iSCSI PERFORMANCE OVER RDMA-ENABLED NETWORK

A Thesis by

Amarnath Pallampati

Bachelor’s Degree in Electronics and Communications Engineering, VNR VJIET, 2004

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

July 2006
I have examined the final copy of this thesis for form and content, and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science with a major in Electrical and Computer Engineering.

Dr. Ravi Pendse, Committee Chair

We have read this thesis and recommend its acceptance:

Dr. John Watkins, Committee Member

Dr. Krishnan, Committee Member
DEDICATION

To my parents and my family who have supported me throughout my life. Without their help, it would not have been possible to finish my thesis.
ACKNOWLEDGEMENTS

I would like to gratefully acknowledge my advisor, Dr. Ravi Pendse, for giving me an opportunity to do my Master’s thesis under him in the field of storage networking. I would also like to thank my committee for taking the time to work with me in this endeavor.

I am thankful to Amarnath Jasti and all my friends at the Advanced Networking Research Center (ANRC) at Wichita State University for their help they have given me during this time.

Finally, I am forever indebted to my parents and family for supporting and encouraging me throughout my life to achieve higher goals.
ABSTRACT

With an increase in the amount of information being exchanged every day, there is a need for storage repositories and faster networks to retrieve data with a minimum amount of time delay. Keeping this in mind, inroads have been made in developing different types of storage network technologies including Direct Attached Storage (DAS), Network Attached Storage (NAS), and Storage Area Network (SAN). Developments have also been made in building faster and parallel Input/Output (I/O) peripheral interface systems such as the Small Computer System Interface (SCSI). With the help of SCSI high-speed bus systems that quickly transfer large amounts of data requested by the user have been made, which is a very important need in storage are networks.

Internet SCSI (iSCSI) protocol is a Small Computer System Interface (SCSI) transport protocol developed by the Internet Engineering Task Force (IETF), which maps block-oriented storage data over Transmission Control Protocol/Internet Protocol (TCP/IP) networks. This iSCSI storage technology is a viable solution utilizing IP networks for low-cost implementation of managing storage networks. Unlike the file access mechanism provided by protocols such as Network File System (NFS), iSCSI implementation has a block access mechanism to provide better performance and throughput. Remote Direct Memory Access (RDMA) technology provided by Internet Wide Area RDMA Protocol (iWARP), which runs over TCP/IP networks provides efficient data transfers. RDMA protocol is primarily chosen because it efficiently uses storage I/O (Input/Output) systems by providing a Zero Copy transfer mechanism.

Implementing iSCSI over RDMA-enabled network adapters exploits RDMA operations for efficient data transfers using the existing infrastructure. This thesis involves the implementation and performance evaluation of iSCSI over RDMA, a process that maps iSCSI
protocol over the iWARP protocol suite. The main idea behind this implementation is to offload the data transfer operation incurred in iSCSI from TCP to RDMA operations, which has shown the benefits of attaining better performance. Since a large portion of data is transferred during the iSCSI storage I/O process, RDMA with its Zero Copy transfer ability provides better performance.

iSCSI over RDMA uses the generic data movement of iSCSI to deliver data efficiently. This particular implementation provides a chance for iSCSI to operate in conjunction with RDMA over TCP/IP since RDMA-enabled Network Interface Controllers (RNICs) are expected to become pervasive.

This study analyzes the performance of SAN storage devices, demonstrating that iSCSI over RDMA performs better than running only iSCSI. In addition, the low cost and ease with which an iSCSI over RDMA can be managed make it superior to iSCSI, considering the performance gained.

The iSCSI target from iSCSI Enterprise Target (IET) installed in a Fedora Core 5 machine with a 2.6.16 kernel was used to evaluate the iSCSI. The Initiator, was built using a Fedora Core 5 source iSCSI Initiator in a Fedora Core 5 machine with a 2.6.16 kernel and Microsoft iSCSI Initiator in a Microsoft server 2003 machine [1][2].

This research used RNICs developed by Ammasso Corporation, to directly place data into Initiator buffers using Zero Copy transfer. Hence, unnecessary multiple copying procedures are eliminated, thereby memory bus utilization and improving performance, which definitely impacts the overall performance of a system when there is large amount of data being transferred as occurs with iSCSI. The iSCSI over RDMA protocol implementation gives high performance and low overhead for I/O storage.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>INTRODUCTION</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>Growth of Storage Area Networks</td>
<td>1</td>
</tr>
<tr>
<td>1.1.1</td>
<td>Direct Attached Storage</td>
<td>2</td>
</tr>
<tr>
<td>1.1.2</td>
<td>Network Attached Storage</td>
<td>2</td>
</tr>
<tr>
<td>1.1.3</td>
<td>Storage Area Network</td>
<td>4</td>
</tr>
<tr>
<td>1.2</td>
<td>Objectives</td>
<td>5</td>
</tr>
<tr>
<td>1.3</td>
<td>Organization of Thesis</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter</th>
<th>INTRODUCTION TO SCSI</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.</td>
<td>INTRODUCTION TO SCSI</td>
<td>7</td>
</tr>
<tr>
<td>2.1</td>
<td>SCSI Commands</td>
<td>7</td>
</tr>
<tr>
<td>2.2</td>
<td>SCSI Messages and Handshaking</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>SCSI Bus Types</td>
<td>8</td>
</tr>
<tr>
<td>2.3.1</td>
<td>SCSI – 1</td>
<td>8</td>
</tr>
<tr>
<td>2.3.2</td>
<td>SCSI – 2</td>
<td>9</td>
</tr>
<tr>
<td>2.3.3</td>
<td>SCSI – 3</td>
<td>9</td>
</tr>
<tr>
<td>2.3.4</td>
<td>SCSI Bus Signals</td>
<td>10</td>
</tr>
<tr>
<td>2.3.5</td>
<td>SCSI Phases</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter</th>
<th>OVERVIEW OF iSCSI PROTOCOL</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.</td>
<td>OVERVIEW OF iSCSI PROTOCOL</td>
<td>12</td>
</tr>
<tr>
<td>3.1</td>
<td>Introduction to iSCSI</td>
<td>12</td>
</tr>
<tr>
<td>3.2</td>
<td>iSCSI Naming</td>
<td>14</td>
</tr>
<tr>
<td>3.3</td>
<td>iSCSI Session Establishment Procedure</td>
<td>17</td>
</tr>
<tr>
<td>3.4</td>
<td>iSCSI Implementation</td>
<td>19</td>
</tr>
<tr>
<td>3.5</td>
<td>iSCSI Read and Write</td>
<td>19</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter</th>
<th>INTRODUCTION TO RDMA PROTOCOL</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.</td>
<td>INTRODUCTION TO RDMA PROTOCOL</td>
<td>22</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction to RDMA</td>
<td>22</td>
</tr>
<tr>
<td>4.2</td>
<td>RDMA Overview</td>
<td>23</td>
</tr>
<tr>
<td>4.3</td>
<td>RDMA Layering Overview</td>
<td>24</td>
</tr>
<tr>
<td>4.3.1</td>
<td>RDMA</td>
<td>24</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Direct data placement</td>
<td>25</td>
</tr>
<tr>
<td>4.3.3</td>
<td>MPA</td>
<td>26</td>
</tr>
<tr>
<td>4.3.4</td>
<td>TCP and IP</td>
<td>26</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS (continued)

4.4  RDMA data transfer operations ................................................................. 27
4.4.1 RDMA Send ................................................................. 27
4.4.2 RDMA Write .............................................................. 28
4.4.3 RDMA Read ............................................................. 28
4.4.4 RDMA Terminate ....................................................... 29
4.5  RDMA Data Transfer Example ............................................................... 29
4.6  AMMASSO RNIC ................................................................. 30
4.7  Advantage of Using RDMA ................................................................. 31

5.  PERFORMANCE ANALYSIS OF OF iSCSI OVER RDMA................................. 32
5.1  Test Objectives ........................................................................ 35
5.2  Test Procedure ........................................................................ 36
5.3  Theoretical Calculations .............................................................. 37
5.4  Analysis of Test Results and Parameters ....................................... 38

6.  CONCLUSIONS AND FUTURE WORK .................................................. 43

LIST OF REFERENCES ................................................................................. 45
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>4</td>
</tr>
<tr>
<td>2.1</td>
<td>11</td>
</tr>
<tr>
<td>3.1</td>
<td>12</td>
</tr>
<tr>
<td>3.2</td>
<td>16</td>
</tr>
<tr>
<td>3.3</td>
<td>18</td>
</tr>
<tr>
<td>3.4</td>
<td>20</td>
</tr>
<tr>
<td>3.5</td>
<td>21</td>
</tr>
<tr>
<td>4.1</td>
<td>24</td>
</tr>
<tr>
<td>5.1</td>
<td>34</td>
</tr>
<tr>
<td>5.2</td>
<td>35</td>
</tr>
<tr>
<td>5.3</td>
<td>38</td>
</tr>
<tr>
<td>5.4</td>
<td>40</td>
</tr>
<tr>
<td>5.4</td>
<td>41</td>
</tr>
</tbody>
</table>
# LIST OF ACRONYMS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIC</td>
<td>Application Specific Integrated Circuit</td>
</tr>
<tr>
<td>CCIL</td>
<td>Cluster Core Interface Language</td>
</tr>
<tr>
<td>CDB</td>
<td>Command Descriptor Block</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclical Redundancy Check</td>
</tr>
<tr>
<td>DAS</td>
<td>Direct Attached Storage</td>
</tr>
<tr>
<td>DDP</td>
<td>Direct Data Placement</td>
</tr>
<tr>
<td>EUI</td>
<td>Extended Unique Identifier</td>
</tr>
<tr>
<td>FPDU</td>
<td>Framed PDU</td>
</tr>
<tr>
<td>FC</td>
<td>Fedora Core</td>
</tr>
<tr>
<td>HBA</td>
<td>Host Bus Adapter</td>
</tr>
<tr>
<td>IET</td>
<td>iSCSI Enterprise Target</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output</td>
</tr>
<tr>
<td>IOPS</td>
<td>Input/Output Per Second</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IPSEC</td>
<td>Internet Protocol Security</td>
</tr>
<tr>
<td>ISID</td>
<td>Initiator Session Identification</td>
</tr>
<tr>
<td>IT</td>
<td>Information Technology</td>
</tr>
<tr>
<td>iSCSI</td>
<td>Internet Small Computer System Interface</td>
</tr>
<tr>
<td>iqn</td>
<td>iSCSI Qualified Name</td>
</tr>
<tr>
<td>iWARP</td>
<td>Internet Wide Area RDMA Protocol</td>
</tr>
</tbody>
</table>
LIST OF ACRONYMS (Continued)

JFS......................................................................................................................Journal File System
LAN ..................................................................................................................Local Area Network
LBA...............................................................................................................Logical Block Address
MPA..............................................................................................................Marker-Based Protocol-Data-Unit-Aligned
NAS.........................................................................................................Network Attached Storage
PDU..................................................................................................................Protocol Data Units
RDMA.............................................................................................. Remote Direct Memory access
RNIC........................................................................................................ RDMA network interface controller
R2T ......................................................................................................................Ready to Transmit
RSH..................................................................................................................Remote Shell
SAN................................................................................................................ Storage Area Network
SCSI ............................................................................................ Small Computer System Interface
TCP/IP.............................................................................................Transmission Control Protocol/Internet Protocol
TSIN........................................................................................................ Target Session Identification
ULP.................................................................................................................Upper Layer Protocol
1.1 Growth of Storage Networks

With the enormous amount of data being exchanged daily there is a definite need for storage repositories and the ability to retrieve data with a minimum amount of time delay. Storage capacity has grown significantly, and due to shrinking budgets, Information Technology (IT) managers are forced to find storage solutions that improve disk utilization, data availability, performance, and protection; and reduces the number of data centers, maintenance costs and Central Processing Unit (CPU) loads on storage devices.

Basically, network storage is simply a method to store data and make it available to clients over the network. Over the year’s, keeping the above requirements in mind, developments have been made in technology in order to transfer data efficiently over networks in a minimum amount of time [3].

Brief summary of the various phases of the development of storing and accessing data over the network.

1.1.1 Direct Attached Storage

First and foremost, Direct Attached Storage (DAS) is a simple and easy storage mechanism where by storage device is directly attached to a host system. An Ideal example of DAS would be an internal hard drive connected directly to the computer, i.e., the client has direct access to the storage device. However, with the need to maintain storage devices with increased capacities and the need to share data over the network, this particular technology cannot be implemented over that could be accessed appropriately by clients.
1.1.2 Network Attached Storage

Efforts to achieve these needs led to the development of a storage technology called Network Attached Storage (NAS) as show in Figure 1.1. Here data is stored over the network, and clients/servers access the data in the form of files [4] by connecting to a storage device identified by a unique Internet Protocol (IP) address assigned to a Network Interface Card (NIC). The advantage of this type of network over DAS is its centralized administration and easier, more secure. Also, there is no single point of failure, as in DAS; hence, there are more available storage devices over the network. This particular technology has widely grown over the years and used in many firms and organizations.

However, IT managers and companies needed an even better system that would provide faster transmission of data between clients and storage devices, and also include the ability to provide data back-up without impacting the Local Area Network (LAN) where actual clients reside. Hence the need for a Storage Area Network (SAN).

1.1.3 Storage Area Networking

SAN is a storage network specifically designed to interconnect high-speed storage devices and servers, as show in Figure 1.2. This specific network can span a large number of systems and geographic locations. Unlike in NAS, SAN is connected behind the servers where storage devices and clients/servers are on same local segment. SAN provides a block-level interface to clients as opposed to the file-level interface in NAS, which provides a faster mechanism to transfer data over the network, i.e., reading and writing of data occurs at block level or the transfer of raw data at physical level.

An example of how data is transferred in actual SAN, can be seen in Figure 1.2. When a client on a network requests information from a storage device, the request is received by server,
Figure 1.1. Network attached storage.
Figure 1.2. Storage area network.
which in turn obtains the data from the storage device and sends it to the appropriate client over the network. Hence, the server has full control over the storage device and transfers the appropriate data requested by the client in the minimum possible amount of time. This particular block-level transfer provides a faster level of data transmission than to file-level transfers in NAS systems, since there is no overhead involved.

Also, backups in the SAN network do not affect the rest of the LAN, since the back-up data passes only over the SAN, providing a LAN free back-up environment and less chances of network congestion occurred in frequent backups of data over the LAN.

Hence for providing faster transmission of data over SAN, devices like SCSI are used to transfer block of data unlike other devices that are used for storage.

1.2 Objectives

The major issue involving an Internet Small Computer System Interface (iSCSI), is the overhead incurred in its data transfer, i.e., degradation incurred due to extra memory copies with the host and due to an increase in CPU utilization. To address this issue, iSCSI over RDMA was implemented to eliminate extra memory copies in Transport Control Protocol/Internet Protocol (TCP/IP).

This particular implementation allowed RDMA-enabled Network Interface Controllers (RNICs) to directly place data into Initiator buffers using Zero Copy transfer. Packet reordering, which is also required in the iSCSI protocol, experiences a few problems when there is a large amount of data transferred over the network. Since each TCP segment does not have an iSCSI header and there is a need for an Upper Layer Protocol (ULP) for signaling out-of-order packets, a large amount of buffering is necessary, which is very costly and hence an obstacle to wide deployment.
In order to avoid this bottleneck and to better utilize the memory subsystem, memory bandwidth, and CPU cycles, a RDMA data-transfer procedure was employed. The iWARP protocol provided RDMA semantics over TCP/IP networks, and iSCSI over RDMA mapped the iSCSI protocol over iWARP protocol.

This Thesis analyzes the performance of SAN storage devices, demonstrating that iSCSI over RDMA performs better than running iSCSI alone. In addition, the low cost and ease with which an iSCSI over RDMA can be managed make it superior to iSCSI when considering the gain in performance. Also this thesis studies the role that implementing iSCSI over RDMA played in evolving the RDMA-enabled Network Interface Controller (RNIC) and the functionality of the iWARP protocol suite.

1.3 Organization of Thesis

The organization of the remainder of this paper is as follows: Chapter 2 provides an overview of the SCSI protocol and SCSI devices, Chapter 3 gives a summary of the iSCSI protocol, Chapter 4 provides an overview of the RDMA protocol, Chapter 5 presents performance analysis of iSCSI over RDMA and results, Chapter 6 provides conclusions and future work.
CHAPTER 2
INTRODUCTION TO SCSI

The Small Computer System Interface (SCSI), which is primarily used in SAN environments, is a computer industry standard for connecting computers to peripheral devices such as hard disk drives, CD-ROM drives, etc., where there is a need to transfer large amounts of data quickly. SCSI is a local I/O (Input/Output) high-speed bus technology used to interconnect peripheral devices to computers. The SCSI interface defines a logical-level rather than a device-level interface [5] to the disk, which allows the system's view of the disk to be independent of the physical geometry of the disk device in order to independently develop systems and peripheral devices that could be used together. This allows companies to integrate technology and cost-saving advancements rapidly.

2.1 SCSI Commands

SCSI devices are connected to the computer using an SCSI bus, and each SCSI device connected is identified using a unique SCSI identification number which ranges from 0 to 16 [6]. SCSI devices transfer data using instructions called SCSI commands for reading and writing data in blocks. In turn, these SCSI commands are contained in a Command Descriptor Block (CDB), specifying the operation requested and number of bytes required to complete that operation.

The major difference between the SCSI protocol and other interfaces used for storage devices is that SCSI commands address a device as a series of logical blocks rather than in terms of heads, tracks, and sectors, as done in file-transfer protocols. This particular implementation allows SCSI to be used with multi-vendor devices.
2.2 **SCSI Messages and Handshaking**

SCSI messages are used for communicating a number of possible messages between the Initiator and Target, as described below, that indicate the successful completion of an operation, requests, and status information. All messages are sent during the message phase. The SCSI standard uses handshaking procedures so that SCSI devices request and acknowledge data and control signals for reliable communication.

The SCSI information transfer phase uses this handshaking to transfer data or to control information between the Initiator and Target, in either direction. As an example, the Initiator senses a REQ signal, reads the data lines, and then asserts the ACK signal. When the Target senses the ACK signal, it releases the data lines and negates the REQ signal. The Initiator then senses that the REQ signal has been negated, and negates the ACK signal. After the Target senses that the ACK signal has been negated, it can repeat the whole process again, to transfer another byte of data.

2.3 **SCSI Bus Types**

2.3.1 **SCSI-1**

SCSI-1 [7] initially called SCSI, shared an eight bit-wide bus that operated at up to 5 MHz, and up to eight devices could be attached to it and daisy-chained together to form a bus. Any two devices on the SCSI bus could communicate by setting up a connection, exchanging control information, and transferring data between each other. The device that initiates the connection is called the Initiator, and the destination of the Initiator connection is called the Target. In practice, generally the host-system interfaces that initiate communications over the bus are Initiators, while peripherals such as disks, etc., are generally the Targets of these communications. Individual devices on an SCSI bus are distinguished from one another through
a unique SCSI identification number or SCSI ID, and the SCSI allows each Target to be subdivided into logical units called Logical Unit Numbers (LUNs) up to a maximum of eight logical units per Target.

2.3.2  SCSI-2

SCSI-2 [8] was developed in order to keep pace with technology changes, operating at 10 MHz as opposed to 5 MHz in SCSI-1 and the bus was widened to sixteen bits called a Wide bus. Widening the bus allowed for increased data transfer rates and up to 16 devices. Data transfer rates on this 16 bit Wide bus give twice the performance of that an 8 bit Wide bus.

These additional features gave way to the development of Fast SCSI, Wide SCSI, and Fast Wide SCSI. A Fast SCSI bus is 8 bits in width and can support eight devices operating at 10 MHz a Wide SCSI bus is 16 bits in width, supporting a connection of up to maximum of 16 devices operating at 5 MHz, a Fast-Wide SCSI bus is 16 bits in width and supports up to 16 devices operating at 10 MHz. Along with the increased number of bus types, SCSI-1 and SCSI-2 are compatible with each other.

2.3.3  SCSI-3

Further research led to the development of the SCSI-3 standard, which increased the data transfer cycles to 20 MHz (Ultra) and again up to 40 MHz. (Ultra2). Ultra and Ultra2 SCSI are supported in either 8-bit or 16-bit Wide Ultra SCSI and Wide Ultra2 SCSI.

2.3.4  SCSI Bus Signals

The SCSI specification defines the 18 SCSI bus signals with others used for grounding. Nine of these signals are used to initiate and control transactions, and another nine are used for data transfer, including a parity bit as follows:
a. Busy signal to indicate that currently SCSI bus is in use.

b. Select signal to choose the Target among those available for communication.

c. Control signal and Data signal to indicate if the command used is for sending control information or data information.

d. Input/Output signal to indicate direction of movement of data in respect to Initiator.

e. Message signal to indicate message phase by Target.

f. Request signal to indicate the handshake procedure during the connection setup process with SCSI devices.

g. Acknowledge signal in response to request signal sent.

h. Attention signal used by Initiator to indicate Target of its readiness to send messages.

i. Reset signal to release SCSI bus so that other devices can use the SCSI bus.

j. Data signal along with parity bit set used during data transfer operation.

2.3.5 SCSI Phases

Phases involved in SCSI transfer are as follows:

a. Bus Free: Indicates that no SCSI devices are using the bus, and that the bus is available for SCSI devices as show in Figure 2.1.

b. Arbitration: Permits an SCSI device to gain control of the SCSI bus. Also permits other devices wishing to use the Busy signal to gain control and put their SCSI ID on the bus. The device with the highest SCSI ID wins the arbitration.

c. Selection: Lets the device that won arbitration use the bus for data transfer.

d. Reselection: Disconnects and reconnects SCSI devices from the bus during lengthy operations.

e. Command: Allows Target to requests a command from the Initiator.
f. Data: Allows Target to request a transfer of data to or from the Initiator.

g. Status: Occurs when Target requests that status information be sent to the Initiator.

h. Message: Allows Target to request the transfer of a message to the Initiator. Messages are small blocks of data that carry information or requests between the Initiator and the Target. Multiple messages can be sent during this phase.

Figure 2.1. SCSI phases.
3.1 Introduction to iSCSI

Internet SCSI (iSCSI) is an Internet Engineering Task Force (IETF) draft standard protocol. It is known as a client/server protocol or an Initiator/Target protocol that uses the TCP/IP connection to exchange SCSI commands. These SCSI commands are encapsulated in Protocol Data Units (PDUs) called iSCSI PDUs [9]. The Target is responsible for providing proper data requested by the Initiator and reporting to the Initiator completion of the data-transfer operation. The iSCSI Protocol [10], as show in Figure 3.1, maps the SCSI remote procedure model to the TCP/IP protocol.

![iSCSI protocol model](image-url)
The iSCSI protocol [11] locates itself on top of Transport Control Protocol. Since features like congestion control and flow control already exists in TCP, there are very limited constraints in iSCSI, unlike other protocols which take care of these features independently, making the protocols design and use more complex.

Since iSCSI runs over the TCP, it can utilize features such as guaranteed in order delivery of data, congestion control, etc., which provides a reliable connection and works over a variety of physical media and interconnect topologies. Added to these features, discussed above, TCP offers an end-to-end connection model independent of the underlying network. Also, since TCP has a mechanism to acknowledge all TCP packets that are received and to re-send packets that are not acknowledged within a certain time-out period. Packets may be re-sent automatically using TCP. Also, congestion control techniques are remedied by TCP. Hence, the iSCSI design is made simpler by eliminating those features that are already provided using TCP.

Although TCP has additional features that are not needed for transport of SCSI in general, the designers of iSCSI felt that the benefits of using an existing, well-tested transport protocol like TCP would justify its use. As mentioned previously, the iSCSI protocol defines its packets as iSCSI Protocol Data Units (PDUs) consisting of a header and possible data, where the data length is specified within the PDU header.

iSCSI runs on different modes, based on the parameters that were negotiated during the login phase. This protocol can establish a single TCP connection between the Initiator and Target or can have multiple paths using multiple TCP connections [12] for a single session, and can exchange both data and control messages on all connections. The multiple paths are beneficial for data integrity; when one of the links goes down, there is an alternate path to complete the task.
iSCSI performs the data integrity Cyclic Redundancy Check (CRC) on its header and PDU in the CRC mode. It can also use authentication protocols and encryptions such as Internet Protocol Security (IPSEC) that are negotiated at the beginning of the session. These however were not the focus of this research work.

Two main techniques are used to accessing remote data, namely file-access protocols and block-access protocols. In file-access protocols, remote files are made to appear as local files and in block-access protocols, remote disk blocks are made to appear as local. A block is the smallest amount of data that can be read from a disk or written to a disk by a device issuing surface number, cylinder number, and sector number.

The size of the disk sector is the size of the block. iSCSI, like most IP-based SAN [13] protocols uses the block-access protocol that is also used in the SCSI protocol. Unlike Network File System (NFS) where files may be shared among multiple-client machines, block protocols such as iSCSI support a single client for each volume on the block server.

As mentioned earlier, Targets are accessed as block devices. Hence, only one system can use the iSCSI device at a time as opposed to a Fiber Channel where multiple machines may access one file system. In iSCSI, each machine is allocated a chunk of one large device.

### 3.2 iSCSI Naming

All iSCSI devices have a unique address scheme namely, iSCSI Qualified Name (iqn) and the Institute of Electrical and Electronics Engineers Extended Unique Identifier-64 (eui), commonly known as the IEEE EUI-64 format.

A sample iSCSI address in iqn format [14] is given below:

```
```
The string iqn. identifies iSCSI initiator name as an iSCSI Qualified Name to distinguish it from an iSCSI initiator name in the "eui." format.

The notation 1994-05. is a date code in yyyy-mm format followed by a dot. This date MUST be a date during which the naming authority owned the domain name used in the iqn-formatted iSCSI initiator name. The naming authority for the Target is represented by giving the domain name in the reverse order. (i.e., com.Cisco) and then following a colon (:) is the Target. The notation :Target is an optional string that must comply with a character set and length boundaries that the owner of the domain name deems appropriate. The optional string must be preceded by a colon. The optional string may contain product types, serial numbers, host identifiers, or software keys i.e., Target.storage1.jfs to identify the iSCSI address for Target device 1 that has a Journal File System (JFS). IEEE EUI-64 format addressing can be accessed by following the link given in reference [15].

For example, if the Hewlett-Packard Company owned the domain name "stor.hp.com," registered in 2001, the iSCSI qualified names that might be generated by the Hewlett Packard Company appear in the following example:

<table>
<thead>
<tr>
<th>Type</th>
<th>Date</th>
<th>Naming Authority</th>
<th>String defined by &quot;stor.hp.com&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>“iqn”</td>
<td>“2001-04”</td>
<td>“com.hp.stor”</td>
<td>“initiator:master-host-ae12345”</td>
</tr>
</tbody>
</table>

The iSCSI address provides a mechanism for multiple Initiators or Targets to share a common IP network address, e.g., if the above address belongs to one of the three Target devices of a particular Target machine, all these Targets can be reached by an Initiator with a unique iSCSI name in conjunction with the IP address of the machine. Similarly, multiple Initiators or
Targets can be reached via multiple IP addresses. For example, if the above system had two NICS with two different IP addresses, then the Target identified by the above iSCSI Target address can be reached by an Initiator either through NIC1 or NIC2, which is the very basic idea to provide a redundant path to improve data integrity in a link-failure situation.

The iSCSI node is a single iSCSI Initiator or iSCSI Target and can contain more than one iSCSI node within a network entity, i.e., a device or gateway that is accessible from the IP network. This could be the local system where the Initiator or Target device resides. As shown in Figure 3.2, a network entity must have one or more network portal such as NICS. Each network portal is used by the iSCSI Initiators or Targets within that network entity to gain access to the remote IP network. The Network portal is a component having a unique TCP/IP network address used for the iSCSI sessions. Both Initiator and Target machine network portals are identified by their IP addresses and TCP port number pair. If the port number is not specified, the default port number 3260 will be used. These concepts are described in Figure 3.2 [16]. As shown, iSCSI Client and iSCSI Server are nodes in the network and a unique way of identifying each network node by using both IP Addresses and Port numbers.

![Figure 3.2. iSCSI network entity.](image)

16
3.3  iSCSI Session Establishment Procedure

For an iSCSI Initiator to communicate with a Target, it first needs to establish a session between them. The Network Entity may also contain one or more iSCSI Nodes, represented by unique iSCSI names. Since, Initiators establish iSCSI sessions with targets, session IDs are generated to uniquely identify individual conversations between specific iSCSI Nodes within the corresponding Network Entities. An Initiator logging on to a target, for example, 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), which likewise, in combination with its iSCSI name, gives that session a unique identity within the Network Entity in which it resides.

A session [17] has two phases: login phase and full-featured. Within a session, one or multiple TCP connections are established. The data and command exchange occurs within the context of the session.

The login phase is started when the Initiator establishes a TCP connection to the Target via a specified Target port. During this time, the Initiator and Target may authenticate each other and parameters are negotiated in the form of login request and login response. Once the login phase is completed, the session enters the full-featured phase where the Initiator discovers the available Targets.

Only after that phase is completed, can the SCSI I/O begin. In the Command phase, SCSI commands in the form of a Command Descriptor Block (CDB) is encapsulated in an iSCSI command PDU. The CDB describes the operation and associated parameters, e.g., the logical block address (LBA) and the length of the requested data in the form of a “MaxBurstLength” parameter, once it enters into the full-featured phase. During this phase, the data PDUs are
transmitted from an Initiator to a Target. The iSCSI session establishment and phases are shown in Figure 3.3.

![Diagram of iSCSI session establishment and login phase.]

**Figure 3.3.** iSCSI session establishment and login phase.

iSCSI Initiators and Targets come either as hardware components or as a software installation. The Hardware component is an iSCSI Host Bus Adapter (HBA), which is a variant to a normal Ethernet card with an SCSI Application Specific Integrated Circuit (ASIC) onboard to off-load all the work from the system CPU.

Software installation is done by a software driver that combines an NIC driver and an iSCSI driver to handle all iSCSI and other requests. Since IT managers we are expected to choose the least expensive and most productive solution, software Initiators and Targets without any added cost are chosen.
3.4 iSCSI Implementation

Several iSCSI protocol implementations are available commercially and as open-source free versions. Under the Linux platform, this thesis work used iSCSI Target implementation from iSCSI Enterprise Target (IET) running Fedora 2.6.16-1.2096_Fc5smp kernel, and a built-in Initiator software package available in Fedora Core 5 release. Using a Microsoft operating system environment, this research work employed Microsoft iSCSI Initiator 2.01 running in a Windows 2003 Server Environment.

3.5 iSCSI Read and Write

As previously discussed in iSCSI, all data and control traffic flows through the existing TCP/IP infrastructure. Hence, in order to perform a SCSI read request command, Initiator first creates an iSCSI PDU to be sent to the Target and once it reaches the Target machine, it sends back the data requested to be read. Since the iSCSI PDU is limited, a read request might need more than one PDU for a particular request. Meanwhile, flow control is handled by TCP to ensure that no buffer overflow occurs at the receiving end and that all data sent is acknowledged by the receiver.

Figure 3.4 indicates the normal iSCSI write procedure [18] without involvement of RDMA-enabled network adapters. The iSCSI protocol involves an exchange of commands and responses between Initiator and Target using iSCSI PDUs, which encapsulate SCSI CDB commands, status, and data.

In the SCSI write example, shown in Figure 3.4, the Ready to Transmit (R2T) PDUs perform, the role of SCSI flow control between Target and Initiator. These PDUs are issued by the Target device as buffers become available to receive more data. At the completion of the write, the Target issues the status and sense, indicating a successful transaction. The Target
controls the flow of data by indicating the amount of data it is able to receive via a transfer length field present in the iSCSI header field. If the Target does not respond, or responds with corrupted or incomplete test data, the Initiator may close the connection and establish a new one for recovery.

Figure 3.4. Normal SCSI write.

In the SCSI read example, shown in Figure 3.5, the Ready to Transmit (R2T) PDUs perform the role of SCSI flow control between Target and Initiator. These PDUs are issued by the Initiator device as buffers become available to receive more data. At the completion of the read, the Initiator issues status and sense, indicating a successful transaction. The Initiator controls the flow of data by indicating the amount of data it is able to receive via a transfer length field present in the iSCSI header field. The status of the SCSI data transport during reads
and writes is monitored through status and data sequence numbers via TCP, which encapsulates the iSCSI PDU.

Figure 3.5. Normal SCSI read.

All features of the iSCSI protocol discussed above provide an easier way to deploy SANs using an iSCSI software package that can be included with operating systems. Also, reduced latency due to the block level transfer of data and the lower-cost infrastructure provides easy for iSCSI to reach SAN solutions. The only downside comes from utilization of the CPU for processing TCP stacks for every I/O occurring in the Initiator or Target device, which requires accessing the memory bus a minimum of three times per data packet. This remains the barrier as the data transfer rates increase over TCP/IP, which can be overcome by using technologies such as RDMA.
CHAPTER 4

INTRODUCTION TO RDMA PROTOCOL

4.1 Introduction to RDMA

With the advancement in computing and storage technologies, information technology managers are forced to build a faster data center network which requires more central processing unit [19] power to process communication. Adding to this problem, TCP/IP data consumes significant memory bus bandwidth because this data typically crosses the memory bus three times, will be explained below. This overhead keeps the CPU busy by not allowing it to do other useful work, increasing latency, etc.

Also since most networks use a wide variety of links to interconnect devices in the network, the use of multiple system and peripheral [20] bus interconnects decreases compatibility, interoperability, and management efficiency with additional cost for equipment and special training. Hence, to increase efficiency and lower costs, the data center network infrastructure needs to be modified into a unified, scalable, high-speed framework.

This concept of a modified network infrastructure is relatively new, requiring high bandwidth and low latency that can move data efficiently over the network [21]. With the use of more efficient communication protocols, processors can be less burdened, giving them a chance to work in a more useful way.

Remote Direct Memory Access (RDMA) technology [22] is an emerging technology that promises to accomplish these goals. This data exchange technology is used by systems, applications, and storage to communicate directly over the existing infrastructure such as TCP/IP, which means packets are processed by the main system CPU. This creates a problem by
consuming more CPU resources for processing incoming and outgoing network traffic. In order to efficiently use CPU resources, RDMA is employed. According to the research, even with an increase in CPU power, the actual processing unit itself is still overburdened with system memory bandwidth, for efficient use of CPU processor cycles, system memory access must be reduced. Since memory bandwidth in modern architectures has rapidly become a scarce resource, gaining memory bandwidth is a major benefit.

4.2 RDMA Overview

As is commonly known, that TCP and IP are a suite of protocols [23] providing Intranet and Internet access, and every device in the network uses this suite of protocols to communicate with each other. Information is transmitted in the form of packets so that multi-vendor communication is possible.

Today, TCP/IP stacks are implemented in operating system software and because of this implementation all the packets transmitted or received are processed by the system’s CPU [24]. As a result, protocol processing of incoming and outgoing network traffic consumes CPU cycles, which can be more effectively used for other useful purposes. The amount of time consumed by the CPU during traffic processing will lead to a reduction in the number of processes handled by the CPU, increasing the overall delay in processing the packets.

With this already-burdened CPU, a finite amount of memory bus bandwidth in the system causes even more delay in transmitting or receiving packets over the network. Both the TCP/IP protocol overhead and limited memory bandwidth available hinder the deployment of faster Ethernet networks. The use of RDMA over TCP technology [25] can provide a chance to overcome these barriers by providing a chance to effectively use faster Ethernet networks.
RDMA technology was developed to move data from the memory of one computer directly into the memory of another computer with minimal involvement from their CPUs. This Zero Copy or Direct Data Placement (DDP) [26] capability provides the most efficient network communication possible between systems.

4.3 RDMA Layering Overview

The RDMA protocol [27] suite eliminates data copy operations and therefore reduces latencies by allowing an application to read and write data directly to the memory of a remote system with minimal demands on memory bus bandwidth and CPU processing.

4.3.1 RDMA

RDMA over TCP [28] technology involves a set of layers as show in Figure 4.1 performing different operations, the RDMA protocol uses RDMA write and RDMA read commands to transfer data between Initiator and Target devices, and sends them to the layer which resides below it namely Direct Data Placement (DDP). DDP protocol in turn segments the

<table>
<thead>
<tr>
<th>APPLICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDMA</td>
</tr>
<tr>
<td>DDP</td>
</tr>
<tr>
<td>MPA</td>
</tr>
<tr>
<td>TCP</td>
</tr>
<tr>
<td>IP</td>
</tr>
<tr>
<td>DATALINK</td>
</tr>
<tr>
<td>PHYSICAL</td>
</tr>
</tbody>
</table>

Figure 4.1. RDMA layering overview.
data obtained from the upper RDMA layer and also reassembles these segments into a DDP message. The Marker-Based Protocol Data Unit Aligned (MPA) protocol adds a backward marker at a fixed interval to the DDP segments, and it a length and Cyclical Redundancy Check (CRC) to each MPA segment.

The TCP transmits or receives traffic in terms of bytes, while DDP uses fixed protocol data units to transmit or receive data. Hence, to enable RDMA, DDP needs a framing mechanism for the TCP transport protocol which is taken care of by MPA. This facility allows the network interface to place the data directly in the receiver’s application buffer based on control information carried in the header. Hence, an efficient use of DDP comes only with additional usage of the MPA layer, allowing the system to avoid memory copy overhead and reduce the memory requirement for handling out-of-order and dropped packets.

4.3.2 Direct Data Placement

DDP allows Upper Layer Protocol (ULP) data [29], such as application messages or disk I/O, which are generated every time the data is read or written from or to Initiator or Target devices and are contained within DDP segments, to be placed directly into memory at the final destination without further processing, by the ULP. A DDP segment includes a DDP header and ULP payload, providing control and placement fields that define the final destination for the payload, which is the actual data being transferred.

A DDP message is a ULP-defined unit of data interchange that is subdivided into one or more DDP segments. This segmentation may occur for a variety of reasons, including to the respect of maximum segment size of TCP. A sequence of DDP messages is called a DDP stream.
DDP uses two data transfer models:

- **Tagged Buffer Model** – Used to transfer Tagged buffers between the two members of the transfer, namely the local peer and the remote peer. Tagged buffers are explicitly advertised to the remote peer through exchange of a steering tag (STag), Tagged offset, and length. An STag is simply the identifier of a Tagged buffer on a node, and the Tagged offset identifies the base address of the buffer. They are typically used for large data transfers, such as large data structures and disk I/O.

- **UnTagged Buffer Model** – Used to transfer UnTagged buffers from the local peer to the remote peer. UnTagged buffers are not explicitly advertised to the remote peer. They are typically used for small control messages, such as operation and I/O status messages.

### 4.3.3 MPA

The MPA creates a Framed PDU (FPDU) by pre-pending a header, inserting markers, and appending a CRC after the DDP segment. The MPA delivers the FPDU to the TCP. The MPA-aware TCP sender puts the FPDU$s into the TCP stream and segments the TCP stream so that each TCP segment contains a single FPDU. At the receiver, the MPA locates and assembles complete FPDU$s within the stream, verifies their integrity, and removes information that is no longer necessary. The MPA then provides the complete DDP segment to DDP.

### 4.3.4 TCP and IP

The TCP takes care in guaranteeing transfer of those segments received from the upper-layer protocols. It also takes care of flow control, congestion control, error correction, etc. Storage devices make use of a TCP connection to build iSCSI sessions to transfer data between Initiator and Target devices. Packets transmitted from Initiator to Target or vice-versa are acknowledged by informing the recipient of the packet sent. IP does the actual routing needed
to reach the destination over the network. Each device in the storage network identifies itself uniquely with the assigned IP address and iqn name identifier so that packets are routed to their proper destinations.

4.4 RDMA Data Transfer Operations

RDMA protocol provides seven different data transfer operations. RDMA information is included inside of fields within the DDP header. With a RDMA Network Interface Controller (RNIC), CPU’s of both source and destination devices are not involved in the data transfer operations, and RNIC is responsible for generating outgoing and processing incoming RDMA packets. In addition, the data is placed directly where the application advertised that it wanted it and is pulled from where the application indicated it was located.

4.4.1 RDMA Send

RDMA uses four variations of the send operation:

- **Send** – Transfers data from the data source (the peer sending the data payload) into a buffer that has not been explicitly advertised by the data Target (the peer receiving the data payload). The send message uses the DDP UnTagged buffer model to transfer the ULP message into the data Target’s UnTagged buffer. Send operations are typically used to transfer small amounts of control data where the overhead of creating a STag for DDP does not justify the small amount of memory bandwidth consumed by the data copy.

- **Send with Invalidate Operation** – Includes all functionality of the send message, plus the capability to invalidate a previously advertised STag. After the message has been placed and delivered at the data Target, the data Target’s buffer, identified by the STag included in the message, can no longer be accessed remotely until the data Target’s ULP re-enables access and advertises the buffer again.
- **Send with Solicited Event** – Similar to the send message except that when the Send with solicited event message has been placed and delivered, an event (for example, an interrupt) may be generated at the recipient end, if the recipient is configured to generate such an event. This allows the recipient to control the amount of interrupt overhead it will encounter.

- **Send with Solicited Event and Invalidate** – Combines the functionality of the Send with an invalidate message and the Send with a solicited event message.

### 4.4.2 RDMA Write Operation

The RDMA write operation is used to transfer data from the data source to a previously advertised buffer at the data Target. The ULP at the data Target enables the data Target Tagged buffer for access and advertises the buffer’s size (length), location (Tagged offset), and STag to the data source through a ULP-specific mechanism such as a prior-send message. The ULP at the data source initiates the RDMA write operation. The RDMA write message uses the DDP Tagged buffer model to transfer the ULP message into the data Target’s Tagged buffer. The STag associated with the Tagged buffer remains valid until the ULP at the data Target invalidates it or until the ULP at the data source invalidates it through a Send with Invalidate Operation or a Send with Solicited Event and invalidate operation.

### 4.4.3 RDMA Read

The RDMA read operation transfers data to a Tagged buffer at the data Target from a Tagged buffer at the data source. The ULP at the data source enables the data source Tagged buffer for access and advertises the buffer’s size (length), location (Tagged offset), and Stag [30] to the data Target through a ULP-specific mechanism such as a prior send message. The ULP at the data Target enables the data Target Tagged buffer for access and initiates the RDMA read
operation. The RDMA read operation consists of a single RDMA read request message and a single RDMA read response message, which may be segmented into multiple DDP segments.

The RDMA read request message uses the DDP UnTagged buffer model to deliver to the data source’s RDMA read request queue the STag, starting Tagged offset, and length for both the data source and the data Target Tagged buffers. When the data source receives this message, it then processes it and generates a read response message, which will transfer the data. The RDMA read response message uses the DDP Tagged buffer model to deliver the data source’s Tagged buffer to the data Target, without any involvement from the ULP at the data source.

The data source STag associated with the Tagged buffer remains valid until the ULP at the data source invalidates it or until the ULP at the data Target invalidates it through a Send with invalidate or Send with Solicited Event and Invalidate operation. The data Target STag associated with the Tagged buffer remains valid until the ULP at the data Target invalidates it.

4.4.4 RDMA Terminate

A terminate operation is included to tear down the connection when an error is encountered. The Initiator or Target device can disconnect using this signal in case either the Initiator or the Target device is not responding to the requests made to them.

4.5 RDMA Data Transfer Example

In a typical network data transfer, each packet accesses the memory bus three times before storing it. The first instance occurs when the receiving device writes the data to the device driver buffer. From the device driver buffer, the data is copied to the temporary buffer and finally copied to the application memory. This appears simple in a typical network scenario, but as the network grows and packets become bigger, the problem worsens and will have adverse effects, possibly not meeting the data transfer requirements.
With an increase in data rates, access to the memory bus is made more frequently. Below is a brief explanation of how a packet transfer occurs using RDMA. RDMA enables the applications to directly issue commands to the NIC without having to execute a kernel call. This is known as “Kernel Bypassing.” Then the RDMA-enabled NIC in the local system, transfers data from the local buffer to the RDMA-enabled NIC in the remote system. From here, the remote NIC places the data directly into its local memory without the intervention of its system CPU. Once the data is placed, the NICs inform their system CPUs of the completion of the operation.

An application requests a read or write request, and for every request made of this kind, two types of transfer phases take place, data and control operations. In RDMA, transfer memory needs to be registered before the application requests data transfer in the form of a special Tag called a Steering Tag or Stag, which is located in the RDMA header field. Hence, during the control operation, since memory is already registered, data transferred can be stored in memory regions advertised in the receiver or destination. Hence, the above-discussed advantages prove that RNICs provide better performance by reducing the load of the percentage of CPU utilized.

4.6 AMMASSO RNIC

The AMMASSO Gigabit Ethernet Network Interface Card [31] used in this thesis provides an implementation of the RDMA over TCP/IP-enabled NIC, which in turn provides low latency and high bandwidth on a Gigabit Ethernet network. This card supports the legacy sockets interface and the Cluster Core Interface Language (CCIL) interface. The CCIL [32] interface uses the RDMA layer and off-loaded TCP/IP on the NIC to transmit the data. On the other hand, the sockets interface still sends and receives the data through the traditional TCP/IP implemented
in the operating system kernel. The CCIL interface enables Zero-Copy and Kernel-Bypass data transmission.

4.7 Advantage of Using RDMA

This method of transferring data directly to or from application memory and data buffers to a remote memory over a network is known as Zero Copy networking. Since no CPU and cache overhead are involved, Zero Copy networking is particularly useful in applications where low CPU utilization, and low latency [33] in the network are desired. The RDMA engine notifies the CPU [34]. The addition of RDMA capability to Ethernet will simplify server deployment and improve infrastructure flexibility. As a result [35], the data center infrastructure will become less complex, easier to manage, and more adaptive. Hence, this immense potential of RDMA to improve communication performance while being extremely conservative on resource requirements has made RDMA the most popular of the current and future generation network infrastructures.
Typically, memory bandwidth is consumed by buffer copying because the data received from the network adapter did not arrive in the buffer that the application required. This buffer copy occurs both on the transmit or receipt of a packet. There is no known general-purpose algorithm for solving the receive copy problem for TCP/IP without the application being rewritten to accept buffers from the protocol stack instead of supplying buffers which is the causes the largest source of CPU utilization except for RDMA. Also, the interrupts generated when the adapter has finished transmitting or receiving posted data keep the CPU busy. Hence, the CPU offload can also solve latency bottlenecks, which can slow down the application and also can affect distributed applications running over the network. Latency in this case is a significant problem for client-to-server database communications, which have many outstanding transactions; therefore, the end-to-end latency of a single transfer is very critical.

During TCP implementation, data arriving on a TCP connection is first copied into temporary buffers then the TCP driver checks connection identification information, such as port numbers, and source and destination addresses, to determine the intended receiver of the data. The data is then copied into the receiver’s buffers. For SCSI data, there might be many pending SCSI commands at any given instant, and the received data must typically be copied into the specific buffer that was provided by the SCSI layer for the particular command. This entire procedure might require the receiving host to copy the data a number of times before the data reaches the final destination buffer. Such copies require a significant amount of CPU and
memory bus usage that would adversely affect system performance. Therefore, it is most desirable to place the data in its final destination with a minimum number of copies.

Currently, an I/O bottleneck has always been around the memory and the I/O subsystems and the formula for calculating I/O throughput from memory and I/O bus frequency is calculated as given in Equation 5.1. Hence, when I/O performance is particularly hindered because of the percentage of CPU utilization, this research will enhance the overall throughput of the system in terms of percentage CPU utilized or IOPS.

The RDMA protocol did not make quick enough progress towards standardization, so the iSCSI protocol could not make use of such a mechanism being available for increasing the overall throughput of the system. Once the RDMA proposal standardized all problems, such as Zero Copy transfers were answered and since then were largely put in use.

The information provided in an iSCSI data PDU header include the following: a transfer tag to identify the SCSI command and its corresponding buffer, a byte offset relative to the beginning of the corresponding buffer, and a data length parameter indicating the number of bytes being transferred in the current data packet. As described previously in section 3.5, we see that for each SCSI read or write command, a large number of flow control packets need to be sent from one end to the other and these packets need to be copied into temporary buffers and then to device driver buffers multiple times before storing the data at the destination.

Hence, each packet will generate multiple interruptions at the destination. In a network where there is enormous amount of data traffic, multiple copy operations are needed, and the system suffers with interrupt overhead, and thus lowering the system performance. Also, having multiple small packets means that the iSCSI protocol layer and the underlying communication layer must interact with each other multiple times to process these packets.
These interactions can also increase communication overhead. Without going through all these unnecessary procedures, RNICs enable the data to be placed at a destination without multiple copy procedures, with the help of RDMA semantics. In order to prevent this inherent performance degradation from occurring RDMA along with implementation of iSCSI was used. In iSCSI over RDMA, the iSCSI nodes use RDMA operations and the Initiator advertises the buffer using the identifier i.e, STag to the target when the SCSI Command for the data transfer is issued by the iSCSI layer. Hence, for a SCSI read command as shown in Figure 5.1, the STag identifies the tagged buffer into which data from the target will be directly placed by the initiator RNIC using the RDMA write operation. For a SCSI write command as shown in Figure 5.2, the STag identifies the tagged buffer on the Initiator from which data is directly transferred by the Target RNIC using the RDMA read operation.

As show in Figure 5.1 instead of using the normal SCSI read function as described in the SCSI section 3.5. The RDMA operations provide several advantages. The RDMA operations transfer data directly to buffers at the Initiator from the Target, since the memory regions are

---

![Figure 5.1. SCSI read with RDMA.](image-url)
already registered before the data transfer starts using RDMA. Hence, software flow control is not needed, thus eliminating multiple copy operations required in each data transfer operation. Hence, the interrupt frequency is reduced, thereby increasing overall system performance. As shown in Figure 5.2, instead of using the normal SCSI write operations from section 3.5, as described in the RDMA section, RDMA operations transfer data directly from the buffers at the Initiator to the Target since the memory regions are already registered before the data transfer begins. Hence, the need for software flow control and multiple copy operations are no more needed, thus improving the overall network performance by reducing the percentage CPU utilized.

If the buffers are contiguous, only one RDMA operation is needed for the entire SCSI read or write request. As a result, the software overhead of processing multiple packets is also eliminated.

5.1 Test Objectives

Understanding the sources of disk I/Os made for every data transfer requested by the Initiator or the Target helps to plan and configure nodes in SAN networks in a way that maximizes performance of the overall system. The objectives of this performance testing is
aimed to show that iSCSI in conjunction with RDMA improves throughput in terms of IOPS, which defines the access rate of storage devices at different transaction or block sizes compared to running only iSCSI. The observations made in this research work helps to understand how I/O performance changes with the variation in block size and helps to understand I/O requirements, to optimize storage devices. This research work focusses primarily on the I/O behavior, the percentage of CPU utilized which is generated in the log file generated by an IO Meter during data transfer between the Initiator and Target devices both in iSCSI and iSCSI over RDMA.

5.2 Testing Procedure

The following steps describe the general test procedure implemented in this research work:

a. Start all services, like IO Meter and iSCSI software’s on Initiator, Target and Manager systems.

b. Verify that cables used are not faulty, and check for proper connectivity between nodes using Ping utility.

c. Configure and enable the Remote Shell (RSH) program on all nodes in the network. This allows execution of a single command on a remote host without login to the remote host for performing RDMA operations.

d. Connect each of the participating Initiators to its iSCSI Targets.

e. Login to the iSCSI Target using iSCSI by mounting all the storage devices available on the Target device from the Initiator.

f. Once logged into the iSCSI Target device, tune the IO read sizes provided by the IO Meter.
g. Start the IO Meter on the Manager and the Dynamo on the Initiators.

h. Create the IO Meter .icf configuration files, and set the access specifications (request sizes) and the test setup parameter.

i. Start the test.

j. Name the output CSV file.

5.3 Theoretical Calculations

The generic formula used to calculate I/O throughput from memory and I/O bus frequency [36] is

\[
\text{I/O Throughput} = \frac{\text{Memory Bandwidth}}{\left( \frac{\text{Memory Clock}}{\text{I/O Bus Clock}} \right) + 2}
\]  

(5.1)

With the implementation of RDMA, the additional two memory cycles needed to transfer data are no longer needed, therefore the new I/O throughput is

\[
\text{I/O Throughput} = \frac{\text{Memory Bandwidth}}{\left( \frac{\text{Memory Clock}}{\text{I/O Bus Clock}} \right)}
\]  

(5.2)

Hence the difference in the I/O throughputs obtained in both the cases is calculated in terms of percentage as

\[
\text{Percentage} = \frac{\text{new Throughput} - \text{old Throughput}}{\text{new Throughput}}
\]  

(5.3)

An Example of a system might be an I/O bus speed of 80 MHz and memory clock speed of 400 MHz used in this test scenario, when these values are included in the above equations, the performance gain obtained because of Zero Copy implementation ranges from 20 to 30 percent.

5.4 Analysis of Test Results and Parameters used

An experiment was executed using the test bed setup as shown in Figure 5.3, the following for scenarios: iSCSI running without RDMA and iSCSI running over RDMA.
Performance measurements were made using the industry-standard IO performance analysis and testing tool known as IO (Input Output) Meter, which measures performance in terms of Input/Output per Second (IOPS) and percentage CPU utilization. Tests were carefully performed using IO Meter [37], the results predict the performance of complex applications running over the production networks. The tests measured the throughput performance, in terms of IOPS using RNICS provided by Ammasso, Inc. with various I/O request sizes provided by the IO Meter.
The IO Meter runs on the “Manager” running on Windows system and on the managed client system (iSCSI Initiator) for work load generation, i.e, generates read requests as requested by user. The IO Meter here used was version 2003.12.16.win32, run on the Manager system.

Both the IO Meter manager and managed systems, used workstations running Windows 2003 Server, Fedora Core 5 (kernel 2.6 smp) with dual 2.4-gigahertz (GHz) Intel Pentium Xeon processors, 1 gigabyte (GB) of PC2100 RAM, a RNIC copper gigabit NIC and a single 80GB SCSI hard disk drive. Microsoft’s iSCSI Initiator version 2.0 was used for all the test configurations. The iSCSI Target host, Dual 2.4-gigahertz (GHz) Intel Pentium Xeon processors, 1 gigabyte (GB) of PC2100 RAM, a RNIC copper gigabit NIC, and a single 80GB SCSI hard disk drive running Fedora Core 5 (Kernel 2.6 smp) was used.

Performance tests were run using, the IO Meter iSCSI Initiators, and Targets were connected to a 6 Port Ethernet switch using separate 100 Mbps copper connections. All systems were configured on the same subnet, and all traffic used standard 1,500 byte Ethernet frames. IO Meter [38] was configured to generate load on the iSCSI Initiators consisting of 100 percent sequential reads, etc.

During each test, the IO Meter Initiators or clients made sequential read requests using different block sizes. For example, when performing the 512-byte request size, the worker process made sequential write requests for 512 bytes of data. The test was conducted using the following request sizes (in bytes):

512B Reads, 1KB Reads, 2KB Reads, 4KB Reads, 16KB Reads, 32KB Reads, 64KB Reads

Results are represented graphically in Figures 5.4 and 5.5 for the test scenario shown in Figure 5.3.
Figure 5.4. IOPS in iSCSI and iSCSI over RDMA.

For the maximum number of sustained IOPS tests, the results in Figure 5.4, show that iSCSI over RDMA achieved its peak number of IOPS at a lesser value to the value obtained when running only iSCSI at a request size of 512 bytes and performing 100 sequential reads. The same trend was observed by changing request sizes varying in number as depicted in the Figure 5.4.

As shown in the Figure 5.4, there is decrease in number of IOPS required to transfer the same amount of data in iSCSI over RDMA compared to running only iSCSI because of the Zero Copy implementation used in iSCSI over RDMA. With RDMA implementation, as discussed in previous sections, multiple copy overhead was eliminated and the requirement for flow control was eliminated. Hence, the CPU interrupts were reduced and the number of IOPS required were also reduced. Overall, the test results shown in Figure 5.4 show that fewer number of IOPS are needed in iSCSI over RDMA compared to iSCSI, thus making the system more effective.
As shown in Figure 5.5, the percentage of CPU utilized in iSCSI is comparatively higher than that of iSCSI over RDMA because RDMA relieves the CPU time with the Kernel Bypass mechanism, thus providing efficient use of CPU cycles.

Calculations made in section 5.3, and from the results obtained coincide with one another, i.e., both show an increase in the performance of the system by 20 to 25 percent when RDMA was implemented. Hence, RDMA provides a chance to efficiently utilize CPU resources for running other important applications in SAN environments and reduces the processing delay in the system, thereby providing efficient use of all network resources.

The difference in the ratio of the number of IOPS for each request size, as plotted in Figure 5.4, is approximately between 20 and 30 percent i.e., iSCSI over RDMA provides a 20 to 30 percent increase in throughput in terms of IOPS. The difference in the ratio is calculated using equation 5.3 from the test results.
CHAPTER 6
CONCLUSIONS AND FUTURE WORK

The protocol iSCSI over RDMA was successfully implemented on the existing Ethernet network keeping all the objectives discussed in this research work.

- The major issues of CPU and memory bandwidth addressed in this work are as follows:
  - Zero Copy Receives—RDMA Writes place the data directly into the application buffer.
  - Protocol overhead is reduced.
  - Completion events are reduced.

- Performance improved in terms of IOPS required, and percentage of CPU utilized when data was transferred over the network.

- Since IP storage is based on industry standard protocols, like iSCSI, the RDMA significant performance gain can be achieved when the same is implemented in cluster sizes that are large in number.

- Storage networks implemented in this fashion effectively use an existing IP infrastructure and provide better results at a reduced price.

- There are few security concerns while implementing RDMA technologies in SAN environments, which might increase a chance to open memory on the network.

- Hence, a better security implementation should be developed to enhance the security of SAN environments using iSCSI over RDMA.
REFERENCES
LIST OF REFERENCES


