RJ 10293 (A0305-006) May 7, 2003
Computer Science

IBM Research Report

ISCSI Performance and Architecture and Comparison to Other Network Protocols*

Jai Menon
IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

Carlos Fuente
IBM Hursley, UK

* This report is based on work performed prior to February 21, 2000 and has not been modified since it was written on February 21, 2000.

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

ISCSI performance and architecture and comparison to other network protocols 1
. 1
1. Introduction 3
2. How is Storage Accessed Today? 3
2.1 SCSI bus 3
2.2 File Protocols 3
2.3 SANs 3
2.4 Summary of today's approaches to storage access 4
3.0 SCSI Protocol Over TCP/IP 6
3.1 What Transport Protocol to Use? 7
3.2 Potential Applications of SCSI over TCP/IP 8
3.3. Performance of SCSI over TCP/IP 8
3.3.1 Host Processor Tasks 9
3.3.1.1 Data moving 9
3.3.1.2 Handling Protocol Information 11
3.3.1.3 Interfacing to Hardware 11
3.3.2 Description of the Experimental Setup 11
3.3.2.1 Choice of Operating System 11
3.3.2.2 Alternative Communication Mechanisms Measured 12
3.3.2.3 Measuring Microprocessor Utilization 13
3.3.2.4 Utilization Model 14
3.3.3 Results for SCSI over GE versus SCSI over FC 15
3.3.4 Comparing NAS and GE SAN 16
3.3.4.1 Measuring utilization 16
3.3.4.2 Comparisons on Linux 17
3.3.4.3 Results of NAS versus GE SAN 18
4.0 Acknowledgments 19
5.0 References 19

1. Introduction

This paper discusses various approaches to accessing storage over IP networks. In particular, we discuss an approach for accessing storage using the SCSI protocol layered over TCP/IP, called **iSCSI**. We discuss the pros and cons of such an approach versus alternative schemes for storage access.

2. How is Storage Accessed Today?

2.1 Block Protocols

Direct access to storage has generally been through buses such as SCSI or IDE, using **block protocols**. A typical block protocol is the SCSI block protocol. This protocol can run on a SCSI bus, but later we will see examples where the SCSI protocol is layered on top of other networks.

2.2 File Protocols

Storage is also accessed through networks and file servers using **file protocols** such as Network File System (NFS), Common Internet File System (CIFS), Andrew File System (AFS) and Distributed File System (DFS). NFS is typically used on Unix systems and CIFS is used on Windows systems and typically operate in LAN environments. AFS and DFS are designed to operate in wide-area environments. The file protocols are typically layered on top of TCP/IP (though NFS was first designed to operate on top of UDP).

Direct access to storage via block protocols has been used where performance is important. It is the typical form of access used by servers when they need to access storage. File protocol access to storage has been used where there is a need for sharing of files between multiple users, but where performance is less critical. It is the typical form of access used by clients when they need to share storage, though clients also use block protocols for accessing non-shared storage.

2.3 SANs

More recently, a new model for direct access to storage has emerged, called the **Storage Area Network or SAN**. As with the SCSI bus, SANs use block protocols, there is no server in the way between the computer and the storage, and performance is as good or better than using the SCSI bus. Unlike with the SCSI bus, multiple servers can directly access the same storage. There is general agreement that SANs offer considerable value to servers, including higher data availability, less expensive data and storage management, and more scalable storage operations. By adopting SANs, customers hope for lower cost operations and faster deployment of applications, even while the amount of managed data grows explosively. While the SCSI bus provides direct access to storage for both clients and servers, SANs only provide direct access to storage for servers. This is because the SAN network typically does not extend to the user's office, and is limited to being within a machine room.

SAN is synonymous with the Fibre Channel (FC) standard today. The SCSI block protocol has been mapped to run over FC, and this mapping of SCSI on FC is called FCP[]. The SCSI block protocol was invented to run over a bus to dedicated storage. It has some shortcomings for use in a shared environment such as a SAN. These shortcomings are being overcome through SCSI extensions. One such SCSI extension, proposed by Almaden Research, is an access control list extension, called SCSI ACLs, which prevents servers from accessing storage they are not allowed to. This is now an IETF standard [].

Unlike file servers, SANs do not automatically provide data sharing. However, they are an enabler for data sharing, since they allow multiple servers to access common storage. SAN file sharing software must be developed in order to allow data sharing. Examples of SAN file systems which allow sharing of data between homogeneous servers (servers running the same OS) are CFS for Numa-Q and GPFS for SP2. An examples of a SAN file system which allows sharing of data between heterogeneous servers is Storage Tank [] from IBM Almaden Research.

SANs have approached network storage by trying to meet direct attach performance while improving the scalability and availability characteristics of storage. In doing so, SANs have rejected existing networking technologies as too slow (IP over ethernet), and worked on providing an optimized network for storage. These networks are today deficient in interoperability, network management and security characteristics.

Finally, since SANs seek to optimize performance and storage device simplicity they do not address long distances. In a typical I/O environment, it is relatively rare to send data at very large distances. For the applications that require long distance (disaster recovery and data migration) bridging technologies that connect the edge of a SAN to another remote SAN have been devised. The most common approach is to tunnel SAN traffic from one edge connect product across a WAN to another edge connect product on the remote SAN. FCIP is a new emerging standard in this area [].

In spite of these shortcomings, SANs are now being deployed in large server environments.

Today, SANs are built using Fibre Channel (FC) technology, but it is possible that the underlying technology for SANs could be replaced in the future. FC offers 100Mbyte/sec performance today, with up to 90% efficiency and low CPU utilization. Host adapters, hubs, switches, storage and WAN bridges all exist today for FC. A technical roadmap for link speed improvements has been established by the Fibre Channel Industry Association, and standards are provided by ANSI NCITS T11. From 100Mbyte/sec today, FC will improve to 200Mbyte/sec in 2001 (initial products are available now), and a standard has been drafted for 1 Gigabyte/sec speeds (4x2.5Gigabit) which could be available by 2002.

2.4 Summary of today's approaches to storage access

The table below summarizes the 3 approaches commonly used today to access storage.

As discussed before, both SCSI and SANs have high performance, but access to storage via a file server is not as fast.

The SCSI bus provides no ability to share storage devices. File servers allow storage and file sharing. SANs can provide storage sharing, but extensions to the SCSI protocol are needed. Almaden Research has proposed such extensions to the ANSI T10 committee [Hafner].

The SCSI bus provides no file sharing mechanisms. File servers provide file sharing between multiple users. SANs enable file sharing by providing direct connectivity to common storage from multiple servers, but file sharing requires other software which leverages the connectivity provided by the SAN.

SCSI buses allow attachment of storage devices at distances of 10s of metres, SANs allow attachment at distances of 10s of Kms and file servers allow attachment of storage over WAN distances.

Desktop client computers can uses storage attached via SCSI buses or file servers, but SANs do not extend to the desktop today. Servers can use all three types of storage.

SANs and SCSI buses allow direct connectivity to storage, file servers impose an additional server in the path to storage.

Home computers can use either SCSI bus attached storage or file servers, but SANs are not present in homes today.

The SCSI bus and SANs use block protocols, file servers use file protocols. All applications can use SCSI attached storage and SAN attached storage, because SCSI block protocols are universally used to access storage. Most applications also use a file system, so they can also use storage attached via a file server. However, some applications such as some databases systems and paging supervisors do not use file systems, so will be unable to use file servers.

All kinds of storage can be attached via a SCSI bus (disks, tapes, etc.), but file servers only attach disk drives. SANs can also attach any kind of storage.

	SCSI bus	**File Server**	**SAN**
Performance	High	Medium	High
Storage Sharing?	No	Yes	Yes, needs extensions
File Sharing?	No	Yes	Enabler
Distance	Few meters	Internet	Few Kms
Clients can use?	Yes	Yes	No
Servers can use?	Yes	Yes	Yes
Needs intermediate server?	No	Yes	No
Can be used in home	Yes	Yes	No
Protocol	Block (SCSI)	File (NFS, CIFS)	Block (FCP)

Driver software	standard	standard	Becoming standard
Application coverage	Universal	High	Universal
Device coverage	All SCSI devices	Disk drives only	Universal
Host OS coverage	Universal	Windows, Unix	Server OSes
Choice of file system	Anything supported by OS	No choice	Anything supported by OS

3.0 SCSI Protocol Over TCP/IP - iSCSI

Internet technology, based on IP, is replacing a wide-range of specialized networking solutions, including leased lines, proprietary network architectures (e.g. IPX), and so on. IP-based solutions are more cost-effective, support wider range of connectivity and are familiar to a huge pool of network administrators, managers and engineers.

IP technology has moved to performance levels that are adequate to meet storage network requirements, including 100 Mbps and upwards transfer rates and few millisecond latency.

Using IP provides all the compelling benefits of the Internet. Using an IP network as the Storage Area Network eliminates the need for building and operating a separate network infrastructure just for storage. Because of their enormous volumes, our assumption is that IP switches and fabrics will eventually be cheaper than FC switches and fabrics, though their costs are likely to be similar. IP allows us to use standard routers, instead of FC to WAN bridges, making it much easier to access storage at great distances (cross-country). Interoperability of IP switches and routers is significantly better than that of FC switches and routers. Since clients already have Ethernet cards, such an approach allows storage to be extended to the clients, something not possible with FC SANs. Finally, since Ethernet and TCP/IP are already available on Windows machines, it allows storage to be accessed on home networks. The table below compares FC SANs and IP-based SANs. Differences are shown underlined and in red.

	FC SAN	**TCP/IP SAN**
Performance	High	**Unknown**
Storage Sharing?	Yes, needs extensions	Yes, needs extensions
File Sharing?	Enabler	Enabler
Distance	Few Kms	**WANs**
Clients can use?	No	**Yes**
Servers can use?	Yes	Yes
Needs intermediate server?	No	No
Can be used in home	No	**Yes**
Protocol	Block (FCP)	Block (FCP)
Driver software	Becoming standard	**Not standard**
Application coverage	Universal	Universal
Device coverage	Universal	**Universal, through controller**

Host OS coverage	Server OSes	**New drivers needed**
Choice of file system	Anything supported by OS	Anything supported by OS

In IBM Research, we started a joint project between the Almaden and Haifa labs to understand the benefits and problems of running SCSI over TCP/IP in September 1998. Steve Hetzler of Almaden Research was the first evangelizer of this concept within IBM. Outside IBM, Rodney Van Meter at Information Sciences Institute (ISI) appears to have been the earliest to suggest this approach [NAP]. In his March 1998 paper, he already describes a working system.

We had two key concerns when we started. One was regarding the performance of such an approach. This paper takes a first stab at some performance measurements. The second concern was that no standard was in existence. The latter problem is likely to be solved through the joint IBM/Cisco initiative for standardization through the IETF. Julian Satran of IBM Haifa Research has been the driving force behind the iSCSI standards initiative, and many other IBMers such as John Hufferd, Prasenjit Sarkar, Kal Voruganti, Jim Hafner, Kalman Meth and Daniel Smith have been very active in the iSCSI standardization effort. Many other companies have joined the initiative and a standard is likely to be approved by the IETF in 2001.

3.1 What Transport Protocol to Use?

The first issue we had to resolve was regarding whether to use TCP, UDP, or some other transport protocol on top of IP.

If we only want something which operates well in LAN environments, but not in the wider Internet, then alternatives to TCP are worth considering. If the requirement is to operate on the Internet, TCP is the only realistic choice, since TCP currently constitutes the majority of the Internet traffic. Our feeling was that we wanted something which operated well in both LANs and WANs. If we built something which only operated well in LANs, the advantages over using FC would be less compelling.

A large amount of time and effort has been gone into TCP to achieve its current maturity and congestion handling. This effort has produced Nagle's algorithm, slow-start, sophisticated round-trip time estimation, silly-window syndrome avoidance, exponential back-off, etc. These mechanisms are known to provide fair sharing and efficient use of bandwidth in large scale networks and avoid congestion collapse, based on hundreds of research studies and years of operational experience.

TCP is already a standard, so selecting to use TCP dramatically reduces the time and effort to establish a standard for storage area networks. Realistically, development of a new protocol will require several years, witness the work on XTP, ISO TP and others in the 1980s. TCP itself is a product of 20 years of research, experimentation and implementation.

New protocols can be invented that are better than TCP in a LAN environment, but it is unlikely we are going to find anything better for the WAN. Furthermore, since all Internet traffic uses TCP, its performance is going to be continually improved through better software stack implementations and hardware offloading.

Current semiconductor technology allows the implementation of high-performance TCP stacks in both hardware and software. This is a significant change over previous years where the cost of silicon was much higher than it is today. The TCP fast path has been identified, extensively used in software implementations, and well within the capabilities of hardware to implement at wirespeed. Contemporary network interface controllers already provide hardware support for TCP transmit, reducing CPU utilization. Thus hardware implementation of TCP is feasible that would handle TCP transmission and reception at wire-speed. The header overhead with TCP is low compared to bulk transfer so there is also minimal gain to be had there in a new protocol.

3.2 Potential Applications of SCSI over TCP/IP

For applications that would benefit from existing network infrastructure, use of TCP/IP as the networking technology for storage is a big advantage. Desktop or mobile clients could use a single interface for all data traffic, including storage access. Remote data access by servers, or remote copy functions for backup or disaster recovery could be done with a single protocol without bridging. Low end applications would have adequate performance right away, even over 10-100 ethernet LAN connections. High-end server applications would have to wait for TCP/IP hardware acceleration, but there is work underway in several companies to provide this new hardware.

The SCSI over TCP/IP approach might have early success in low end SOHO and small business applications. The competition in the home is NAS. However, SCSI over TCP/IP has the advantage that it can provide shared access to storage other than disks - for example, this technology might be used for CD or DVD sharing. This is not possible with NAS. Furthermore, as we will will show iSCSI has better performance than NAS for the small business user.

Another natural use for this technology is for long distance storage applications such as remote copy, remote archive, and disaster recovery services over the Internet. Today, remote copy requires special leased lines and proprietary protocols, but having remote copy work over the Internet would be much more convenient.

In the long run, SCSI over TCP will occupy some share of the SAN market - probably complementing FC SANs.

3.3. Performance of iSCSI

The simple bandwidth capability of Gigabit Ethernet has been demonstrated [Prasenjit]. However, this demonstration also illustrated the main issue with using TCP/IP and a conventional network interface card (NIC) for storage traffic; the proportion of the main processor's time (as measured by the utilization) vs. the bandwidth achieved, is higher than a traditional, high-performance, storage adapter would consume.

The importance of the processor efficiency of the storage attachment is exemplified by the attention given to this by FiberChannel adapter vendors [HP]. Also, numerous studies in the network arena have focussed on the efficiency of network implementations, and the challenges of scaling these to gigabit rates [Trapeze, Osiris]. And, much of the effort in the latest I/O architectures, such as InfiniBand, seeks to improve the efficiency of I/O operations.

To measure performance, we used 2 450MHz Pentium III machines, with 256 MB of memory, running Linux 2.1.132. Each machine had an ACEnic Gigabit-Ethernet NIC card. One machine acts as the computer initiating I/O (**the initiator**) and the other machine is the target storage device (**the Target**). The Initiator is performing IOs using the raw I/O path. The target is a quick-and-dirty implementation handling only the basic SCSI commands with minimum processing overhead. Hence, processing overhead at the target can almost all be attributed to the overhead of the communication protocol. Measurements were taken at a variety of IO sizes for each of the several methods of communication used between the initiator and the target.
We used four different communication methods. First, we used SCSI implemented over TCP/IP. Second, we used SCSI implemented over raw Gigabit Ethernet (GE), bypassing the overhead of TCP/IP. The third communication method was the same as the second, except that all copy operations were avoided on data transmitted from the source, by using the scatter/gather capability of the ACEnic card. Finally, we used FC cards instead of GE cards - one from Qlogic and 1 from HP.

3.3.1 Host Processor Tasks

There are many tasks a processor might be responsible for in handling an I/O adapter. These include data movement, protocol processing and hardware interfacing.

3.3.1.1 Data moving

Many I/O implementations require that the microprocessor copy data from one buffer to another, when traveling between a user application and the I/O adapter. There are a number of reasons for this, but they can include:

Crossing protection domains. In some environments, data located in the virtual storage of a user level application must be copied into kernel mapped virtual storage, so it can be manipulated by the kernel driver code for the IO adapter.

Constructing frames. Sometimes, data has to be organized into contiguous pieces, based on some feature of the underlying protocol, such as a frame. If the adapter interface supports scatter-gather facilities, this can usually be avoided. However, sometimes there are limitations in the scatter-gather support, such as an inability to handle odd-byte alignments, which require data to be moved to a supported alignment before it can be managed by the adapter.

Demultiplexing concurrent streams. Many I/O interfaces support multiple concurrent operations to be outstanding. The individual I/O operations proceed independently, such as I/Os outstanding to multiple disks, or many TCP/IP applications running at the same time. This can lead to multiple incoming data streams arriving close together in time, with no predefined order

for the data. Some feature of the protocols in use must distinguish the streams, and arrange that the data eventually reside in the correct user application buffer.

In most existing SCSI and FC implementations, the different streams are identified and separated by the hardware. The hardware is also informed of separate destinations for each possible stream, and can correctly locate each stream to the appropriate destination. In most network adapter implementations, separating the streams is a function of the network protocol software. The hardware places incoming data into buffers which are shared by all streams. The data is then often copied to the required buffer.

The TCP RDMA feature seeks to avoid this.

Retransmission in event of error. Most I/O protocols are subject to errors, and must implement protocols to make the I/O reliable in the presence of temporary or transient errors. Typically, the recovery involves retransmitting the data associated with the I/O operation. This means that a copy of the data must be preserved, such that it can be retransmitted in the event of error. One strategy is simply to take a copy of the data, so that it can be retransmitted. A second strategy is to use the original buffer, as provided by the application. However, the semantics of the I/O operation usually demand that the retry transmit identical data to that sent in the original request. Hence, some mechanism must be used to protect the original buffer from being updated. Mechanisms can include blocking a process from executing for the duration of the I/O, or making the buffer read-only, so that the owning process can run, unless it tries to update the buffer.

Existing network implementations typically take a copy before transmitting data. Most storage implementations will transmit directly from the buffer provided. Note, that applications have become attuned to the usual mode of operation of the interface. Network applications can send data, then expect to be able continue operation immediately, perhaps manipulating the memory containing the data just sent. Applications issuing storage I/Os are often aware that reading and writing data can take a long time, and if high throughput is required, will use multiple threads of execution to ensure concurrent operation. See [ZC] for further discussion.

Virtual memory / read cache implementation. This is strictly not part of the I/O protocols operation, but this does figure in the path between an application issuing I/O requests, and an interface adapter. Many operating systems implement a cache as part of their file system or virtual memory implementation. The intention is to improve performance of simple applications, without requiring each application to optimize its I/O mechanisms for the underlying hardware. As part of its operation, the cache may retain a copy of data read or written, which adds to the work of the main system processor.

Operating systems differ widely in the mechanisms they use for disk caching. Most operating systems also offer mechanisms to disable caching, perhaps on a file basis, or sometimes for an entire block storage unit (aka raw I/O). This last mechanism is primarily intended for the use of sophisticated applications, such as relational databases, which have a detailed understanding of the I/O performance profile. These applications will be carefully designed to manage their

internal operations to optimize I/O throughput. These applications can benefit from having the least interference with their I/O from the operating system.

3.3.1.2 Handling Protocol Information

A second class of responsibilities which can fall on the system microprocessor, is providing protocol information. This is extra bytes of information which are required for the I/O protocol, apart from the data itself. This information can take the form of address bytes, I/O request definitions (Command Descriptor Blocks), checksums, and others.

For a single I/O request, the amount of information required can vary widely, according to I/O protocol and a vendor's implementation. At a minimum, a single concise control block (perhaps only 64 bytes) is required. The interface adapter will often then use fields in the concise block as indices to lookup the more detailed information required.

At the other extreme, the main system microprocessor must provide all the protocol information explicitly, sometimes repeating the information many times for a single I/O request, such as a transfer which must be broken up into many frames.

When receiving data, protocol information must be examined, and stripped from the incoming data stream. There is a similar spectrum of mechanisms by which this responsibility can be divided between I/O adapter and host microprocessor.

3.3.1.3 Interfacing to Hardware

Performing I/O requires interaction with some hardware. This requires handling of interrupts and manipulating hardware registers to communicate between the I/O adapter and the system microprocessor. The precise work required is specific to a particular vendor's implementation.

All of these responsibilities involve execution of host code, and manipulation of data structures in the microprocessors memory. Apart from taking time, these operations also change the contents of the processors caches.

3.3.2 Description of the Experimental Setup

This experiment seeks to measure the cost of performing I/O over a number of connection implementations, operating at Gigabit speeds.

3.3.2.1 Choice of Operating System

Linux was chosen for these experiments because it provided the cheapest route to implementing the required code and because of driver availability. It also provided a more familiar environment for the experimenters.

Note, that some of the connection technologies involve executing significant amounts of OS code, specifically portions of the network protocol stack. Necessarily, these have a heavy dependency on the precise OS chosen. However, the general qualitative results should apply equally to any operating system which might have been chosen.

3.3.2.2 Alternative Communication Mechanisms Measured

We compared both GE and FC point-to-point, communication methods. For the Gigabit Ethernet network adapter, we used the Alteon Networks ACEnic adapter. For FC, we used a QLogic FiberChannel adapter (ISP2100A). All the communication was between two Linux machines. One ran a SCSI target emulation (the storage device), the other implemented a SCSI initiator, providing well-defined I/O.

We looked at several alternatives for the GE connection. One was to use the Linux TCP/IP kernel socket interface, to provide a connection between target and initiator.

The other implementations drove the network adapter directly, thus avoiding most of the operating system code involved in the TCP/IP implementation. When transmitting data, two mechanisms were used. One involved constructing the outbound frame contiguously in main memory. This require copying the user data from the source buffer to a special frame buffer, before pre-pending the necessary frame header and protocol information. This provides the single copy raw ethernet mechanism.

Another mechanism took advantage of the scatter-gather capabilities of the ACEnic adapter to avoid the copy, and provide a zero-copy transmit raw ethernet mechanism. The frame headers were still constructed in a separate memory buffer, but the user data was transferred directly from the source buffer.

In receiving data, there were also two implementations for raw ethernet, a single-copy and zero-copy. The single copy mechanism simply received all inbound frames into receive buffers, and performed the necessary copies to move data to the required destination buffers.

Implementing zero-copy receive required a somewhat contrived mechanism. The ACEnic adapter provides two receive queues, which act as FIFOs. One handles 'standard-sized' frames, the other handles 'jumbo' frames. To segregate control and data flows, the data flows were engineered to ensure their frames were larger than the control frames, such that they were exclusively handled by the 'jumbo' receive queue. To ensure that this mechanism did not receive benefit from using larger frames, the definition of the 'standard' frame size was reduced to 196 bytes, so that data frames with as little as 512 byte data payload still used the 'jumbo' queue.

Further, the order of inbound data was assumed to be known, through prior arrangement of the data transport protocol. The jumbo receive buffers were constructed from the intended final destination buffer provided in the SCSI command. The ACEnic's scatter-gather capability was used to strip the protocol header before locating the frame payload in the data buffer.

This mechanism will not scale well to a more complex environment, with multiple targets and multiple initiators, due to the need to engineer the order of frames received. Also, the overloading of the meaning of the size of a frame is clumsy. A real mechanism would require the use of a field in the inbound frame to identify the stream, and the NIC would need to be pre-advised of the final destination for each stream. This would require changes to the NIC itself, which was outside of the scope of this study.

However, the mechanism implemented does give a fairly good emulation of the workload required to achieve zero copy operation.

3.3.2.3 Measuring Microprocessor Utilization

Though seemingly a simple concept, this is actually a complex question. Simply measuring the amount of time spent executing code for the I/O protocol is one mechanism. However, the Linux tools for measuring utilization were found to be highly suspect. For some workloads, they failed to detect any CPU activity, even though the system was transferring some 40 MB/s. Also, such a mechanism does not account for the impact that the I/O protocol has on the microprocessor's cache, and might not account for the impact of context switching between the 'useful' work, and the work required to perform the I/O.

Given the problems we ran into, we ended up using an alternative mechanism. This involved implementing a program which executed a tight loop, incrementing a counter for a defined period. When the system was idle, a reference time was taken for how long the program executed.

A given I/O load was then applied. Once this stabilized, the execution time was again measured. The cost of the I/O handling was perceived as an extended execution time, which was attributed to the I/O handling itself. Note, the extended execution time does not just represent time executing the I/O code, but also the context switching required, and any extra delays caused by the need to reload caches which had been disrupted by the I/O handling.

The program chosen, a tight loop, has the smallest 'cache footprint' possible for any target workload. In real systems, it is likely that the I/O workload is competing for processor time with applications with much larger working sets. This might lead to results which differ significantly from those below. Also, real applications are likely to actually be driving the I/O being generated, which might lead to some improvement through 'cache warming' affects (but see [ZC]).

We only measured the utilization at the target device. The target implemented a very simple ramdisk device, coupled as cheaply as possible to the interrupt and scheduler system in Linux. Hence, the bulk of the perceived code execution was due to the communication overhead. All the GE implementations used essentially the same ramdisk code. The QLogic measurements used a slightly more elaborate implementation, which was provided with that driver.

The initiator was required to execute rather more code, since it involved a user application to drive the I/O. To allow precise control of the I/Os delivered to the target, a Raw I/O mechanism

was used. Thus a single application I/O mapped to a single target I/O, in all circumstances. Nevertheless, the raw I/O path still involves executing code to traverse the operating system, which would imply a relatively large code execution time which might mask some of the measurements required.

The raw I/O mechanism is a relatively recent addition to Linux. As was later discovered, it is not particularly efficient, as compared to the implementation in other operating systems.

The I/O workloads used were all single-threaded. That is, one I/O outstanding at a time. This had the practical benefit of simplifying the coding requirements, and also avoided the need to drive multiple concurrent I/Os through the raw I/O mechanism.

It also had the benefit of keeping the operations of the protocols more consistent, even as the workload was varied. With multiple concurrent operations, there are many strategies possible which can coalesce work when under heavy load, and reduce the total cost of a set of operations, perhaps at the expense of increased latency. There are wide variations between vendors in their ability to work more efficiently under heavy load. Using a simple workload minimized the opportunity to optimize the work, and helped ensure the measurements focussed on the basic cost of the communication mechanism.

To improve the quality of the results, care was taken to ensure that the target system operated in a consistent environment at all times. The same applications ran during all measurements to ensure the background workload was constant; otherwise activity on the target was minimized. Time measurements were made using the same system clock on the target machine, using the time-of-day function. All measurements were averaged over a long period of time, of the order of 30 seconds. Only workloads which achieved 'stable', consistent, operation were measured and used in the calculations.

3.3.2.4 Utilization Model

Apart from simply measuring the processor utilization, this experiment sought to try and characterize the factors contributing to the utilization. An attempt was made to break the cost down into three factors:

Cost per byte - These are the costs associated with each extra byte in each I/O, perhaps as a result of having to copy the extra data.

Cost per frame - These are the costs associated with each extra frame, perhaps through having to construct or parse extra protocol header information, or through manipulating extra scatter-gather elements.

Cost per operation (op) - These are the fixed costs associated with each I/O operation, regardless of the amount of data or number of frames required. These might be associated with sending or receiving messages to start or finish the I/O operation.

A model might be proposed of the form:

Utilization = A * Bandwidth + B * Frame Rate + C * Op Rate

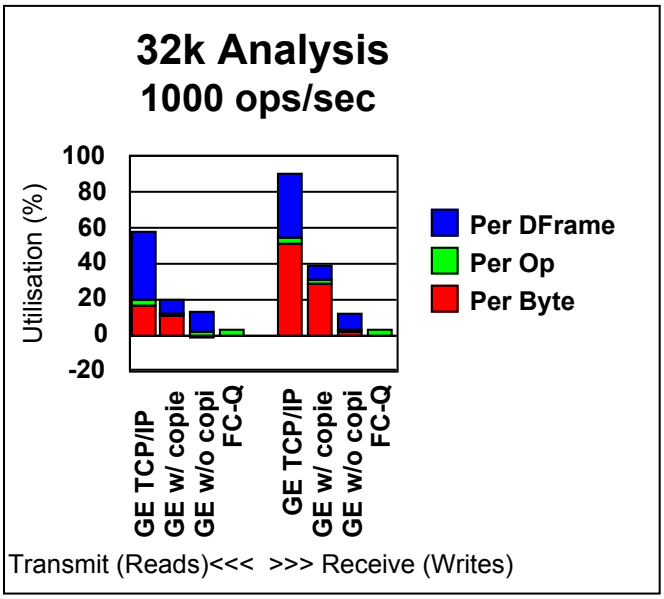
where A, B and C represent the per byte, per frame and per op costs respectively. This model provides a baseline for characterizing the work in the system, and is not intended to accurately describe the operation of the microprocessor. There are many reasons why the microprocessor's work is not simply additive, such as caching affects, greater efficiency under higher loads, cache thrashing under higher loads, etc.

However, the model does give some insight into how the utilization varies with op size and frame size, which gives clues as to what makes one mechanism more efficient than another.

In order to determine the coefficients A, B and C, measurements of IO rate and utilization were taken for a variety of IO and frame sizes. A linear regression 'least squares' calculation was then performed to determine the values A, B and C, assuming the model applied.

3.3.3 Results for SCSI over GE versus SCSI over FC

bw6.123



As can be seen, the SCSI over TCP/IP overheads are 30 times higher on reads to the storage device, and 45 times higher on writes to the storage device. When TCP/IP is eliminated, the ratios are 10x and 20x respectively. When copying is also eliminated, the ratios are 7x and 6x respectively.

Its clear that the TCP/IP stack processing must be offloaded to hardware. Its also clear that, additionally, eliminating any per byte processing, such as data moving or checksumming, is

important in reducing the overhead of using IP. Even a single copy of the data adds significant overhead for large IOs.

The next most significant contributor to utilization, is the existence of per-frame overhead, in the form of work the main processor must do to construct and deconstruct each frame of an I/O. Offloading this work to an adapter is required if GE is to match FC for processor utilization at high frame rates.

3.3.4 Comparing NAS and GE SAN

Given a SCSI over GE implementation, a GE network could be used to implement either a SAN, as above, or use existing file server (NAS) technology.

Implementing a NAS target requires quite different software than a SCSI SAN target. The NAS must implement networking protocols and a file system, before it can generate block oriented I/Os. By comparison, the SAN target, such as a controller, simply implements the SAN physical and transport protocols, and receives block oriented I/Os as a direct result.

On the initiator, the NAS implements storage at the file system level. So, to compare with a GE SAN, we must add file system processing to the processing we measured above. The protocol stacks which are compared are:

> NAS file system, plus network protocol, plus hardware interface
> Local file system, plus SCSI disk driver, plus SCSI to GE overhead

The addition of the file system adds significantly to the baseline cost of a SCSI implementation, which will tend to reduce the relative impact of the various SCSI over GE implementation options. It should be borne in mind however, that adding a file system is not always desired. The use of raw I/O by specialist applications has already been mentioned. Another example is paging space which can use some non-file system representation of block storage.

3.3.4.1 Measuring utilization

The addition of a file system further complicates the measurement of processor utilization. A file system can drastically alter the IO profile, through the application of cache algorithms. Examples include volatile caching of writes, read cache hits, write pre-emptions, coalescing and read ahead.

As a result, it becomes difficult to devise a sustained workload, and measure utilization and bandwidth achieved, and relate this to the properties of the interconnect.

A different measurement strategy was used, to try and minimize these affects. Rather than issue a continuos stream of Ios, the test driver simply issued a burst of write I/O. This might be initially cached, but would subsequently be driven to the target device. The sum of this activity comprises the work the microprocessor must perform to complete the IO. It is much easier to

design a short burst of I/O which will not be cached. The result is that all the data is transmitted over the SAN fabric.

The same looping program as before was used to act as a reference workload. The burst of I/O activity is perceived as an extended execution in one or more cycles of the program.

3.3.4.2 Comparisons on Linux

A set of experiments were performed on Linux systems, that compared the various options for how to use GE to attach storage. The mechanisms compared were:

Local File System over SCSI over TCP/IP over GE
Local File System over SCSI over single-copy (1C) GE
Local File System over SCSI over zero-copy (ZC) GE
Local File System over SCSI over QLogic FC-AL
Remote NFS File System

Each used a ramdisk within the Linux target to implement the storage. A burst of non-coalesceable writes was issued. The system was monitored until these were flushed to backing store. The total extra execution time as perceived in the execution of a loop program were measured.

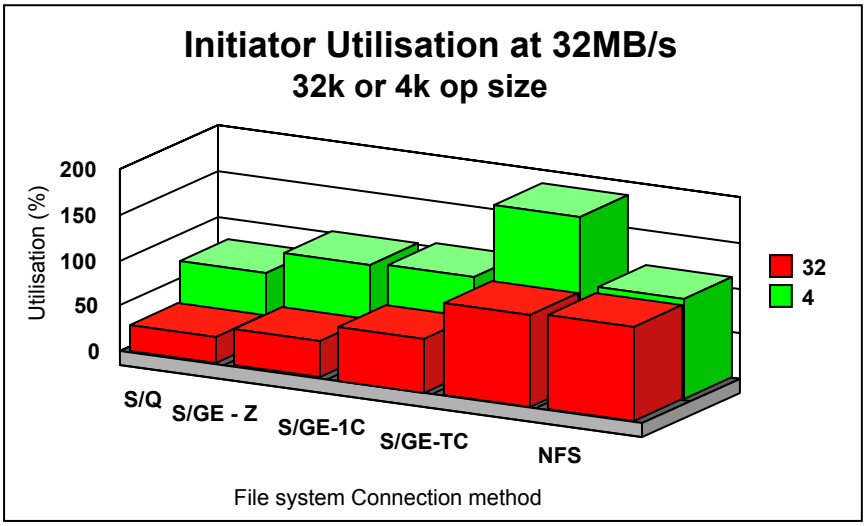
It must be understood that the introduction of a file system adds considerably more variability into the system. This makes these results difficult to generalize.

The comparison between Local FS and NFS is perhaps of limited value, since their handling of caching is quite different. Other OS implementations may give different results.

Also, even the same file system code can behave quite differently when layered on top of different SCSI drivers. For instance, the QLogic driver used in the test had a feature which caused a huge increase in overhead in the file system code which attempted to perform coalescing. The driver had to be changed to match the GE competition for the purposes of this comparison.

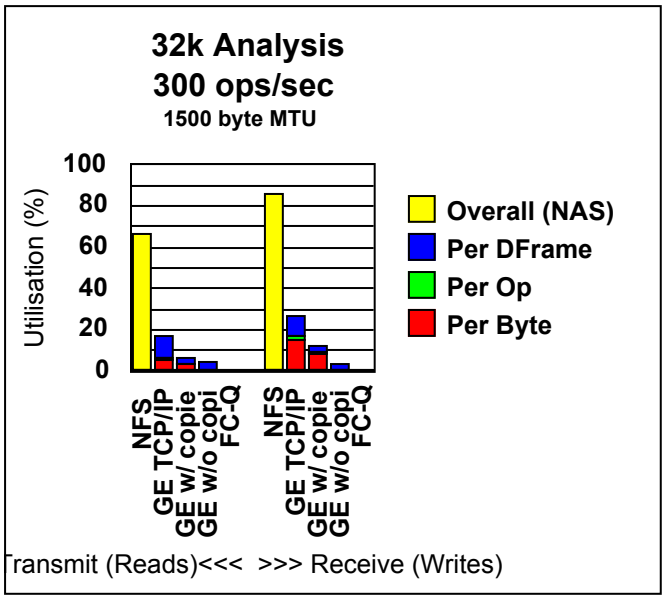
Another example of variability was observed in the Adaptec driver used to attach the system's own SCSI drives. This also showed very high utilization. This might be due to use of a real disk vs. a ram disk, or again some interaction between file system and driver for the particular I/O profile used.

3.3.4.3 Results of NAS versus GE SAN



The results indicate that GE SANs and NAS will have comparable performance on the inititator side.

We also compared performance of GE SANs and NAS on the target side and got the following.



The numbers are now:

Transmit	Total Utilization	Ratio to FC
GE TCP/IP	17.4	20.52
GE w/copies	5.86	6.91
GE w/o copies	3.99	4.71
NFS	66.6	78.35
FC-Q	0.85	

Receive	Total Utilization	Ratio to FC
GE TCP/IP	27.15	31.31
GE w/copies	11.75	13.56
GE w/o copies	3.67	4.23
NFS	86.4	99.31
FC-Q	0.87	

As can be seen, NAS requires anywhere from 78-99 times the processing power of FC SANs and 16-25 times the processing power of the best GE SANs.

4.0 Acknowledgments

Steve Hetzler was the first person in IBM to suggest the use of SCSI over TCP/IP, and he has been its strongest evangelizer. Carlos Fuentes and Prasenjit Sarkar did all the measurements reported in this paper. Manoj Naik and Prasenjit Sarkar did all the SCSI over TCP/IP implementations reported in this paper. Separate implementations were done by Daniel Smith and Efri Zeidner (IBM Haifa Research). The SCSI over TCP/IP specification was chiefly the work of Julian Satran and Daniel Smith.

5.0 References

[Prasenjit] Prasenjit Sarkar "Comparison of Gigabit Ethernet and Fibre Channel" gefc.PRZ

[HP] http://www.hp.com/HP-COMP/promo/hbas_perform.html

[Trapeze] Duke University Trapeze Project http://www.cs.duke.edu/ari/trapeze/

[Osiris] Osiris project ftp://ftp.cs.arizona.edu/xkernel/Papers/osiris.ps

[ZC] "Zero Copy TCP in Solaris" - 1996 USENIX Technical Conference

[SCSI-Over-TCP/IP]http://www.haifa.il.ibm.com/satran/draft-satran-scot-01.txt

[RDMA Option]
http://www.haifa.il.ibm.com/satran/draft-costa-rdma-01.txt

[NAP] Steve Holtz, Rodney Van Meter, Gregory Finn, “Internet Protocols for Network-Attached Peripherals”, Proceedings of Sixth NASA Goddard Conference on Mass Storage Systems, March 1998, http://www.isi.edu/netstation/ip-for-naps.ps

[Hafner] ftp://ftp.t10.org/t10/document.99/99-245r4.pdf