yusupov, “Open-iSCSI project”, October 2004.

[12] Shuibing He, Dan Feng, "Implementation and Performance Evaluation of an Object-based Storage Device" , 2003.

[13] Michael Mesnier , Sandeep Nair, “Intel's Reference Implementation for osd based iSCSI”.

[14] Benny Halevy, “Open-osd project”.

[15] “SCSI Standards Architecture, SCSI-3 Architecture model”, T10 Technical committee on SCSI storage interfaces.

[16] Eric Riedel, "Object-based Storage Device (OSD) - Architecture and Sysems" April 2006.

[17] Takahiro Matsuo, "Scalable Automatic Buffer Tuning to Provide High Performance and Fair Service for TCP Connections", IEEE INET 2000.

[18] “TCP Tuning guide”, Lawrence Berkeley National Laboratory.

[19] Swedish University Computer Network “TCP Tuning parameters for different OS”.

[20] “iSCSI T10 draft” T10 Technical committee on SCSI storage interfaces.

[21] Christian Bandulet, “Object-Based Storage Devices”, Sun Microsystems July 2007.

[22] “SCSI IETF Draft” T10 Technical committee on SCSI storage interfaces.

[23] Mark Bakke, “Recovering from iSCSI Digest Errors” June 2001.