

***i*SAN - An intelligent Storage Area Network Architecture**

Ganesh Narayan and K. Gopinath

Computer Science and Automation,
Indian Institute of Science.
{nganesh,gopi}@csa.iisc.ernet.in

Abstract. This paper describes the motivation, architecture and implementation of *i*SAN, an “intelligent” storage area network. The main contributions of this work are: (1) how to architect an intelligent SAN that understands the storage-consumers¹ to serve them better; (2) how to realise this abstract architecture using existing technologies; and (3) to demonstrate the benefits that would accrue from such an intelligent SAN. Our results show that the *i*SAN approach has important benefits when compared with conventional SANs: *i*SAN facilitates storage sharing, is secure and offers better throughput. In this paper, we discuss two case studies as well as discuss how *i*SAN approach has been generic enough to capture a wide range of other requirements of SANs.

1 Introduction

Internet revolution drives a relentless demand for data to match the accelerating growth in users, digital content and network bandwidth availability. The need to scale data/storage independently has been the primary catalyst for the emergence of a storage tier providing logical/physical separation of storage from the other services in a data center. The result is the advent of an I/O architecture wherein the storage devices and high-speed networks are integrated into forming I/O networks: both the storage and the storage-consumer remain connected to a high speed network and communicate using SCSI commands. These I/O networks, called Storage Area Networks (SANs), provide better scalability and throughput as compared to traditional bus-based storage architectures.

Yet, scalability and throughput requirements are not the only requirements required from such SANs. Multitude of application domains, from content distribution networks to providing storage services, demand a range of properties/services that a successful SAN architecture should support. Unfortunately, SANs, whether based on FC [26] or iSCSI [22], do not export sufficient *functionalities* that are of direct use to storage-consumers. This is because traditionally SANs are seen merely as a replacement for parallel SCSI bus. But as a distributed shared storage system, SAN is more than an extended SCSI bus: SAN based systems demands functionalities which are otherwise not needed in parallel SCSI based systems.

For instance, consider storage sharing. Coordinating processes that access shared storage is not a problem in bus-based storage systems as any access to the storage is arbitrated by the storage-consumer to which the storage is physically connected; essentially there is no direct sharing of storage. In a shared storage system like a SAN where storage is directly accessible from multiple storage-consumers, sharing becomes a critical issue.

¹ A storage-consumer is the software layer that builds storage abstractions from block level storage provided by SANs. This layer typically includes, but not limited to, Volume Managers, File Systems and Data Base Management Systems.

Besides, contemporary SANs are generally unaware of storage-consumer's exact requirements. For instance, different storage-consumers expect different security guarantees from a SAN depending upon the threat model that they foresee. Hence, it is beneficial to enforce security properties judiciously especially when different security guarantees exhibit significantly differing cost/performance profiles. For example, a storage-consumer that is built assuming a byzantine storage, block level encryption done in SANs may not be of much use. Similarly, with different concurrency control schemes providing varying degrees of consistency guarantees with varying costs[15], one may want to selectively enforce that particular scheme which is economical and best suits the storage-consumer's needs. Hence, SANs should be *intelligent* enough to provide a set of guarantees that serves a storage-consumer best.

In this paper, we propose a novel SAN architecture called *iSAN*. *iSAN* identifies and provides services that are of direct use to the storage-consumers. In providing these services, *iSAN* also "understands" the service semantics sought by the storage-consumers and provides the needed service in a way that best suits the storage-consumer's requirements. The proposed *iSAN* architecture is extensible and generic; it switches the protocol stack based on the storage consumer.

The rest of the paper is organised as follows. Section 2 discusses the requirements that an I/O architecture should satisfy. In Section 3, we describe the architecture of *iSAN*. Section 4 describes our implementation of the proposed architecture which uses Linux kernel and Ensemble [10]. In section 5, we discuss how two important SAN services – concurrency control, security – are realised in *iSAN*. We compare the performance of the suggested solutions with that of existing solutions in Section 6. We conclude the paper in section 8.

2 Design

iSAN design is founded on a number of requirements; some of the them, like throughput and scalability, are inherited from basic SAN architecture with little or no enhancements. This section explains each of these requirement and discusses their applicability in state of the art SANs.

Interoperability In a SAN, a path from a storage-consumer to any storage device may include various combinations and permutations of host bus adaptors, hubs or switches and SCSI peripherals. Not all permutations and combinations are feasible even if all the subsystems have been built using the same network technology, let alone with dissimilar technologies. Thus a critical and essential feature of any SAN architecture is ensuring interoperability of components within the SAN.

Fibre Channel SANs have been suffering from interoperability problems as the higher level FC standards (especially planned at the FC-3 layer) have never been standardised. Zoning implementation is very much vendor dependent; FSPF protocol which ensures switch-to-switch routing is still proprietary; public and private loop devices still have problems when it comes to fabric logins. While network technologies like TCP/IP and Ethernet have a strong tradition of multi-vendor interoperability, FC devices often do not. Only recently approaches such as SMIS have shown interoperability at the level of discovery of device information but this does not still guarantee interoperability at the services level. This poses serious problems in the design, procurement, and operation of SANs and interoperability is thus becoming an increasingly important requirement.

Throughput With high speed network and disk interfaces, SANs are expected to be able to move the data as fast as possible. However, as the carrier bandwidth increases, the transport protocol inefficiencies at the end-points have a negative impact on the effective delay and throughput. For instance, the effective throughput of TCP over a gigabit network, without considerable hardware support from NIC, is only a fraction of the realisable throughput [9]. Even with zero-copy and checksum support from gigabit NICs, TCP has other problems. Extensions have been proposed but not many NIC products properly implement these extensions. In addition, the complexity of TCP has to be

taken into consideration. Also, with multiple independent TCP connection(s) between the SCSI end-points, iSCSI handles issues like congestion control less effectively. Importantly, failure detection becomes non-trivial as different TCP connections can break at different time instants, depending upon their respective past activity. Each of these shortcomings can affect the observable throughput in iSCSI SANs.

Even though FC SANs have better throughput than iSCSI SANs in local environments, this advantage is considerably less when it comes to WAN links due to the credit based flow control scheme used in FC SANs [25]. Given these observations, it is appropriate that a SAN leverages a SCSI transport protocol that is fast, efficient and simple, and has better flow control support.

Availability Today faced with critical need to ensure the availability and continuous operation in spite of isolated failures of disk, switch and links or the catastrophic loss of the computing/communication facilities, SANs need to be highly available. While FC SANs provide subsecond reconfiguration periods in case of a component failure, the traditional Spanning Tree Protocol (STP) employed in Ethernet takes tens of seconds to converge; it is to be noted that during the reconfiguration phase the extended Ethernet LAN is “frozen”. STP also has other problems: inefficient bandwidth usage, link blockage and STP is Virtual LAN (VLAN) unaware. A combination of Rapid Spanning Tree protocol (RSTP) and link aggregation would reduce the reconfiguration stalls to even tens of milliseconds. However, RSTP does not use the bandwidth effectively and still has link blockage problems.

Apart from ensuring the availability of SAN infrastructure, a SAN should also provide primitives that help storage-consumers to ensure data availability: multicast is one such primitive that helps in providing availability using replication. However, properties provided by the traditional hardware multicast are not *sufficient* to ensure the mutual consistency of replicated copies; often times the network multicast is combined with protocols providing stronger guarantees like failure atomicity and message ordering to handle replication more effectively. If a SAN is to provide stronger multicast guarantees, it will ease the effort needed in providing transparent, block level replication.

Storage Sharing An I/O architecture permitting sharing enables seamless fail-over of the storage-consumers that share data sets. Thus storage sharing is crucial to provide uninterrupted service. Shared storage architectures also provide better scalability since storage capacity and processing power could be added dynamically to the pool by adding more storage-consumers and storage. Additionally, data sharing gives high flexibility for dynamic load balancing since the data is uniformly accessible from any storage-consumer. Sharing also facilitates storage consolidation which reduces management and operation costs while increasing system usage.

But to achieve effective data sharing, SANs may need to assure certain properties at the network level. For instance, in order for a storage-consumer to failover correctly, SAN may need to ensure mechanisms such as I/O fencing. In fact, many commercial parallel database clusters expect the underlying cluster transport to provide I/O fencing. Also, concurrent access to the shared storage has to be mediated through certain concurrency control mechanisms. Neither FC nor iSCSI provide any concurrency control primitives; they do not provide I/O fencing also.

Security The traditional, bus-based based storage-consumers are built assuming that the connected storage is inherently secure. But in a distributed storage system like SAN, such assumptions hold no longer true. In order to bridge the easy migration path for legacy systems, which were built assuming the physical security of storage, SAN should provide means of enforcing the needed security by other means, say, cryptographically. Presentday FC SANs are built around relatively secure fiber transport and are not yet equipped to enforce cryptographic security. On the other hand, iSCSI SANs – which are built around insecure IP networks – enforce the necessary security using cryptographic protocols; to this effect, IPSec/IKE have been chosen as the cryptographic infrastructure for iSCSI SANs.

But, IPSec has many problems that are yet to be resolved [8]; so does IKE ([18], [23]). Also, the iSCSI level CRC mechanism and TCP checksum do not co-exist harmoniously owing to strict layering restrictions; with multiple TCP connections and SCSI command ordering in place, handling CRC error induced resynchronizations efficiently could get problematic. Comprehensive IP multicast security is still very much in infancy; of the many suggested key management protocols, only SKIP [2] discusses security in multicast communications explicitly. However, the problem concerning SKIP, and in general, IP multicast is the fact that they are *membership unaware* – a potentially inappropriate design for a restricted environment like SAN.

Intelligence Traditional storage-consumers interact with the storage using standard storage protocols like SCSI. The storage-consumers are unaware of the underlying storage technology. This enables an easy migration path from a direct access storage system to a SAN. However, the converse is that a SAN is unaware of storage-consumers that could result in poor performance. For instance, in a shared storage system like SAN, the correctness criterion for permissible concurrent interleaving is very much storage-consumer/application dependent.

If a SAN uses strong consistency models like linearizability [16] as default, it may be grossly inefficient since there are many storage-consumers who can manage with much weaker consistency guarantees. Thus a SAN should be intelligent enough to deploy the right concurrency control mechanism that is sufficient for the storage-consumer's requirements; this is especially important as different consistency mechanisms may incur different cost/performance tradeoffs [15]. In general, the data access path of a particular storage-consumer should be efficiently tailored to the exact needs of the storage-consumer and SANs should have provisions for doing so.

Though the “Keep it Simple, Stupid!” [21] approach works well in networks, it may not always be efficient: there are certain problems that do not permit efficient solutions in systems designed strictly using end-to-end arguments. QoS, multicast and VPN are such problems whose efficient solutions demand certain amount of intelligence in the intermediate nodes or *switches*. Thus, SANs should have enough intelligence to handle these critical problems efficiently.

3 Architecture

*i*SAN uses Type I Logical Link Control (LLC) [11] for transport, Group Communication System (GCS) for membership services and VLAN for grouping; VLANs in *i*SAN, pruned based on MAC address, provide efficient application/SCSI level routing. The edge switches that are part of a VLAN form a group with membership respecting strong virtual synchrony (Fig. 1). The group end-point that is housed in an edge switch, called *Sanlet*, acts as a SCSI Target emulator: SCSI commands sent by storage-consumers are received by the Sanlet and are dispatched to the appropriate physical storage device. Thus Sanlet can be seen as a thin Volume Manager residing in the edge switch that manages the virtualized storage. Fig. 2 depicts the flow of data in *i*SAN.

In *i*SAN, service discrimination is done by allowing a group to have a *tailored* protocol stack that matches the storage-consumers. We argue that this approach – called protocol composition, provides enough flexibility so that storage-consumers with very different goals can potentially agree on sharing of a common infrastructure, within which their commonality is captured by layers that they share, and their differences reflected by layers built specifically for their needs. Since the VLAN, by construction, houses storage-consumers with similar requirements, it is natural to use the VLAN tag to choose between various built-in protocol stacks. Presently all *i*SAN stacks share a common set of lower layers – the set formed by set of microprotocols that assure strong virtual synchrony; any further service specialization is done on the foundation of virtual synchrony.

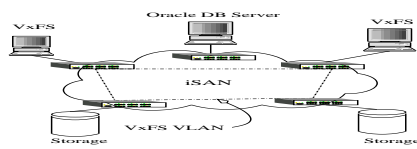
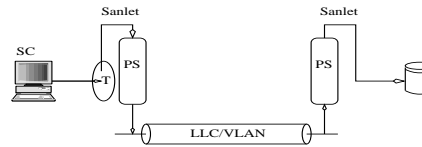


Fig. 1. Usage of VLANs in iSAN



SC–Storage Consumer; S–Storage; PS–Protocol Stack; T–Target

Fig. 2. Flow of data in iSAN

3.1 Design Criteria Revisited

The *iSAN* architecture proposed is interoperable and is likely to provide better throughput and latency guarantees as it is based on standards such as 802.3 LLC and VLAN. By providing virtual synchrony and stronger multicast guarantees (atomicity and ordering), *iSAN* effectively facilitates replication based high availability schemes. However, the high availability of *iSAN* infrastructure itself does not directly follow from the proposed architecture, and we do not discuss this aspect of *iSAN* any further in this paper. Based on strong virtual synchrony model, *iSAN* automatically provides basic sharing protection like I/O fencing. Other critical aspects of sharing like concurrency control are discussed in section 5.1. As discussed in section 5.2, *iSAN* deploys Ensemble’s *fortress* model of security which is amenable to proofs of correctness. Above all, the protocol composition mechanism presents *iSAN* with an efficient means of distinguishing and servicing different storage-consumers effectively.

4 Implementation

iSAN is implemented using Ensemble group communication system [10] and Linux kernel v2.4. The reason for choosing Ensemble are many fold. First, it is a well engineered GCS toolkit with a number of unique functionalities such as support for composable protocol stacks. Secondly, Ensemble has a powerful, provably correct, multicast aware security infrastructure [19]. Besides, Ensemble is event driven and is considered to be superior to thread based systems like Horus .

Ensemble is implemented using OCaml and is provided as a user level library that could be linked to an Ensemble application. But for our purposes, we needed a version of Ensemble that is written in C so that we could port it to Linux kernel. C-Ensemble serves this purpose precisely. We ported C-Ensemble distribution to Linux kernel with minimal changes: This is achieved by wrapping system calls to provide a libc like interface accessible from inside the kernel and by rewriting the event handling code. We also ported the Ensemble’s total, causal ordering protocols to C-Ensemble Linux kernel port because C-Ensemble provides only part of Ensembles’ functionality. We added a simple application level credit based flow control protocol to C-Ensemble. We modified both Ensemble and C-Ensemble to include LLC transport provided by Linux native LLC implementation. However, we have not yet ported the security protocols of Ensemble to C-Ensemble. Hence, *iSAN* currently uses the Ensemble/user (v1.33) for the security experiments while the other experiments are done with C-Ensemble/kernel. Sanlet uses a simple implementation of the target emulator software to emulate the SCSI target. This has been written from scratch to be Ensemble friendly. Please consult [17] for further information on implementation and configuring *iSAN*.

5 Case Studies

This section discusses the implementation of two critical storage services – concurrency control and storage security – in *iSAN* and shows how the advantages of the *iSAN*

approach. We demonstrate that these two critical services, though seemingly dissimilar, can be implemented efficiently in an *i*SAN. We believe that this diversity speaks for the generality and expressibility of *i*SAN.

5.1 Concurrency Control

In a distributed, shared storage system like SAN, the logical view of the storage seen by the storage-consumer can be very much different from the physical view. For example, what the storage-consumers see as a contiguous blocks of storage may not be contiguous at all; worse, may not even be from a single storage device. This is because the underlying storage system may transparently offer functionalities like striping and virtualization. Also, storage systems might impose hidden relationships among the stored data, for example, in the form of shared parity blocks which needs careful interleaving of I/O accesses. Thus, unless proper care is taken to resolve concurrent stripe accesses, a storage-consumer may see inconsistent data irrespective of the fact that it may itself be orchestrating some concurrency control mechanism at higher levels [1].

Concurrency control is the activity of coordinating the actions of processes that operate in parallel, access shared data, and therefore potentially interfere with each other [3]. The unit of a concurrent access, called transaction, consists of several lower level operations which are expected to be executed *atomically*. There are four types of concurrency control schemes that are prevalent in the literature – locking, timestamp ordering(TO)², optimistic and hybrid.

Of the four afore mentioned schemes, TO emerges out as the optimal mechanism for shared storage ([1], [24]). However, there are atleast three problems that are associated with TO. First, it requires synchronized clocks. Highspeed networks like SAN may not increase the synchronization accuracy dramatically for atleast two reasons: synchronization accuracy is bound by message delay variance and not by the absolute delay ([14], [4]); also, clock synchronization messages – being few tens of bytes long, may not see significant latency reduction even in gigabit networks. However, highspeed networks will need to handle higher number of active transactions in a given time-slice and hence require *better* clocks. Second, if the transactions are to come in some wildly different order from the original issue order, TO will reject many transactions; [7] shows that probability of such an occurrence could be high. The magnitude of network reordering depends on the existence of redundant links, their configuration and network load; not all of them are completely controllable. Thirdly, the order of transaction executions as governed by the TO scheduler may or may not be conforming to certain expected ordering like causal order [13], depending upon the granularity of clock synchronization and the delay characteristics prevailing. Fig. 3 depicts how causal violation could happen in a master slave clock synchronization setting. Thus, if one needs deterministic ordering of messages, TO cannot be used to enforce such ordering reliably.

*i*SAN uses message ordering protocols for concurrency control³. It is understood that different ordering mechanisms – FIFO, casual, causally constrained total order and unconstrained total order⁴ – guarantee different consistency semantics with cost and strictness increasing in that order. *i*SAN, being storage-consumer aware, deploys the suitable message ordering that *suffices* the storage-consumer's requirements. To this effect, we have considered five classes of storage-consumers and mapped their requirements to particular message ordering protocol.

² The timestamp ordering discussed here and elsewhere in the paper is Basic Timestamp Ordering; we also assume strict schedules.

³ We assume that the storage device commits operations in issue order. This is not a unreasonable assumption as most of the commercial RAID systems guarantee this.

⁴ A total order that respects causal order is referred as causally constrained total order while a total order that may not respect causal order is referred unconstrained total order.

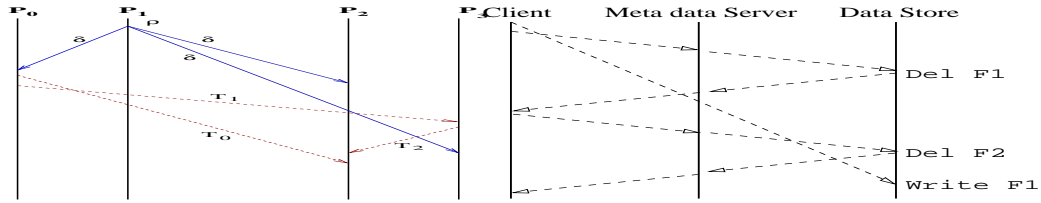


Fig. 3. Master P_1 broadcasts the recent clock reading δ , which includes the drift ρ , to slaves. T_i represents the clock reading prefixed to the message(m_i) sent. Since these two events, clock correction and message exchange, happens independently, causality violation may occur.

Fig. 4. After sending the *write F1* to the data store, the application deletes the file (*delete F1*) and creates a new file using *create F2*. The metadata server allocates the deallocated blocks from F1 to F2. Thus a delayed *write F1* corrupts the F2's data that was freshly written.

Parallel File Systems Parallel File Systems (PFS) cater to the I/O requirements of multiprocessor/computer systems. Traditionally, PFS is organised as a set of clients – where the applications run, and servers – which serve the storage. Most PFSs do not favour client side caching and it is to be noted that the PFS clients *can* tolerate minor inconsistencies in the shared state when the conflicts occur. In a PFS VLAN, *iSAN* will not provide any ordering save the FIFO ordering of stripe updates between the client and the IOD. This provides the expected behaviour with little or no additional cost. If one is to deploy TO or strict 2PL, the overhead is very likely to be high as it provides *stricter* consistency guarantee than what is actually needed. Thus, by making use of the semantics of PFS like filesystem, *iSAN* *increases* the amount of concurrency available for shared accesses, leading to better performance.

Hybrid Storage In many of the SAN based architectures, SAN is hidden behind the filesystems or database servers and the effective throughput seen by the clients is thus still limited at the rate at which these storage-consumers are able to cater to the requests. One way to solve the problem is to let the clients to access storage directly while expecting the storage-consumer to maintain the necessary metadata including the block map information; once the client gets the metadata information from the storage-consumer, say after a file open, it can access the storage directly. Any metadata data update will *still* involve the storage-consumer while the data is accessed directly from the storage. This architecture will be referred here as hybrid storage (often called as out-of-band virtualization).

Given that metadata server and the client may both access the storage simultaneously, one needs to ensure correct interleaving of these operations; Fig. 4 depicts one such problem case that would arise otherwise. A closer examination of Fig. 4 reveals that the problem is indeed due to causal violation: *delete F1* should have been delivered after *write F1* message as the latter *causally precedes* the former. So, in hybrid storage, Sanlets will enforce causal ordering of requests. Such mechanisms would improve the asynchrony of the system while adding very little overhead.

Database Systems Database systems do not favour one serial schedule over the other: all the strict executions are equally correct. Yet, in order to avoid distributed deadlocks, a DB may want to prune a total order out of conflicting transactions as in TO. However, the overhead of synchronizing clocks could be averted if one is to use unconstrained total ordering protocol in place of clocks. The Sanlet connecting DB to storage will thus need to enforce the unconstrained total order and, as a side effect, will solve the problem of mis-ordered transactions. [24] provides a total order based concurrency control protocol that is readily deployable in *iSAN*.

Replication Replication is an area of interest to both filesystems and databases and hence is of interest to *iSAN*. Replication protocols come in variety of forms and hues, differing in aspects like models, assumptions, mechanisms, guarantees provided, and

implementation ([28], [29]). Many replication schemes, notably the lazy schemes, will require certain update ordering and in *iSAN* this ordering is effected by using the causally constrained or unconstrained total ordering.

Log Enhanced Filesystems Log enhanced filesystems like VxFS needs to commit the metadata changes to log before it starts making changes to the on-disk filesystem structure. But in order to ensure that the log writes reach the disk before filesystem updates, the log is usually written *synchronously*. Thus for every metadata change, the filesystem suffers a synchronous log write. But, if the underlying storage layer, i.e. *iSAN*, is to provide FIFO ordering of commands, the filesystem need not have to write the log records synchronously; it only has to *queue* the log write before the corresponding metadata update. Since the storage layer assures FIFO delivery of commands, by the time the metadata updates reach the disk, the *previously* scheduled log writes would have reached the disk too. Thus, providing FIFO ordering at *iSAN* layer would improve the observed filesystem throughput of a journaling filesystem.

5.2 Storage Security

Organizations increasingly depend on their storage infrastructure for storing critical information. Thus the I/O subsystem should *understand* the sensitivity of the data its serving and should ensure confidentiality, integrity, and availability of the data both in-storage and in-transit. Please consult [17] for further details.

6 Performance

This section describes the experimental setup and results of the conducted experiments. The setup consists of Intel machines ($\geq 700\text{Mhz}$) running Linux (v2.4) acting as “edge switches”; these edge switches are connected using a 100Mbps Ethernet. The Target and Initiator are housed in the same switch to reduce the network interruption; it is a priori observed that doing so does not significantly change the performance profile. A FC JBOD, organized as a RAID 5 with three disks, is connected to one of the “edge switches” and is used as the data store. Block level traces for VxFS are generated in a Ultra Sparc machine running Solaris by running 4 benchmarks – ssh, ssl, gcc and postmark [12]. The other traces used are the HP traces [20]. For a detailed description of experimental setup and the related information, please consult [17].

	fifo	causal	total	total+causal
#3 %	0	7.36	17.43	30.06
#4 %	0	3.64	16.04	20.23

Table 1. CCTRL - Relative cost of different ordering protocols (with FIFO as base)

	ssl	ssh	gcc	postmark
%	0.97	1.38	0.41	4.73

Table 3. CBC Vs ECB – Overhead

	ssl	ssh	gcc	postmark
%	11.46	20.90	13.64	15.05

Table 2. CCTRL - Ordered Vs Sync Writes - throughput improvement observed

	ssl	ssh	gcc	postmark
	28.90	29.67	16.67	11.29

Table 4. SEC - Throughput improvement for 7% plain data

All the above tables depict the percentage of throughput improvement achieved. Table 1 shows that the *relative* overhead of different ordering protocols (with FIFO as the base) is indeed significant; the rows are indexed by the cardinality of the group and the traces used are HP traces. Since we did not have any shared traces, we used the 3 disk

streams in HP to emulate shared access. The results signify that the storage consumers indeed benefit from the selective deployment of ordering mechanisms. For instance, causally constrained total order is costlier by 30% compared to FIFO for a group size of 3. However, increasing the group cardinality reduces this performance disparity. This is due to the fact that 100Mbps Ethernet do not have efficient flow control. Lack of proper flow control at the lower layer significantly penalises the low overhead/high throughput FIFO ordering and it explains the reduction in relative overhead. Table 2 compares the performance of ordered log writes compares and synchronous log writes. This supports our argument that ordering of I/O commands helps even in a non distributed setting.

For the security experiments, the throughput difference between ECB-for-all and CBC-for-journal-alone are observed to be with less than 5% range (Table 3). *i*SAN thus achieves increased security at almost negligible cost. The results of experiments wherein the storage-consumer/driver controlling the Sanlet *dynamically* using in-band messages are depicted in table 4. The experiment is conducted by allowing roughly 7%⁵ of the block access to be transmitted in plain text; this is because the block level traces generated do not have file names and their attributes. The 7% of block access that are transmitted in plain is uniformly distributed across the total accesses. The results show that the throughput improvement observed is indeed significant.

7 Related Work

Intelligence in *i*SAN is achieved using dynamic protocol composition which stands comfortably midway between the Turing complete Active Networks [27] and the quasi-static Programmable Networks [6]. Composition, unlike Programmable Networks, provides non trivial specialization, yet, unlike Active Networks, can be very efficient and secure. A work that is very similar in spirit to the proposed architecture is that of Virtual Overlay Networks (VON) [5]. However, our work is novel for many reasons: First, the architecture proposed in [5] is generic while our architecture is tuned to the requirements of SAN. Secondly, [5] is abstract and leaves many engineering issues like selection of the minimal Overlay Network, the vantage point where the code-stubs to be deployed, mode of contacting the code stub etc. open. Our architecture is more concrete, pinning down these crucial design parameters. Thirdly, to our knowledge, ours is the *first* application/realization of [5]. For a detailed comparison of similar works and *i*SAN, please consult [17].

8 Conclusions

In this paper, we have presented a design and implementation of an intelligent SAN architecture. We have demonstrated how this architecture can be used to efficiently solve some of the critical problems associated with conventional SANs, and evaluated the suitability of solutions. We would like to conclude that *i*SAN approach of architecting SAN shows great promise as a means of constructing efficient, yet, flexible SANs.

In long run, we would like to extend *i*SAN design to investigate the following aspects. First, we plan to add a scalable and manageable virtualization architecture to *i*SAN. The idea is to balance the virtualization overhead with the consumer-awareness of *i*SAN to arrive at a low cost virtualization scheme. Also, we plan to investigate how increasing asynchrony/concurrency at lower levels would improve performance in the higher layers and to suggest such a scheme as a design principle. Finally, through *i*SAN research, we are attempting to understand the synergy between virtual synchrony and filesystems.

⁵ This number we have arrived at after observing the amount of public domain data that we found in our lab machines.

References

1. Khalil Amiri, Garth Gibson, and Richard Golding. Highly concurrent shared storage. In *ICDCS*, April 2000.
2. Ashar Aziz, Tom Markson, and Hemma Prafullchandra. Simple key management for internet protocols (<http://www.skip.org/>), 1998.
3. Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. 1987.
4. S Biaz and J L. Welch. Closed form bounds for clock synchronization under simple uncertainty assumption. *Info. Processing Letters*, 2001.
5. Ken Birman. Technology requirements for virtual overlay networks. *IEEE Systems, Man and Cybernetics: Special issue on Information Assurance, Vol. 31, No 4*, July 2001.
6. Jit Biswas, Jean-Francois Huard, Aurel Lazar, Koonseng Lim, Semir Mahjoub, Louis-Francois Pau, Masaaki Suzuki, Soren Torstensson, Wang Weiguang, and Steve Weinstein. Application programming interfaces for networks - IEEE P1520, January 1999.
7. C. Bouras and P. Spirakis. Performance modeling of distributed timestamp ordering: Perfect and imperfect clocks. In *Performance Evaluation Journal, Elsevier Science*, April 1996.
8. Niels Ferguson, , and Bruce Schneier. A cryptographic evaluation of IPSec, February 1999.
9. Andrew Gallatin, Jeff Chase, and Ken Yocum. Trapeze/IP: TCP/IP at near-gigabit speeds. In *USENIX Technical Conference*, June 1999.
10. M. Hayden. *The Ensemble System*. PhD thesis, Department of Computer Science, Cornell University, 1998.
11. ISO. Logical link control - ISO/IRC 8802-2.
12. J. Katcher. Postmark: A new file system benchmark. Technical Report TR3022, Network Appliance Inc., october 1997.
13. Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communication of the ACM, vol. 21, no. 7*, July 1978.
14. J Lundelius and N Lynch. An upper and lower bound for clock synchronization. *Information and Control, Vol. 62, Nos. 2/3*, September 1984.
15. David Mosberger. Memory consistency models. *Operating Systems Review*, 1993.
16. Herlihy M.P and Wing J.M. Linearizability: a correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems, 12(3)*, October 1990.
17. Ganesh Narayan and K. Gopinath. iSAN - an intelligent storage area network architecture. Technical Report TR-IISc-CSA-2004-6, Department of Computer Science and Automation, Indian Institute of Science, August 2004.
18. Radia Perlman and Charlie Kaufman. Analysis of the ipsec key exchange standard. *IEEE Internet Computing 4(6)*, November 2000.
19. Ohad Rodeh, Ken Birman, and Danny Dolev. The architecture and performance of security protocols in the ensemble group communication system. Technical Report TR2000-1822, Department of Computer Science, Cornell University, October 2001.
20. Chris Ruemmler and John Wilkes. Unix disk access patterns. Technical report, HP labs, December 1992.
21. J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM TOCS 2(4)*, November 1984.
22. Julian Satran, Kalman Meth, Costa Sapuntzakis, Mallikarjun Chadalapaka, and Efri Zeidner. iSCSI, 2001.
23. W A Simpson. IKE/ISAKMP considered dangerous, June 1999.
24. Rashmi Srinivasa. *Network-Aided Concurrency Control in Distributed Databases*. PhD thesis, University of Virginia, January 2002.
25. Nishan Systems. Data storage anywhere, any time - metro and wide area storage networking with nishan systems ip storage switches, 2000.
26. ANSI NCITS T10/1144D. Fibre channel protocol for scsi, second version (FCP-2) , revision 5, November 2000.
27. David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A survey of active network research. *IEEE Communications Magazine Vol. 35, No. 1*, January 1997.
28. M Wiesman, F Pedone, A Schiper, B Kemme, and G Alonso. Database replication techniques: a three parameter classification. In *19th IEEE SRDS*, October 2000.
29. M Wiesmann, F Pedone, A Schiper, B Kemme, and G Alonso. Understanding replication in databases and distributed systems. In *ICDCS*, April 2000.