Evaluation for High Volume Data Transfer Mechanisms in Grids

Yi Zhu, Domenico Talia
{yzhuz}@cetic.be
{talia}@deis.unical.it
Universita Della Calabria
87036 Rende(CS), Italy

Alessandro Bassi
{Alessandro.Bassi}@Hitachi-eu.com
Hitachi Sophia Antipolis Lab
06560 Valbonne, France

Philippe Massonet
{phm}@cetic.be
CETIC, Charleroi, Belgium

CoreGRID Technical Report
Number TR-0178
October 14, 2008

Institute on Knowledge and Data Management

CoreGRID - Network of Excellence
URL: http://www.coregrid.net

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme

Project no. 004265

Evaluation for High Volume Data Transfer Mechanisms in Grids

Yi Zhu, Domenico Talia
{yzhuz}@cetic.be
{talia}@deis.unical.it
Universita Della Calabria
87036 Rende(CS), Italy

Alessandro Bassi
{Alessandro.Bassi}@Hitachi-eu.com
Hitachi Sophia Antipolis Lab
06560 Valbonne, France

Philippe Massonet
{phm}@cetic.be
CETIC, Charleroi, Belgium

CoreGRID TR-0178
October 14, 2008

Abstract

In the report, a set of benchmarks are designed to help us study mechanisms intended for large volume data transfer in high BDP (Bandwidth Delay Product) environment. As it is widely know that TCP - Transmission Control Protocol, which is the most commonly used protocol for data transfer in the Internet, is not suitable for high volume data transmission in computational Grids. Therefore, in order to solve this issue, different solutions have been proposed and tested in the report. First, experiment results analysis about application-level solutions is given. In addition, as Computational Grid is moving towards Enterprise Data Centers, some different types of data movement mechanisms are also used. Consequently, the report will also focus on these different types of data transfer mechanisms. By making a general review, realizing and analyzing these experiments of protocols, we can make a selection among these protocols for our future study and research in creating and developing a mechanism for transferring high volume data in a Grid environment.

1 Introduction

The advent of high-performance networks with low-cost, powerful computational engines has enabled the introduction of new distributed computing infrastructure termed Computational Grid. It pushes a wide range of advanced high performance distributed applications come to the fore, including distributed collaboration across the Access Grid, remote visualization of terabyte scientific data sets, large-scale scientific simulations, Internet telephony and multimedia applications, by interconnecting geographically distributed computational resources via very high-performance networks [1].

This research work is carried out for the CoreGRID IST project n004265, funded by the European Commission (Contract IST-2002-004265).

However, with the increase of network bandwidth and delay, the current network transport protocols, such as Transmission Control Protocol (TCP) which is most commonly used to achieve reliable data transfer, are becoming bottlenecks. Experts found that advanced distributed applications implementing in existing large-scale computational Grids often performs very poorly - only obtaining a small fraction of the available underlying bandwidth [1] and [2]. The reason for such poor performance is caused by TCP's inefficient over long distance high-speed networks. Networks characterized by links with high bandwidth (bits per second) and long round-trip delays (RTT in seconds) are often called high Bandwidth Delay Product (BDP) networks. As explained in RFC 1072 [3], Bandwidth Delay Product is commonly calculated as the bottleneck link bandwidth times the round trip time. In other words, the product is regarded as the amount of bits filled the "pipe" at any given time

As a matter of fact, when TCP is used in networks with high BDP, most of its performance drawbacks are caused by its own inherent shortcomings – window-based congestion control mechanism [4]. According to Lakshman and Madhow [5], TCP's congestion control mechanism is unfair to flows with different round-trip times (RTTs) - flows with high RTTs will get less bandwidth than those with low RTTs when they pass through a common bottleneck. This is to say, with the increase of link's RTT, the theoretical upper limit of throughput decreases [6]. In addition, the "slow start" mechanism takes a long time for TCP to reach a high available bandwidth over long distance high-speed networks [7]. Moreover, the AIMD (Additional Increase Multiplicative Decrease) algorithm, which is used by the congestion control mechanism to set the sending rate, is poor in discovering available bandwidth in high BDP networks [8]. There are more issues for TCP, all these problems expose TCP, the standard transport protocol used in the Internet, is inadequate for a high-bandwidth, high delay network environment [8] and [9].

In order to obtain better performance in high BDP networks, researchers have been proposing some solutions, including variants of TCP, new transport protocols and application level schemes. Due to the fact that variants of TCP and new transport protocols have inevitable shortcomings [10], this report mainly focuses on application level schemes. Moreover, as most of these application level techniques are widely used in "traditional" Grids where data storage is often tied with Grid nodes tightly as directly attached. However, as Computational Grid is moving towards Enterprise Data Centers, it will face difference resource and service management challenges. In order to overcome these issues, experts come up with several different types of data transfer mechanisms, such as iSCSI (internet SCSI), FCoE (Fiber Channel over Ethernet), and etc.

To have a brief view of these new data movement tools, this study is performed by selecting protocols and using them to transfer large volume files between experimental testbeds. In the experiment, a set of benchmarks were designed to help us analyze protocols intended for high BDP environment. By comparing and contrasting these protocols, a clearer view about these protocols can be obtained. The report is organized as follows: In section 2, we discuss protocols used in "Traditional" Grids that means application-level techniques are discussed and analyzed in the section. Section 3 provides descriptions and observations of storage protocols, like iSCSI. Finally, the conclusions and directions for future work are presented in section 4.

2 Mechanisms for High Volume Data Transfer in "Traditional" Grids

As described in the introduction of the report, application-level techniques are the reliable solutions to the limitations appeared in high BDP networks. According to the transport protocol, it can be classified into two categories - TCP based solutions and UDP based solutions [11]. The first category, such as GridFTP, bbFTP and Psockets, employs TCP connections to move data. As indicated by Anglano and Canonico, "It circumvents TCP problems by utilizing parallel streams. " [11]. As a result, an achieved aggregate congestion window meets the requirements of high BDP networks. UDP based solution, which is provided to be as another scheme for high performance data transfer, uses UDP to transfer data, coupling with rate-based control algorithms for the transmission speed match [11]. Meanwhile, some of them employ congestion and flow control algorithms during transmission in order to keep the loss rate at a low level. In the section, some descriptions of protocols are given, followed by the methodology and test bed setup for these experimental protocols. Further on, experimental results and performance discussion are provided at the end of the section.

2.1 Protocols Descriptions

This section mainly discusses protocols used for high volume data transfer over high BDP networks such as Psockets (Parallel Sockets), UDT (UDP-based Data Transfer Protocol), and RBUDP (Reliable Blast UDP). For more protocols descriptions, please refer to the report TR-0121 [12].

2.1.1 Psockets (Parallel Sockets)

As indicated by Sivakumar, Bailey, and Grossman [13], by using Psockets, applications can achieve almost optimal utilization of the network bandwidth without tuning the TCP window size for the best value. The basic idea behind Psockets is that it opens multiple socket connections between the sender and the receiver. Then it divides the data that needs to be transferred into equal partitions. Here, the number of partitions is determined by the number of open sockets. After that, the data partitions are sent asynchronously over multiple socket connections. According to Sivakumar, Bailey, and Grossman [13], Psockets approach can be broadly viewed as striping data over arrays of disks.

For more Psockets discussion, please refer to TR-0121 [12]. In the experiment, Dataspace version 3.1 released December 6, 2003 was used for Psockets library.

2.1.2 UDT (UDP-based Data Transfer Protocol)

UDT is an UDP-based data transport protocol, not only for QoS-enabled networks but for shared networks as well [14]. It can achieve efficiency, fairness, friendliness and stability by employing window-based flow control and rate-based congestion control mechanisms [15]. According to Gu and Grossman [15], with the help of window-based flow control, UDT can limit the number of unacknowledged packets to be sent out before the sending side receives any control information recording congestion. In other words, UDT provides an approach to avoid packet loss over high BDP networks. In comparison to window-based flow control which is for the improvement of the protocol efficiency by reducing packet loss, rate-based congestion control mechanism helps to control the throughput of UDT flows by means of adjusting the inter-packet transmission time [7]. As a result, fairness bias caused by different network delays can be eliminated. More specifically – when UDT coexists with TCP, TCP dominates most of the bandwidth in low BDP networks, whereas, in high BDP environments, UDT utilizes the bandwidth that TCP fails to obtain, as TCP is inefficient in high BDP, but works quite well in low BDP environments [15].

In UDT protocol, each UDT entity has two parts: a sender and a receiver. The sender in UDT entity is mainly used for data packet sending, while the receiver is for data packet receiving, control packet (which includes flow control and congestion control) triggering and processing, and also for timer expiration detection [7]. In Figure 1, a UDT entity A sends data packets to UDT entity B, while control packets are returned from B's receiver to A's receiver, which is different from data packets sending (from A's sender to B's receiver).

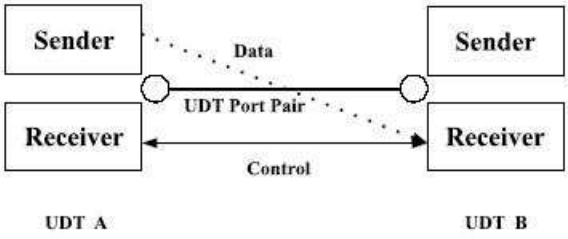

Figure 1: UDT Entity in the Protocol
source: Gu, Y., and Grossman, R (2003) 'Using UDP for Reliable Data Transfer over High Bandwidth-Delay Product Networks'

For more UDT protocol descriptions, please refer to the report TR-0121 [12]. As UDT library is an open source, the test presented in the report is bases on UDT version 4.2, released March 19, 2008.

2.1.3 RBUDP (Reliable Blast UDP)

RBUDP is a protocol only used for dedicated or QoS-enabled high bandwidth networks [16]. In order to fully utilize network bandwidth during bulk data transfer, it not only discards TCP's slow start and congestion control mechanism, but aggregates acknowledgements as well. It is totally different from UDT – in UDT protocol the slow start happens at the beginning of a UDT connection, once the congestion control mechanism is available, the slow start stops [7], According to He, Leigh and Yu [16], in RBUDP, acknowledgements are aggregated and sent at the end of every transmission phase instead of per window of transmitted data. More specifically, in the transmission phase, the sender transfers the whole data to the receiver via UDP datagram at a user-specified sending rate. After collecting a signal - DONE (means no more UDP packets), the receiver sends back a list of all missing packets via a TCP connection. Then these lost packets are retransmitted. The process is repeated until no more packets need to be retransferred [11]. Figure 2 illustrates the process of data transfer.

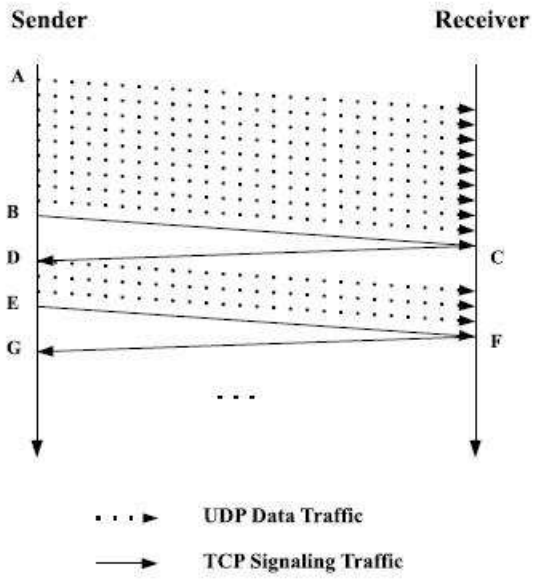

Figure 2: RBUDP Data Transfer
source: He, E et.al (2002) 'Reliable Blast UDP: Predictable High Performance Bulk Data Transfer'

As it is widely known, UDP is an unreliable protocol - some datagrams may be lost during the transmission, therefore the receiver should have a tally of the packets that are received to make sure which packets need to be re-transferred. In addition, in order to minimize packet loss, the rate should not be larger than the minimum bandwidth of the link [16]. For more detailed information about RBUDP, please refer to TR-0121 [12].

Since RBUDP was proposed on the base of QUANTA (the Quality of Service Adaptive Networking Toolkit) which is for an aggressive bulk data transfer scheme intended for high bandwidth, dedicated or Quality of Service-enabled networks. Therefore, in the experiment, QUANTA version 0.4 released April 14, 2004 was tested for the protocol RBUDP analysis.

2.2 Experimental Setup and Methodology

In the section, the testbed setup and experimental methodology for protocols mentioned above are demonstrated. Additionally, the experimental measurement tools are also described in the section.

2.2.1 Experimental Setup

The data is obtained from two cluster environments: CETIC cluster that is located in Belgium, and the cluster in the University of Calabria (Unical), Italy. In the CETIC cluster system, the test platform is composed of two nodes, AMD

bi-opteron 250 2.4GHz with 4GB RAM running Linux Centos 5.2. The bandwidth between two nodes is 1Gbps with 0.1ms RTT (Round Trip Time). The testbed in the Unical cluster environment also consists of two nodes. They are dual Intel Xeon 3.0 GHz Processors with 2GB RAM running Linux Redhat Enterprise Linux ES release 4. The two nodes are connected through a 1 Gigabit Ethernet with 0.076ms RTT. Results listed in the report are obtained from the connection between nodes in CETIC cluster and nodes in Unical cluster (52ms RTT in the test). Moreover, in order to richen the evaluation for high volume data transfer mechanisms, data gathered in low BDP networks, – 0.1ms (nodes in CETIC domain) and 0.076ms (nodes in Unical domain) are analyzed together in the report.

2.2.2 Experimental Metrics

The metrics is designed by identifying desirable characteristics of high BDP protocols. It mainly includes: CPU load, network capacity utilization, data transfer time, fast adjustment to network capacity, the network stability for the data transmission.

1. `CPU load:` CPU utilization is measured on both the sender and the receiver side during the protocol experiment.
2. `Network capacity utilization:` The network capacity utilization for a sustained period time is desirable in the experiment. In order to obtain this metric, 100GB of memory to memory transfer is decided to use. The buffer size in the sender and receiver are set to 400 MB, because of the hardware limitations. The maximum bandwidth is defined as the bandwidth achieved by the protocols for the 100 GB transfer. The results show the average of 5 times for 100GB transfers.
3. `Data Transfer Time:` Transfer time for the data of 100 GB is measured in variant environments (different Round Trip Time).
4. `Fast adjustment to network capacity:` The fast adjustment to network capacity means how fast a protocol can reach its network capacity. It is the time to maximum bandwidth. It can be obtained by measuring data sizes at two specified percentage of maximum bandwidth, such as 50 percent, 90 percent. Also the time to maximum bandwidth also needs to be measured.
5. `The network stability for the data transmission:` By modifying the contents of the memory buffers before sending, and confirming the integrity of the data buffers, the metrics can be succeed obtaining.

2.2.3 Measurement Tools

According to Tirumala et.al [17], Iperf is developed as a modern alternative tool to measure maximum TCP and UDP bandwidth. By tuning parameters such as TCP window size or UDP bandwidth, users can achieve the desired maximum TCP or UDP bandwidth. In the experiment, the Iperf version 2.0.2, released on May 2005 is used.

2.3 Experimental Results and Performance Discussion

The experimental results are discussed in the section. Moreover, on the base of these results, the performance comparison among variant protocols (standard TCP, PSockets, RBUDP and UDT)is provided.

2.3.1 CPU Utilization

As shown in the figure listed below, protocols chosen for the experiment, not only with low RTT but also with high RTT, consume more CPU resource than TCP does. In addition, different from Psockets, and RBUDP, UDT costs more CPU resource at the sending side. There is an interesting observation during the experiment, both senders and receivers consume a huge amount of CPU resource, especially, at UDT sender side in Unical domain. The CPU utlization nearly reaches 99 percent. With the help of Dugan, J.M [18] and Barkakaty, U [19], we suspect the reason for CPU's gratuitous large consumption during high volume data transfer is due to the inherent bug of Iperf version 2.0.2.

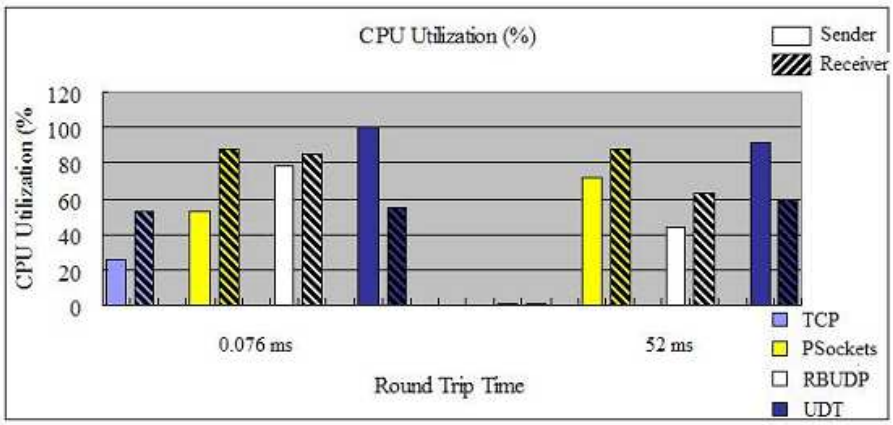

Figure 3: CPU Utilization

2.3.2 Maximum Bandwidth and Data Transfer Time

From figure 4, the maximum bandwidth for variant protocols with different RTT can be clearly discussed. For Psockets protocol, the maximum bandwidth gathered in the environment with low RTT is nearly similar as the TCP bandwidth measured by Iperf. However, in the high RTT environment, the peak bandwidth is remarkably lower than the TCP bandwidth gathered by Iperf. In regard to RBUDP, it performs worse with the increase of RTT. Also, as shown in the figure 4, the maximum bandwidth achieved by UDT in the high RTT environment (852 Mbps) is noticeably lower than the UDP bandwidth obtained by Iperf. In addition, the untuned TCP exhibits miserable poor performance in high RTT environment, dropping to 5.37 Mbps.

Moreover, as expected, the performance for untuned TCP drops with the increase of RTT. From figure 5, it can be obviously observed that the 100 GB data transfer time for untuned TCP is great higher than the time achieved by other protocols - All protocols have the same order of magnitude for 100 GB data transfer time, comparing with time required by untuned TCP (two orders of magnitude greater).

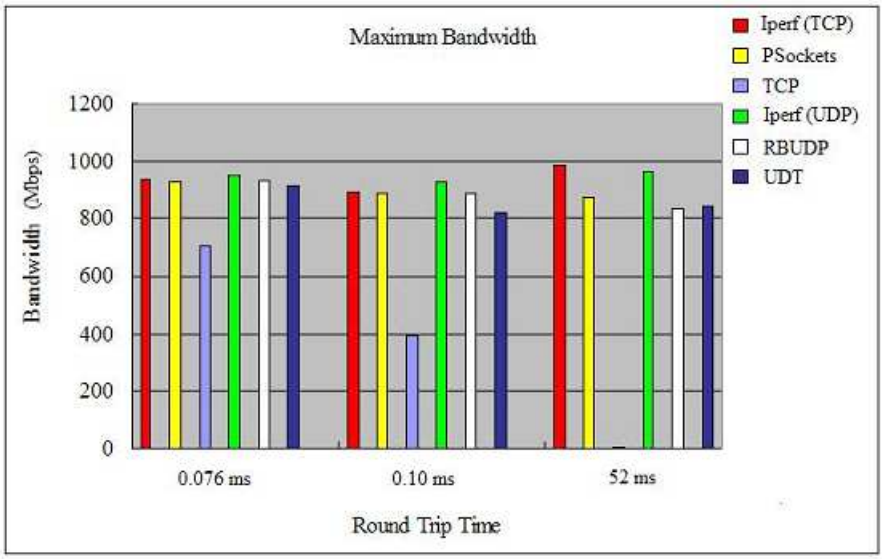

Figure 4: Maximum Bandwidth

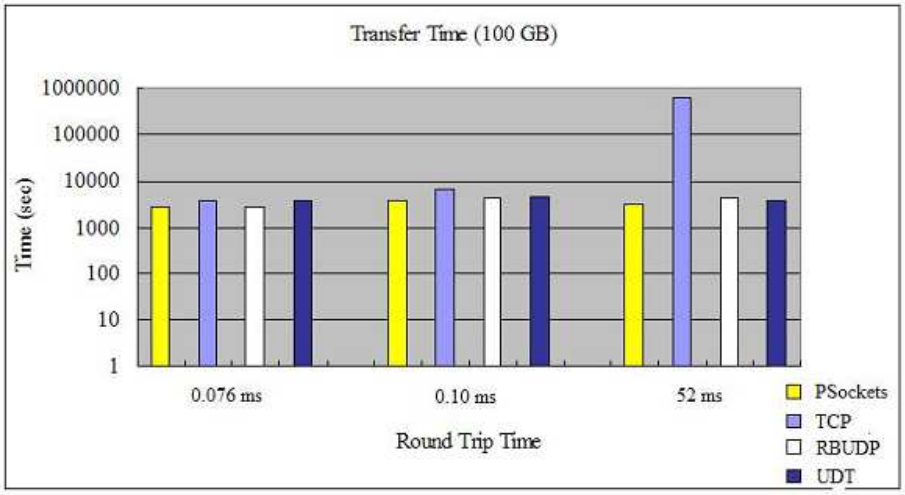

Figure 5: 100 GB Data Transfer Time

2.3.3 Fast Adjustment to Capacity

In comparison with the untuned TCP and UDT, the time required by Psockets to maximum bandwidth in high RTT environment is less (Figure 8). In other words, Psockets is able to reach maximum bandwidth without much data transfer. One interesting point is although untuned TCP has much lower maximum bandwidth than Psockets does in high RTT environment, its time to maximum bandwidth is still higher than the time needed by Psockets - almost one order of magnitude greater. In addition, from Figure 6 and Figure 7, for both Psockets and untuned TCP, the data size for 90 percent maximum bandwidth in all environments (not only with low RTT, but high RTT as well) is about one order of magnitude than it is at 50 percent maximum bandwidth.

However, with regard to UDT, it reaches maximum bandwidth fairly slower than the other two protocols do (Figure 8), because of its highest maximum bandwidth among three protocols. Another interesting thing is that the time required by UDT and untuned TCP to maximum bandwidth in high RTT environment are on the same order of magnitude, although UDT's maximum bandwidth is extraordinarily higher than untuned TCP's. Moreover, in the environment with high RTT, UDT requires one order of magnitude more data to reach 50 percent and 90 percent maximum bandwidth (Figure 6 and Figure 7). All the results listed above imply UDT may not reach its potential when transferring small files, not achieving maximum bandwidth before the termination of files transfer.

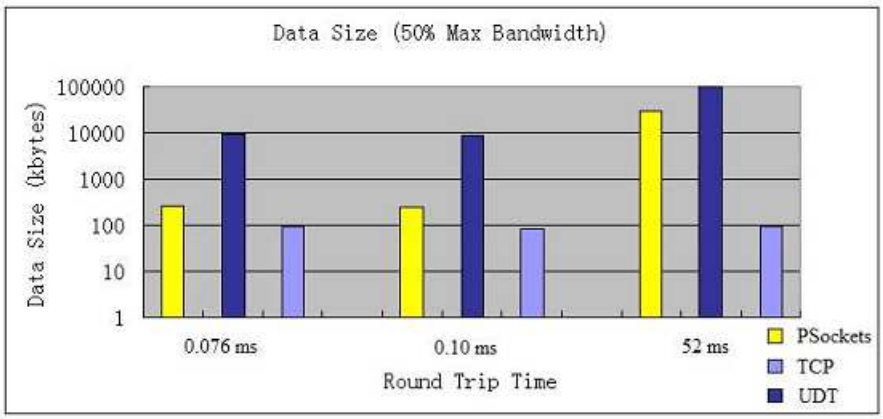

Figure 6: Data Size (50 percent)

Because of the sending rate for RBUDP is decided by user, its results about fast adjustment to capacity are not

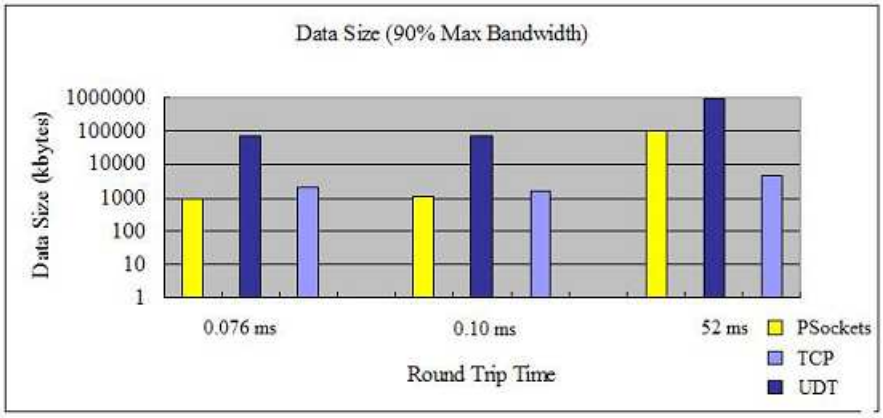

Figure 7: Data Size (90 percent)

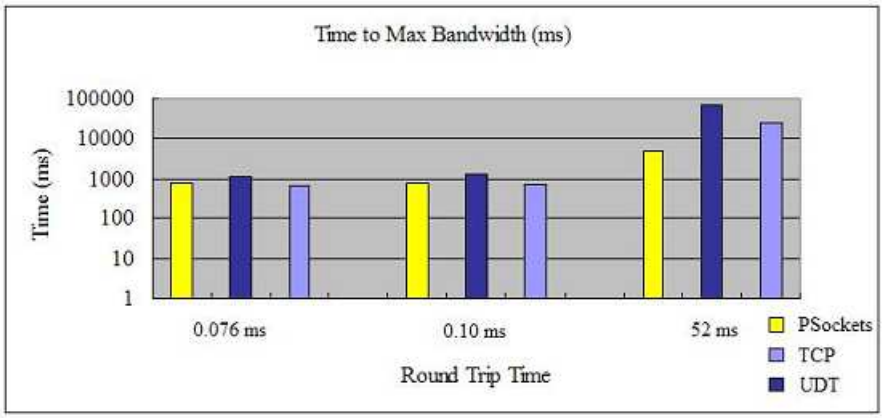

Figure 8: Time to Max Bandwidth

listed in diagrams.

In general, comparing with the results obtained in environments with high RTT and low RTT, more data and more time are required for all protocols in the environment with high RTT in order to reach maximum bandwidth.

2.3.4 The Rest Results and Analysis

Besides the discussion provided above, there are also some performance results obtained from the experiments need to be presented. In the following part, these points are pointed out respectively.

1. `Psockets protocol:` In the environment with high RTT, 256 parallel sockets are firstly used for data transfer; however, the results are poorer than the results by using 8 and 16 parallel sockets. It is observed that the protocol performance is not proportional to the open parallel sockets - 8 parallel sockets in the experiment seems to be optimal. In addition, from the experiment, it also can be easily observed that the whole bandwidth is not linear with the number of open sockets (which should be in theory) - the maximum bandwidth is 2.64 Mbps with one open socket, however, the whole bandwidth is around 600 Mbps with 256 parallel open sockets, less than the theoretical value. Neither of these two issues appeared in the experiment has the answer at moment.
2. `RBUDP protocol:` From the experiment, we can find in the environment with high RTT, the maximum bandwidth reached by RBUDP is lower than the bandwidth achieved by Psockets. In addition, as RBUDP is able to instantaneously adjust to the network capacity, it is easily for user to send data with peak capacity as

soon as the transfer stars. In terms of stability, error data are rarely detected by confirming the integrity of the transferred data.

3. `UDT protocol:` In comparison with Psockets and RBUDP, there is no need for UDT to have some dynamic variables set by user, that means it has the ability to decide the network capacity and also to adjust its rates. In the experiment, there is an interesting observation – UDT can reach a bandwidth slightly higher than the final maximum bandwidth, then it decreases gentlely over time. In addition, UDT is found to be a reliable protocol - no any integrity issues. However, there are several drawbacks discovered in the experiment - UDT has the slow start mechanism as well, and also it reaches maximum bandwidth slowly. In general, UDT does not provide more out-performance than Psockets or RBUDP does in the experiment.

2.4 Summary and Conclusion

In the experiment, an evalution for high volume data transfer mechanisms are provided. From the report, it can be easily found PSockets can reach a maximum bandwidth of 875 Mbps, RBUDP can achieve 837 Mbps, and the highest bandwidth for UDT is 852 Mbps. However, the maximum bandwidth for standard TCP is 5.37 Mbps.

As it is pointed out in the report, TCP has its inherent shortcomings over high BDP networks such as the window-based congestion control mechanism, slow start phase, and etc. From the resutls, it can be observed, in comparison with TCP, PSockets aggregates the window sizes of multiple sockets to address the issue of the default window size. RBUDP overcomes the issues of slow start and default window size. Moreover different congestion control mechanism is utilized in RBUDP. In addition, UDT also has significant improvements to achieve high bandwidth.

However, there are still several issues need to be addressed. The first and most important one is fairness – fairness between the application level scheme and TCP, and fairness to other application level scheme. In addition, the effect of packet loss is the other issue that should be focused. Packets loss from higher capacity links to lower capacity links should be considered in future.

3 Mechanisms for High Volume Data Transfer in Enterprise Data Centers

As proposed in the introduction, application level schemes have the ability to perform better in high BDP networks. However, most of them are widely utilized in “Traditional” Grids. In these environments, data storage is often tied with the Grid nodes as directly attached. With computational Grid moving towards Enterprise Data Centers, difference resource and service management challenges is coming to the fore. In other words, it seems to be challengeable for application level schemes to do data movement in these data centers. In consequence, several different types of data movement mechanisms such as iSCSI (Internet SCSI), FCoE (Fiber Channel over Ethernet), and FC (Fiber Channel) are proposed to access data. In the section, the experiment for Enterprise Data Storage solutions are described and analyzed. The rest of the section is organized as follows: The first part points out the main characteristics of the protocol tested in the experiment. After that, the experimental testbed and methodology are given. In the third part, the test results are highlighted and analyzed. Finally, the summary for Enterprise Data Storage solutions is presented.

3.1 Data Transfer Mechanisms Description

As it is widely known, SAN (storage area network) and NAS (network attached storage) are two storage solutions to provide access to different types of data. According to Meyer [20], SAN is widely for high-volume block-oriented data transfer, while NAS is usually for data access at the file level. In other words, SAN supports a range of applications, including providing storage for NAS applications. NAS storage is typically limited to applications that access data at the file level [22]. Regardless of their differences, both of them play important roles in today’s enterprises - providing many advantages over traditional server-attached storage implementations. For more detailed characteristics of SAN and NAS, please refer to the report TR-0121 [12].

Due to the testbed we have in the experiment, we focused our storage data transfer mechanisms test onto iSCSI (internet SCSI). Therefore, in the following section, the descriptions of iSCSI are given.

iSCSI is a network data transfer mechanism, which allows mapping SCSI protocol over TCP/IP networks. Comparing with other network storage protocols, “it only requires Ethernet interface (or any other TCP/IP - capable network) to operate, which decreases the cost of storage centralization” [21]. With the popularity of gigabit Ethernet, building iSCSI-based storage area networks has become a less costly way to take the place of Fiber Channel-based SAN.

According to Senapathi and Hernandez, some experts worried about its worse performance as its heavy overhead added by the TCP/IP protocol, however, with the introducing of TCP Offload Engine (ToE) - that is “mainly used in network interface cards to offload the processing of the entire TCP/IP stack to the network controller” [23], iSCSI-based SANs have shown excellent performance. In iSCSI-based storage, an iSCSI initiator is allowed to connect to remote targets such as disks, and tape drives [21]. Here, the iSCSI initiator, in client/server terminology, is similar to a client, while the iSCSI target, which provides block level access to storage media, is similar to a server [24]. The main difference between client/server model and iSCSI initiator/target system is that many clients have the ability to simultaneously access the same files offered by a single server, whereas, complicated co-operations are required for iSCSI initiator to synchronize accesses to the same file [21]. In other words, it is a difficult for multiple hosts to have simultaneous access to a single device in an iSCSI-based storage. Figure 9 graphs a part of iSCSI architecture model. For more detailed iSCSI characteristics descriptions, please refer to TR-0121 [12].

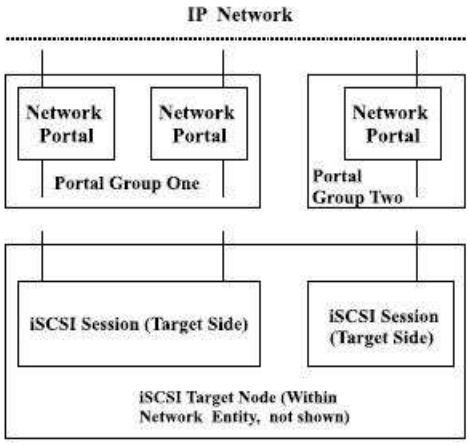

Figure 9: iSCSI Architecture Model
source: Satran, J et.al (2004) ‘Internet Small Computer Systems Interface (iSCSI)’

In the experiment, the open-source Linux-iSCSI version 4.0.2 released March 15, 2005 was employed as the iSCSI initiator, a Linux iSCSI Enterprise Target version 0.4.16 released March 18, 2008 was used for the iSCSI target.

3.2 Experimental Setup and Methodology

One of factors that come to attention is that iSCSI can be implemented in an exiting network with few or no specialized hardware involved. In the section, the factor is demonstrated by the experimental testbed setup. Furthermore, the experimental metrics and measurement tools are proposed in the section.

3.2.1 Experimental Setup

The storage testbed used for the experiment consists of two nodes, node19 and node20 in CETIC cluster. Both of them are AMD bi-opteron 250 2.4GHz with 4GB RAM running Centos 5. In addition, Gigabit Ethernet Controller NetXtreme BCM5703 is mounted on both of node 19 and node 20.

As shown in Figure 10, Node19 is chosen to be as ”server” (target machine), and node20 is considered as ”client” (initiator machine). The iSCSI initiator and the iSCSI target are directly connected with a gigabit Ethernet connection. Therefore, the testbed setup provides a network with no switch delay. Furthermore, data access from the 70 GB node

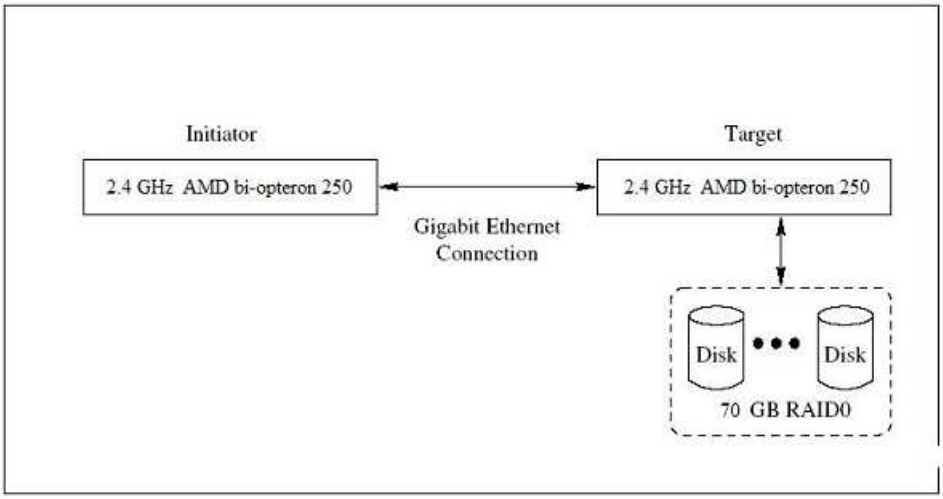

Figure 10: Experimental Setup

storage can be thought as accessing data in the storage system, since the 70 GB node storage was manually mounted as soft-RAID-0 array storage.

3.2.2 Experimental Metrics

Metrics for the experiment is designed by identifying desirable characteristics of network storage protocols. It mainly includes: Network throughput, RAID disk throughput, and etc. As it is widely known, the network performance is impacted by different variables, such as socket buffer size, frame size, I/O block size, filesystem size and etc. By comparing and contrasting every metric under different environments, it would be better to find the iSCSI performance in variant environments.

1. `Network throughput:` Network throughput was tested in the environment of different socket buffer size, but with the same frame size during sequential reads and writes. Also, results were obtained in the environment of different frame size with the same socket buffer size during sequential reads and writes.

2. `RAID disk throughput:` RAID disk throughput was also gathered in the environments of different socket buffer size and different frame size during sequential reads and writes.

In additional, by varying data transfer size, the impact of filesystem was also observed.

3.2.3 Measurement Tools

The Iometer benchmarking kit is an I/O subsystem measurement tool for single and clustered systems [25]. By performing I/O operations, it can be thought as a workload generator. Meanwhile, it can be used as a measurement tool, as it examines and records the performance of its I/O operations, and the impact on the system. Interesting enough, Iometer can be configured to emulate the disk or network I/O load of any benchmark, or can be used to generate entirely synthetic I/O loads [25]. It can generate and measure loads on single or multiple systems.

The Netperf benchmarking can be used to measure various aspects of networking performance. As indicated by [26], its primary focus is to analyze the raw performance of TCP/IP networks on the base of its robustness and ability to segment data into specific block sizes. The characteristics carried by Netperf allows user to compare the network performance with the overall block transfer performance of iSCSI.

Finally, the last benchmark used in the experiment is Bonnie. It provides the ability to measure sequential I/O access to a file system, and also random seek rate [27]. By using Bonnie, the sequential I/O is measured per character and per block. More specifically, Bonnie initially creates a file of predefined size and then performs sequential write

per character and then per block. After that, Bonnie does sequential read also per character and per block. At last, the benchmark performs random seek operations [27]. In the experiment, the sequential file I/O data is obtained for user to compare iSCSI's performance with a filesystem.

3.3 Experimental Results and Performance Discussion

As it is widely known, within storage area networks, a number of different factors have critical impact on network performance. In the following part, several tested variables such as socket buffer size, and Ethernet frame size are discussed.

3.3.1 The Impact of Buffer Size on Throughput Performance

As large socket buffer are required to achieve high performance in large BDP (Bandwidth Delay Product) networks. In order to show the impact of different socket buffer size, it was determined that measuring the performance with the largest socket buffer that is allowed by the used operating system, and the system's default buffer size. Figure 11 presents the experimental results for sequential reads by using largest socket buffer.

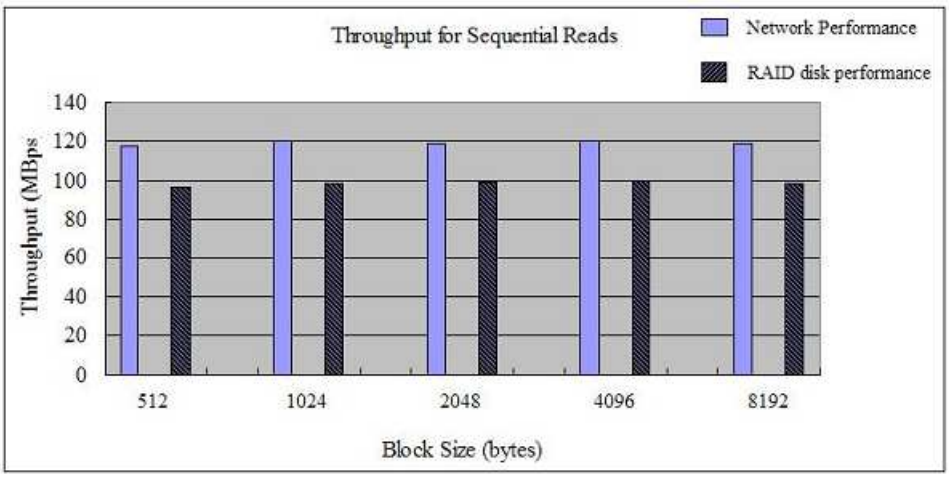

Figure 11: iSCSI's behavior during sequential reads by using largest socket buffer

In the figure 11, it can be easily observed that the peak throughput for the network is around 120 MBps with different I/O transaction size from 512 bytes to 8192 bytes. Here, the reason for the throughput limitation is because of the test bed PCI architecture which can yield throughput no more than 133 MBps theoretically. In comparison with the network performance, the maximum throughput for RAID disk is about 100 MBps.

Figure 12 shows the iSCSI behavior results for sequential writes. The network and RAID disk performance can maximum reaches around 120 MBps and 101 MBps respectively. One interesting observation in the experiment is that when I/O transaction size is set to 512 bytes, the RAID disk performs extremely poor - roughly at 4 MBps. The reason for this poor performance still needs to be analyzed.

In comparison with iSCSI's behavior by using largest socket buffer, the system performance under the default socket buffer size is quite similar. Figure 13 and Figure 14 show the measurements with the system's default buffer size.

By comparing and contrasting the results with different socket buffer size, we can find except the network performance - experiment with the default buffer size has about 3 MBps improvement, no other results have great difference, including the extremely poor performance when block size is 512 bytes.

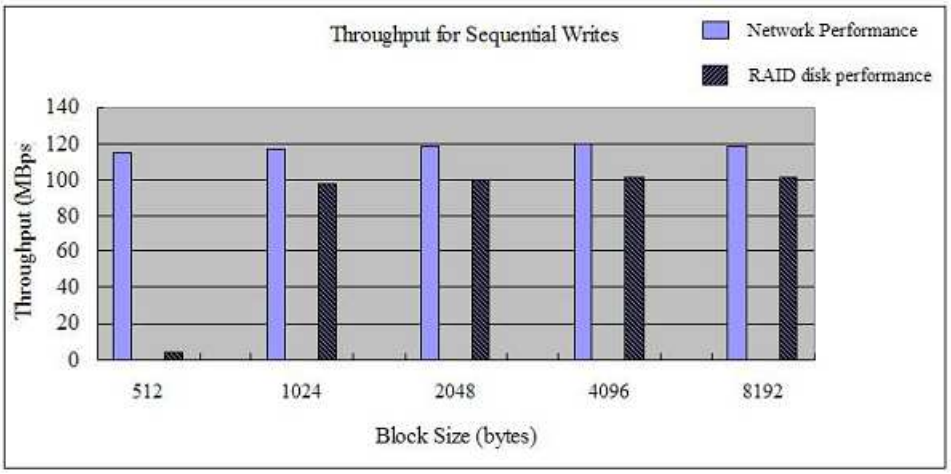

Figure 12: iSCSI's behavior during sequential writes by using largest socket buffer

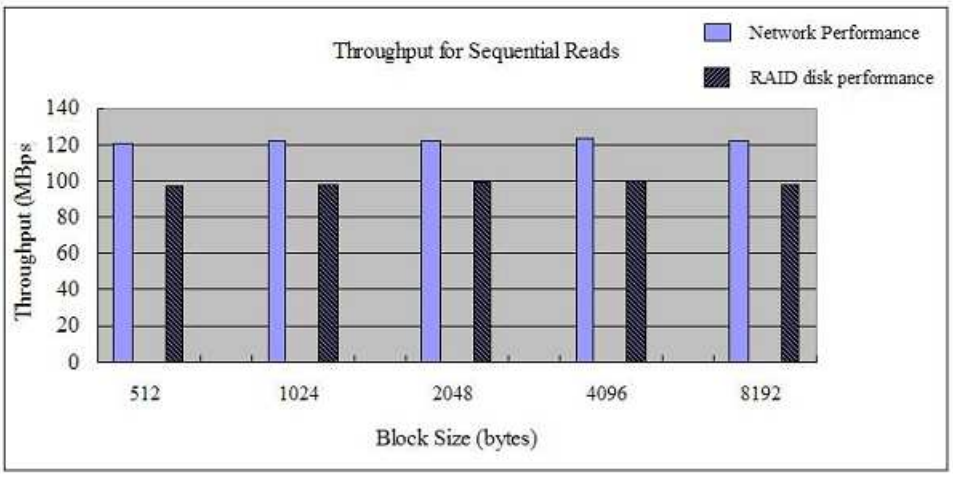

Figure 13: iSCSI's behavior during sequential reads by using the buffer set to the default value

3.3.2 The Impact of Frame Size on Throughput Performance

As indicated by Chase, Gallatin and Yocum [28], Gigabit Ethernet is able to provide the frame from 1.5 KB to 9 KB, which has the significant impact on network performance with high BDP. In order to find out its impact, the same system configurations with different frame size were conducted in the experiment. In comparison with the performance results by using 9 KB Ethernet frame size and 1.5 KB Ethernet frame size, it is interestingly found that all experimental results drop off 10 MBps. That is to say, the performance increases as the increase of frame size.

3.3.3 The Impact of Filesystem

In the experiment, an ext3 filesystem whose size is set to 400 MB is used for file caching. By measuring the performance of the ext3 filesystem locally (on the target) and remotely via iSCSI, one thing that can be observed is filesystem has a great impact on the system performance. From the figure showed below, file size less than 400 MB have the caching effect from the filesystem; however for those whose size are larger than 400 MB, the caching effect is reduced. In addition, accessing filesystem locally on the target is always better than remotely does. According to the discussion presented above, it can be speculated that the performance of the system increases with the cache size increased.

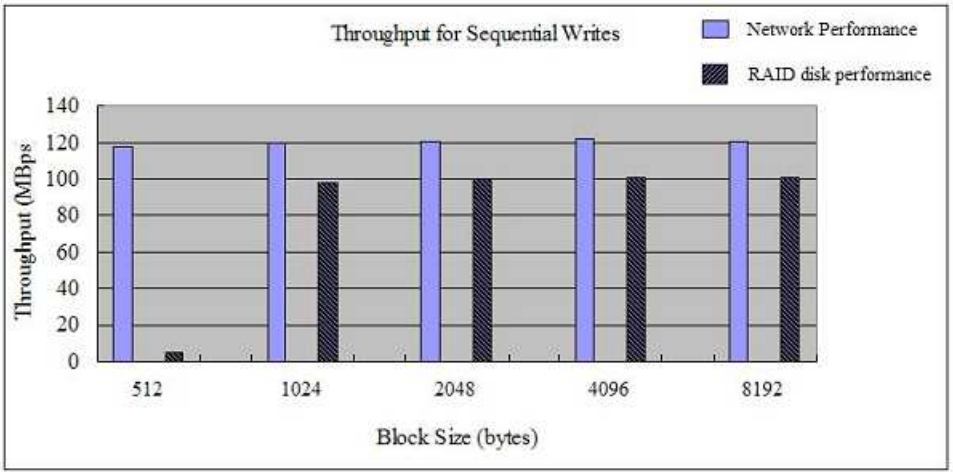

Figure 14: iSCSI's behavior during sequential writes by using the buffer set to the default value

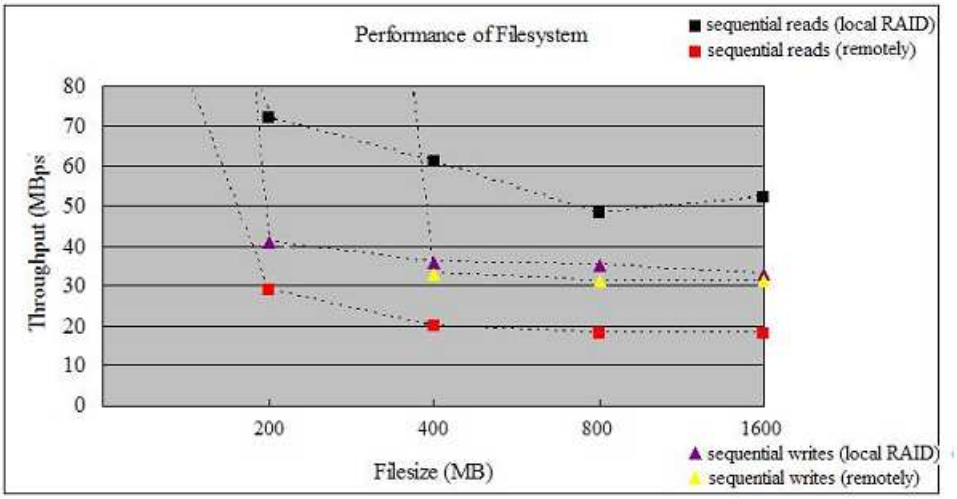

Figure 15: the impact of filesystem

3.3.4 The Rest Experimental Results Analysis

From Figure 11, 12, 13, and 14, an interesting observation is throughput has light variation with the change of block size. In other words, the block size is not a real factor that affects the throughput.

In addition, as iSCSI initiator requesting I/O transaction must wait for the target successfully write the transaction to disk before the target sends the acknowledgement. Furthermore, due to the fact that the bandwidth for the network and the disk are roughly equal. Therefore, the delay imposed by writing can be easily expected.

3.4 Summary and Conclusion

In the experiment, iSCSI tested in different environments was evaluated. The first interesting observation is that socket buffer size does have impact on network performance, but little effect on RAID disk performance. We found that when we set the buffer size to the default value that is defined by the used operating system. The network performance has about 3 MBps improvement comparing with iSCSI tested in the environment with largest socket buffer size.

The second part of the experiment evaluated the impact of frame size. Interesting enough, it was found comparing

with the performance results by using 9 KB Ethernet frame size, all experimental results obtained in the environment with 1.5 KB Ethernet frame size drop off 10 MBps. That is to say, the performance increases as the increase of frame size.

Finally, the impact of filesystem was evaluated in the experiment. One thing that comes to attention is accessing filesystem locally on the target is always better than remotely does. Moreover, we found that cache size has influence on the performance of systems. More specifically, system performance increases with the cache size increased.

4 Conclusions and Future Work

In the report, a set of benchmarks for application-level solutions and storage data transfer mechanisms are presented and designed.

For application-level solutions, selected protocols provided their own advantages: PSockets overcomes the default window size by aggregating multiple sockets' window sizes. UDT shows the improvements on the window size to achieve higher bandwidth. In comparison with Psockets and UDT, RBUDP avoids the slow start, and the window size. Moreover, with the help of different congestion control mechanism, it can perform quite well. However, due to testbeds we have in the experiment, it is difficult to test and discuss the performance of these protocols in environments with different high BDP. Furthermore, the issue of fairness about these protocols was also not considered. It would be better to check how fair these mechanisms are to TCP. One more interesting thing needs to be addressed is the effect of packet loss (transfer data from high capacity links to low capacity links).

For storage data transfer mechanisms, we find several variables that have a significant impact on network performance - such as socket buffer size, frame size, and filesystem size. However, due to the testbed we have, all these results were obtained from software-based SAN environment. That means results were from the network without switch delay, and also without any hard-RAID array storages. Therefore, in order to have a full picture of protocols performance in Enterprise Data Storage, all these issues need to be considered in future.

References

[1] Dickens, Phillip. M. "FOBS: A Lightweight Communication Protocol for Grid Computing". in *Proc. of Euro-Par 2003*.

[2] Dickens, Phillip. M., and Gropp, W. "An Evaluation of a User-Level Data Transfer Mechanism for High-Performance Networks". To appear at HPDC *2002*.
From: http://www-unix.mcs.anl.gov/ gropp/bib/papers/2002/hpdc02-fobs.pdf

[3] Jacobson, V., and Braden, R. "TCP Extensions for Long-Delay Paths". *RFC 1072, October 1988*

[4] Katabi, D., Hardley, M., and Rohrs, C. "Internet Congestion Control for Future High Bandwidth-Delay Product Environments". *Proc. of ACM SIGCOMM 2002.*

[5] Lakshman, T. V., and Madhow, U. "The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss". *IEEE/ACM Trans. On Networking 5, 3 (1997)*.

[6] Clark, D., Lambert, M., and Zhang, L. "NETBLT: A High Throughput Transport Protocol". *Proc. of SIGCOMM '87, (Stowe, VT)*, pp. 353-359.

[7] Gu, Y., and Grossman, R. "Using UDP for Reliable Data Transfer over High Bandwidth-Delay Product Networks". Submitted to Computer Communication Review, *2003*.
From: http://www.ncdm.uic.edu/papers/udt-protocol.pdf

[8] Feng, W., and Tinnakornsrisuphap, P. "The Failure of TCP in High-Performance Computational Grids". *Proc. of Supercomputing 2002.*

[9] Dickens, Phillip. M. "A High Performance File Transfer Mechanism for Grid Computing". *Proc. of the 2002 Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*. Las Vegas, Nevada, *2002*.

[10] Gu, Y., Hong, X., Mazzucco, M., and Grossman, R. “SABUL: A High Performance Data Transfer Protocol”. Submitted to IEEE COMMUNICATIONS LETTERS for publication, *2003*.
From: http://www.rgrossman.com/pdf/sabul-hpdtp-11-02.pdf

[11] Anglano, C., and Canonico, M. “Performance Analysis of High-Performance File Transfer Systems for Grid Applications”. *Concurrency and Computation: Practice and Experience 18(8), pp. 807-816, July 2006.*

[12] Zhu, Y., Bassi, A., Massonet, P., and Talia, D. “Mechanisms for High Volume Data Transfer in Grids”. *CoreGRID Technical Report TR-0121, December 2007*.

[13] Sivakumar, H., Bailey, S., and Grossman, R. “PSockets: The Case for Application-Level Network Striping for Data Intensive Applications using High Speed Wide Area Networks”. *Proc. of Supercomputing 2000.*

[14] National Center for Data Mining, “UDT: UDP-based Data Transfer Protocol”. *July 2007.*
From: http://udt.sourceforge.net/

[15] Gu, Y., and Grossman, R. “UDT: UDP-based Data Transfer for High-Speed Wide Area Networks”. *Proc. of Computer Networks, 2007*.
From: http://www.ncdm.uic.edu/publications/files/journal-035.pdf

[16] He, E., Leigh, J., and Yu, O. “Reliable Blast UDP: Predictable High Performance Bulk Data Transfer”. in *IEEE Cluster Computing 2002*, Chicago, IL, *Sept. 2002*.

[17] Tirumala, A., Qin, F., Dugan, J., Ferguson, J., and Gibbs, K. ”Iperf Tutorial”. *May. 2005*.
From: http://dast.nlanr.net/projects/Iperf/

[18] Dugan, J.M. “iperf-users email list – Iperf 2.0.4 Released”, *April. 2008.*
From http://www.mail-archive.com/iperf-users@lists.sourceforge.net/msg00003.html

[19] Barkakaty, U. “iperf-users email list – Iperf Transmit leading to 100 percent CPU Utilization”, *May. 2008*.
From http://www.mail-archive.com/iperf-users@lists.sourceforge.net/msg00024.html

[20] Meyer, D. “An Introduction to Networked Storage”. *White Paper, Network Appliance, Inc, August 2007*.

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

[22] Garth, A. G., and Rodney, V. M. “Network Attached Storage Architecture”. *Communications of the ACM, November 2000, Vol.43, No.11*.

[23] Senapathi, S., and Hernandez, R. “Introduction to TCP Offload Engine”. *Dell Power Solutions, March 2004.*
From http://www.dell.com/downloads/global/power/1q04-her.pdf

[24] Krueger, M., Haagens, R., Sapuntzakis, C., and Bakke, M. “Small Computer Systems Interface protocol over the Internet (iSCSI) Requirements and Design Considerations”. *IETF, RFC 3347, Informational Standard, July 2002.*

[25] Iometer Project. ”Iometer User’s Guide”. *Dec. 2003*.
From: http://iometer.cvs.sourceforge.net/*checkout*/iometer/iometer/Docs/Iometer.pdf

[26] Information Networks Division Hewlett - Packard Company. ”Netperf: A Network Performance Benchmark”. *Feb. 1995*.
From: http://www.netperf.org/netperf/training/Netperf.html

[27] Bray, T. ”Bonnie”. *Feb. 1996*.
From: http://www.textuality.com/bonnie/

[28] Chase, J., Gallatin, A., and Yocum, K. ”End system optimizations for high-speed TCP”. in *IEEE Communications Magazine 2002, Vol.39(4)*.