Evaluating Availability of Networked Storages Using Commercial Workloads

Ming Zhang and Qing Yang
Dept. of Electrical and Computer Engineering
University of Rhode Island
Kingston, RI 02881 USA
{mingz, qyang}@ele.uri.edu

Abstract—**This paper presents an availability study of storage area network (SAN) systems built on TCP/IP networks. We use a new benchmark tool called N-SPEK (Networked-Storage Performability Evaluation Kernel-module) to measure performance dynamics under commercial workloads with various faulty conditions of different parts of a SAN system. By injecting network faults, controller faults, and disk faults into the SAN system, we observe and analyze system availability in terms of performability (performance + availability). Two specific SAN systems are considered in this paper: iSCSI-based SAN and STICS (SCSI-To-IP Cache Storage) [1] based SAN. Experiments are carried out to measure and compare performability of the two SAN systems under different faulty conditions. It is observed that STICS-based SAN shows more stable performance and better availability than iSCSI-based SAN when network faults are injected.**

I. Introduction

It has been widely recognized in the computer architecture community that system availability is one of the most critically important areas. This is particularly true for data storages that are the center of information services. Financial companies such as brokerage and credit card services may lose as much as 9.1 million dollars for one hour of down time [2]. Yet, there is very little research work done on availability evaluation of data storage systems.

In networked information services, data storages are generally connected to servers in three different ways: directly attached (DAS), network attached (NAS) [3], [4], [5], and storage area network (SAN) [6]. A recent research work done by Brown and Patterson [7] presented a case study on evaluating availability of a software RAID system by injecting disk related faults into the software RAID system. The RAID system they considered is a typical DAS storage, i.e. the RAID is connected to a SCSI interface of a server. NAS storage provides data storage at a file system level through a standard Ethernet connection to servers or clients. The performance behavior and availability of a NAS can be considered to be similar to file servers to some extend. Therefore, the performability study of cluster-based servers [8] provides a fairly good guideline for availability analysis of NAS. SAN system, on the other hands, is a set of storage devices interconnected through a specialized network such as a FC fabric or iSCSI over TCP/IP. Servers share storages through this special network. Data availability of such SAN depends on many factors including network components such as switches, bridges, and cables, controller cards such as NIC (network interface card) and HBA (host bus adapter), and storage devices such as disk arrays, tapes, and CD towers etc. Little is understood about the effects of faults at various parts of a SAN on its availability in the research community. To the best of our knowledge, there is no published study on availability analysis of a SAN under various faulty conditions of different parts of the system.

In this paper, we present an availability analysis of SAN systems using commercial workloads. We consider two types of SAN systems built on TCP/IP network. The first type of SAN is based on iSCSI [9] technology, an emerging standard for building SAN over IP. The other SAN is based on STICS (SCSI-To-Internet Cache Storage) [1], a new technology that enhances the performance and reliability of iSCSI-based SANs. We first set up the SAN systems in our laboratory using iSCSI and STICS technologies for the purpose of availability measurements. Four types of faults are injected into the SAN systems at three major layers of the SAN: network, controllers and storage devices. Some of these faults completely crash the system making data completely unavailable for service. Some faults may result in degraded system performance temporarily while still having partial data availability. We therefore measure the system availability in terms of performance dynamics over time under various faulty conditions. The performance measures we use here are system throughput in terms of megabytes of data transferred between a server and storage per second. We compare the availability of the two SAN systems to show that STICS demonstrated better availability than standard iSCSI under network faults because of packet filtering and efficient data caching.

II. SAN Architectures Studied

We consider two types of SANs in this paper for availability evaluation, namely iSCSI-based SAN and STICS-based SAN.

A. *iSCSI-based SAN*

Traditionally, a SAN consists of networked storage devices interconnected through a dedicated Fiber Channel (FC-4 protocol) network. The basic premise of a SAN is to replace the "point-to-point" infrastructure of server to storage communications with one that allows "any-to-any" communications. A SAN provides high connectivity, scalability, and availability using a specialized network protocol: FC-4 protocol. Deploying such a specialized network usually introduces additional cost for implementation, maintenance, and management. iSCSI is the most recently emerging technology with the goal of implementing the SAN technology over the better-understood and mature network infrastructure: the Internet (TCP/IP). The iSCSI protocol is a mapping of the SCSI remote procedure invocation model on top of the TCP protocol. It is a connection oriented command/response protocol between an iSCSI initiator and an iSCSI target. An iSCSI session begins with an iSCSI initiator connecting to an iSCSI target through a TCP connection to set up a persistent state between the initiator and the target. Once the initialization process is complete, the iSCSI session continues in full feature phase by exchanging commands and data that are encapsulated in the iSCSI protocol over its TCP connection.

Implementing a SAN using iSCSI is fairly straightforward. Servers that share a pool of storages are connected to the storages using a standard TCP/IP network. Each server runs as an iSCSI initiator while each storage device acts as an iSCSI target with unique WWUI (World-Wide Unique Identifier). Currently, there are basically two existing approaches for implementing iSCSI over IP: one carries out SCSI and IP protocol conversion at a specialized switch and the other encapsulates SCSI protocol in TCP/IP at host bus adapter (HBA) level. Both approaches have severe performance limitations. Converting protocols at a switch places special burden to an already-overloaded switch and creates another specialized networking equipment in a SAN. Such a specialized switch not only is costly as compared to off-the-shelf Ethernet switches but also complicates installation, management, and maintenance. To encapsulate SCSI protocol over IP requires significant amount of overhead traffic for SCSI commands transfers and handshaking over the Internet. On a typical iSCSI implementation, we have measured around 58% of TCP/IP packets being less than 127 bytes long, implying an overwhelming quantity of small size packets to transfer SCSI commands and status (most of them are only one byte). Majority of such small packet traffic over the net is not necessary because of the reliable and connection-oriented services provided by underlying TCP/IP. Our experiments [1] using PostMark benchmark have shown that efficient caching can reduce total number of packets transferred over the net from 3,353,821 to 839,100 for same amount of remote storage data, a 4 times reduction!

B. *STICS-Based SAN*

Using iSCSI to implement SAN over IP brings economy and convenience whereas it also raises performance issues. We have recently proposed a new storage architecture: SCSI-To-IP Cache Storage, or STICS for short [1]. The purpose of STICS is to bridge the disparities between SCSI and IP so that efficient SAN systems can be built over the Internet. Besides caching storage data, STICS also localizes SCSI commands and handshaking operations to reduce unnecessary traffic over the Internet. In this way, it acts as a storage filter to discard a fraction of the data that would otherwise move across the Internet, reducing the bottleneck problem imposed by limited Internet bandwidth and increasing storage data transfer rate.

A typical STICS box consists of a disk and an intelligent processing unit with an embedded processor and sufficient RAM. It has two standard interfaces: one is a SCSI interface and the other is a standard Ethernet interface. Via the SCSI interface, STICS may run under two different modes: initiator mode or target mode. When a STICS is used to connect to a host, it runs in target mode receiving requests from the host, carrying out the IO processing possibly through the network, and sending back results to the host. In this case, the STICS acts as a directly

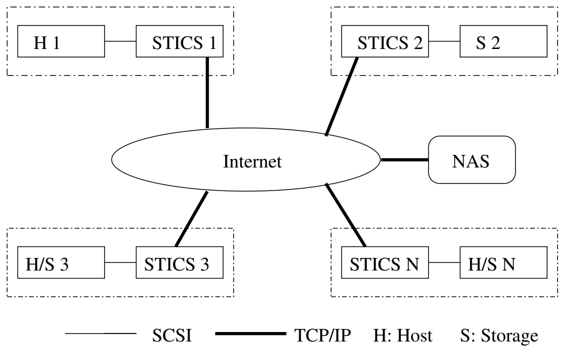

Fig. 1. STICS overview. A STICS connects to a host via a SCSI interface and connects to other STICS or NAS via the Internet or a TCP/IP network

attached storage device to the host. When a STICS is used to connect to a storage device such as a disk or RAID to extend storage, it runs in initiator mode, and it sends or forwards SCSI requests to the extended storage device. For example, in Figure 1, STICS 1 runs in target mode while STICS 2 runs in initiator mode. Via the network interface, a STICS can be connected to the Internet and share storage with other STICS's or network attached storages (NAS). The disk in a STICS box is used as a nonvolatile cache that caches data coming from possibly two directions: block data from the SCSI interface and network data from the Ethernet interface. In addition to standard SCSI and IP protocols running on the intelligent processing unit, it also implements a special caching algorithm controlling a two level cache hierarchy consisting of an NVRAM on top of a log disk.

Figure 1 shows a typical SAN implementation over IP using STICS. Any number of storage devices or server computers can be connected to the standard Internet through STICS to form a SAN. Instead of using a specialized network or specialized switch, STICS connects a regular host server or a storage device to the standard IP network (Ethernet). Consider STICS 1 in the diagram. It is directly connected to a SCSI HBA of Host 1 as a local storage device. It also acts as a cache and bridge to allow Host 1 to access, at the block level, any storage device connected to the SAN such as NAS, STICS 2, and STICS 3 etc. In order to allow a smooth data access from Host 1 to the SAN, STICS 1 provides SCSI protocol service, caching service, naming service, and IP protocol service.

III. Experimental Settings

Since we do not have SAN hardware facilities in our lab, we setup a software iSCSI-based SAN environment for measurement purpose. We choose the iSCSI implementation supporting iSCSI draft 18 that is downloaded from University of New Hampshire [10]. DISKIO mode is used in the iSCSI target allowing it to read/write real or emulated SCSI disks. The iSCSI initiator exports an emulated SCSI devices for the workload generator or STICS box that are discussed in more detail later. Our iSCSI and STICS experimental settings are shown in Figures 2 and 3, respectively. Five PCs are used in our experiments. All PCs are equipped with one Pentium III 866MHz CPU, 512M PC133 memory, and one Intel Pro1000 Gigabit NIC except for the bridge that has two NICs. An Intel NetStructure 470T Gigabit Switch is used to interconnect the PCs. The workload gener-

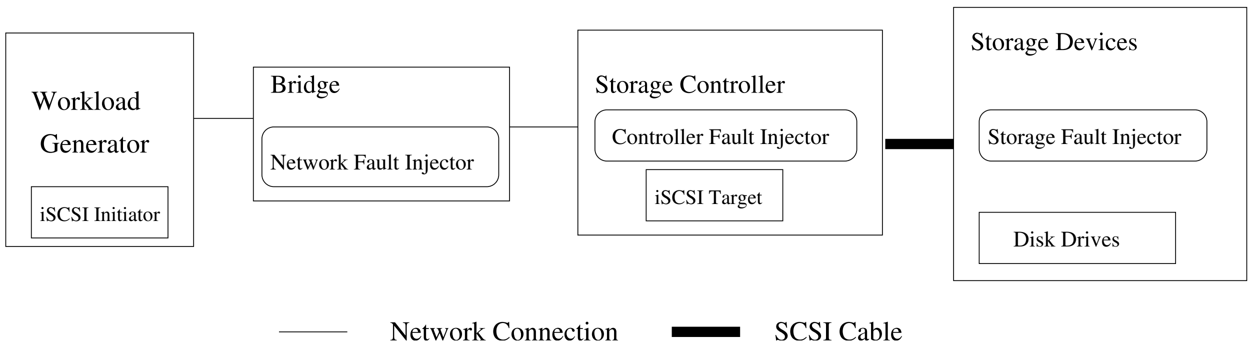

Fig. 2. Block diagram of iSCSI experiment settings. Each block in this diagram, including workload generator, bridge, storage controller, and storage devices, is a PC. The workload generator generates I/O requests to the emulated disk exported by the iSCSI initiator. The iSCSI initiator transmits all the requests to the iSCSI target via IP network and the iSCSI target performs these requests on the disk emulated by the storage fault injector.

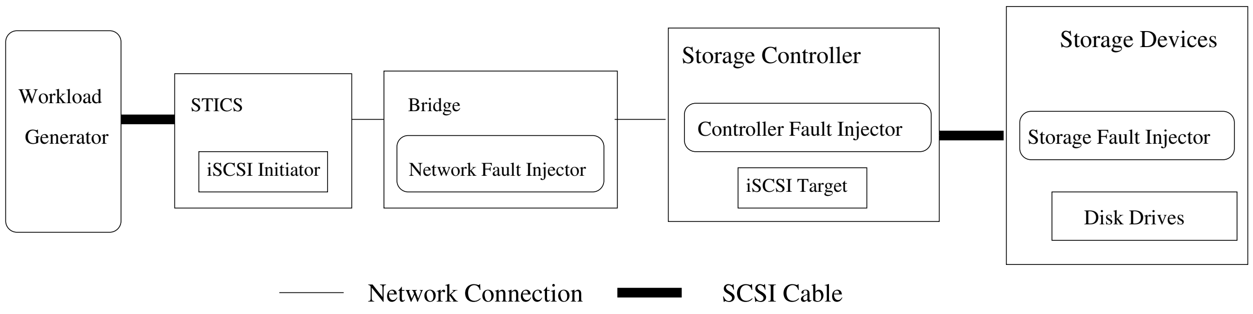

Fig. 3. Block diagram of STICS experiment settings. Each block in this diagram, including workload generator, STICS box, bridge, storage controller, and storage devices, is a PC. The workload generator generates I/O requests to the STICS box. The STICS box performs caching algorithms for these requests and uses the disk emulated by the iSCSI initiator. Other data flow is similar to the iSCSI environment shown in Figure 2.

ator in Figure 2 and the STICS box in Figure 3 are connected to the switch through the bridge using a crossover cable. All PCs run Redhat Linux 7.3 with recompiled standard 2.4.18 kernel except for the bridge that uses FreeBSD 4.6 and STICS box that uses recompiled standard 2.4.4 kernel.

N-SPEK (Networked-Storage Performability Evaluation Kernel-module) [11], a new benchmark tool that we developed, is used to carry out our measurement. N-SPEK consists of a controller, a worker, one or several probers, and different types of fault injection modules. An N-SPEK Controller resides on a controller machine that is used to coordinate N-SPEK Worker and Probers. It can start/stop N-SPEK Worker and Probers, send commands and receive responses from them. A Java GUI interface of the Controller allows a user to input configuration parameters such as workload characteristics and to view measured results. It also has a data analysis module to analyze measured data. In our experiment, we run it on the same machine as the workload generator. The N-SPEK Worker running on a testing client generates storage I/O requests via the low level device driver and records performance data. It is a Linux kernel module running in kernel space with one main thread, one working thread, and one probe thread. The main thread is responsible for receiving instructions from N-SPEK Controller and controls the working thread to execute actual I/O operations. The working thread keeps sending SCSI requests that are eventually sent to remote targets by a lower level device driver. By using an event-driven architecture, it can perform several outstanding SCSI requests concurrently, which is useful and necessary when testing SCSI tagged commands feature and exploring the maximum throughput of a remote SCSI target. The probe thread records system status data periodically and reports to N-SPEK Controller once test completes. Comparing with existing file systems and disk I/O benchmark tools such as PostMark [12], IoMeter [13], and SPEC SFS [14], N-SPEK bypasses the file system layer and block device layer. As a result, measured performance data are not skewed by cache effects and processing overheads from these layers. By running in kernel space, N-SPEK minimizes overheads brought by system calls and context switches compared to other benchmark tools running in user space. On each storage controller, there is an N-SPEK Prober thread that records system status for post-processing. Its functionality is similar to probe thread in an N-SPEK Worker.

To evaluate system availability, we need to inject faults at various parts of the system. Fault injection is commonly used in fault-tolerance community to verify fault tolerant systems or study system behaviors [15], [16]. There are three types of fault injection modules in N-SPEK to support availability evaluation. By using these modules, users can introduce different types of faults to different parts of a networked storage system under test and measure the availability of the tested system at degraded mode. These modules are:

Network fault injector. It resides on a network bridge on the network path between a worker and the measured storage target. It injects unexpected events to network traffics traveling through the bridge by adding excessive delays and dropping packets with a configurable packet loss rate. Note that TCP provides reliable transport over the Internet through flow control, time-out and retransmission mechanisms. Many network faults including hardware failures and software failures in the network result in excessive delays at the transport layer. Injecting excessive delays at TCP layer mimics various network faults. We call this type of faults delay fault. Packet drops may result from hardware failures in the network, buffer overflows due to congestions or inadequate flow control in the SAN. Our fault injector makes use of a program that controls the existing dummynet [17] package in FreeBSD, a network traffic control and shaping software that was previously used by other researchers [18]. Another package, NIST Net [19] is also an excellent package that accomplishes similar functionalities. In our experiments, we only use fixed packet loss rate or network delay to generate network errors to simplify measurement. More calibrated experiments can be carried out under complex degraded network conditions with different packet delay distributions, congestion and background loss, bandwidth limitation, and packet reordering/duplication.

Storage fault injector. Its main purpose is to emulate a normal SCSI device that can be used directly by systems. The research work [7] on software RAID availability shows that a RAID system experiences a degraded performance period after one of its disk drives fails. During this period, the RAID system reconstructs itself while handling incoming requests as well. Our storage fault injector simulates the behaviors of such a RAID system. The storage fault injector finishes incoming requests at a normal speed at the beginning. At time point A, a disk is assumed to fail and the storage fault injector performs in a degraded mode that leads to a longer response time and lower throughput for incoming requests. At time point B, the simulated RAID is assumed to finish its reconstruction process and return to a normal status. In our current experiment, the storage fault injector is implemented as a RAM based virtual SCSI RAID and exports itself as a normal SCSI device to the system under test.

Controller fault injector. Besides hardware failures of a storage controller, major source of faults of a controller can be attributed to malfunction of CPU and RAM. Normal operations of a controller can be compromised if CPU and/or RAM resources are unavailable. Directed by configurable parameters, a controller fault injector can take away most of CPU and/or memory resources from normal storage controller operations by adding unrelated CPU loads and memory loads on the controller.

All these fault injectors are user configurable. One or more types of faults can be injected into and removed from the system at certain time points decided by users. For example, injecting a network delay fault during a certain period will result in delays of all packets that travel through the bridge during that period while packets traveling at other time are unaffected.

IV. Workloads Used in the Experiments

Two commercial workloads are used in our experiments, EMC trace and TPC-C. The EMC trace was collected by an EMC Symmetrix disk array system installed at a telecommunication customer site. The trace file contains more than 230k storage requests with a fixed request size of 2KB. The trace is write-dominated with 89% of operations being write operations. The average request rate is about 333 requests/second.

TPC-C trace is downloaded from the Trace Distribution Center of Performance Evaluation laboratory at Brigham Young University. They have run TPC-C benchmark with 20 data warehouses using Postgres database on Redhat Linux 7.1. The trace was collected using their kernel level disk trace tool, DTB [20], and was time stamped using jiffie [21] as a timer. On most Linux systems, the granularity of jiffie is only 1/100 second that is too coarse for timing purpose. Many requests are invoked with same jiffie values. We therefore change the granularity to 1/1000 second and distribute requests that have same jiffie values evenly during that jiffie. The trace file contains more than 3 million requests with size ranging from 4KB to 124KB. The average request size is about 92.4 KB. The trace is read-dominated with 99% of its operations being read operations. The average request rate of the modified TPC-C trace is about 282 requests/second.

V. Measurement Results

Our measurement results for the iSCSI-based SAN are shown in Figure 4 through Figure 12 and those for the STICS-based SAN are shown in Figures 13 and 14. Curves marked with triangles in these figures represent normal throughputs in terms of Mbyte per second without fault, those marked with "x" represent system throughputs under transient fault conditions, and those marked with circles represent throughputs under sticky fault conditions. Time point "A" marks the start of one or more types of transient faults being injected, and time point "B" marks the removal (or recovery) of the transient faults.

Figures 4, 5, 13, and 14 show throughputs variations over time when network faults were injected for EMC trace workloads and Figures 8 and 9 show the same for TPC-C workloads. For network delay faults, we added 4 ms delay to every packet going through the network bridge and for packet loss faults we set the packet drop rate to 1%. We noticed in Figures 4, 5, and 8 that when a network fault is injected, available system throughputs of the EMC trace dropped by 50%, whereas available throughput of the TCP-C trace dropped to 1/6 of the normal performance values. This observation indicates that network delay faults have significant impact on available throughputs when traffic intensity is high. Note that the traffic intensity of the TPC-C trace is much higher than the EMC trace in our experiments.

Figures 6 and 10 show available throughputs when transient controller faults were injected. A storage controller card hosts many codes such as RAID control code, iSCSI protocol stack, TCP/IP stack, and so on in addition to on-board OS. A software bug may result in CPU temporarily unavailable to normal processes. We use our controller fault injector to emulate such faults by taking away 98% of CPU resources. At point A of Figure 6, such a CPU fault is injected and results in throughput drop to about half. Similarly, significant throughput drop is observed in Figure 10 for the TPC-C trace. The available throughputs gradually go back to normal after the faults are removed (after time point "B") in both figures.

Impacts of disk faults are shown in Figure 7 for the EMC trace and Figure 11 for the TPC-C trace. When a disk failure occurs, the simulated RAID system is assumed to automatically replace the faulty disk with a hot spare. The recovery process is done on-line, which leads to a degraded throughput before the recovery ends. Because the traffic intensity of the EMC trace is not as high, the recovery process is fairly quick as shown in Figure 7. For high traffic workloads such as the TPC-C, we noticed a lengthy recovery process as shown in Figure 11 since the RAID system is loaded with both normal disk requests and background recovery process.

We have also carried out experiments on injecting multiple faults simultaneously to the iSCSI system. Figure 12 shows the available throughput changes when the multiple faults are injected with the TPC-C trace. At the 20th second, network delay fault, CPU fault, and disk fault are injected. At the 40th second, network delay fault is recovered; at the 60th second, CPU fault is recovered; and disk fault is recovered at the 80th second. As shown in Figure 12, when multiple faults are injected, the system is approaching totally unavailable. After both CPU fault and network fault are removed, the available throughput slowly increases. After all faults are removed, the system gradually goes back to normal status.

Figure 15 shows the availability comparison between iSCSI-based SAN and STICS-based SAN. The dash line gives the average throughput of STICS-based SAN with no fault as a reference. The curve marked with circles represents available throughput of the STICS-based SAN when sticky network delay faults are injected and the curve marked with "x" represent the same of the iSCSI-based SAN. It is observed in this figure that with the same faulty conditions, the STICS-based SAN gives much better throughput than the iSCSI-based SAN. Such a stable and better availability can be attributed to the fact that STICS combines many fragmented packets into good size ones reducing the network requirements. This observation is further evidenced in Figure 16 that shows the availability under transient network delay faults. Similar results are also observed for network packet loss faults as shown in Figures 17 and 18.

VI. Conclusions

In this paper, we have presented an availability measurement of two types of SAN systems, iSCSI-based SAN and STICS-based SAN, using commercial workloads. Fault injections are used to measure system availability under various failure conditions. Comparing with iSCSI, STICS improves system availability by efficient caching algorithms and localization of unnecessary protocol overheads. Currently, we are pursuing more adaptive caching, pre-fetching, and destaging algorithms for STICS to further improve the performability of STICS-based SAN systems. And we are also working on proposing comprehensive performability metrics and performability comparison methodology in the near future.

Acknowledgments

This research is sponsored in part by National Science Foundation under grants MIP-9714370 and CCR-0073377. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] X. He, Q. Yang, and M. Zhang, "Introducing SCSI-To-IP cache for storage area networks," in *Proceedings of the 2002 International Conference on Parallel Processing*, Vancouver, Canada, Aug. 2002, pp. 203–210.

[2] D. A. Patterson, "Availability and maintainability $\gg$ performance: New focus for a new century," keynote speech at Conference on File and Storage Technologies (FAST), Monterey, CA, Jan. 2002.

[3] G. Gibson and R. Meter, "Network attached storage architecture," in *Communications of the ACM*, vol. 43, Nov. 2000, pp. 37–45.

[4] E. L. Miller, W. E. Freeman, D. D. E. Long, and B. C. Reed, "Strong security for network-attached storage," in *Proc. of the Conference on Fast and Storage Technologies*, Monterey, CA, Jan. 2002, pp. 1–14.

[5] K. Magoutis, S. Addetia, A. Fedorova, M. Seltzer, J. Chase, A. Gallatin, R. Kisley, R. Wickremesinghe, and E. Gabber, "Structure and performance of the direct access file system(DAFS)," in *Proceedings of USENIX 2002 Annual Technical Conference*, Monterey, CA, June 2002, pp. 1–14.

[6] G. T. R. Khattar, M. Murphy and K. Nystrom, "Introduction to storage area network," Redbooks Publications (IBM), Tech. Rep. SG24-5470-00, Sept. 1999.

[7] A. Brown and D. A. Patterson, "Towards availability benchmarks: A case study of software RAID systems," in *Proceedings of the 2000 USENIX Annual Technical Conference*, San Diego, CA, June 2000.

[8] K. Nagaraja, N. Krishnan, R. Bianchini, R. Martin, and T. Nguyen, "Evaluating the impact of communication architecture on the performability of cluster-based services," in *Proceedings of the 9th International Symposium on High-Performance Computer Architecture (HPCA 9)*, Anaheim, CA, Feb. 2003.

[9] J. Satran, K. Meth, C. Sapuntzakis, M. Chadalapaka, and E. Zeidner. (2002) iSCSI draft standard. http://www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-18.txt.

[10] UNH. iSCSI reference implementation. [Online]. Available: http://www.iol.unh.edu/consortiums/iscsi/

[11] M. Zhang and Q. Yang, "N-SPEK: a performability benchmark tool for networked storage systems," Dept. ECE, Univ. of Rhode Island, Tech. Rep., Dec. 2002. [Online]. Available: http://www.ele.uri.edu/research/hpcl/nspek/

[12] J. Katcher, "PostMark: A new file system benchmark," Network Appliance, Tech. Rep. 3022, 1997.

[13] Intel. Iometer, performance analysis tool. [Online]. Available: http://www.intel.com/design/servers/devtools/iometer/

[14] SPEC. SPEC SFS benchmark. http://www.spec.org/osg/sfs97/.

[15] J. Arlat, A. Costes, Y. Crouzet, J.-C. Laprie, and D. Powell, "Fault injection and dependability evaluation of fault-tolerant systems," *IEEE Transactions on Computers*, vol. 42, no. 8, pp. 913–923, 1993.

[16] S. Dawson, F. Jahanian, and T. Mitton, "ORCHESTRA: A fault injection environment for distributed systems," University of Michigan, Tech. Rep. CSE-TR-318-96, 1996.

[17] L. Rizzo, "Dummynet: a simple approach to the evaluation of network protocols," *ACM Computer Communication Review*, vol. 27, no. 1, pp. 31–41, 1997.

[18] W. T. Ng, B. Hillyer, E. Shriver, E. Gabber, and B. Ozden, "Obtaining high performance for storage outsourcing," in *Proceedings of the Conference on File and Storage Technologies (FAST)*, Monterey, CA, Jan. 2002, pp. 145–158.

[19] NIST. NIST Net, network emulation package. [Online]. Available: http://snad.ncsl.nist.gov/itg/nistnet/

[20] Performance Evaluation Laboratory, Brigham Young University. DTB: Linux Disk Trace Buffer. [Online]. Available: http://traces.byu.edu/new/Tools/

[21] A. Rubini and J. Corbet, *Linux Device Drivers (2nd Edition)*. O'Reilly & Associates, 2001.

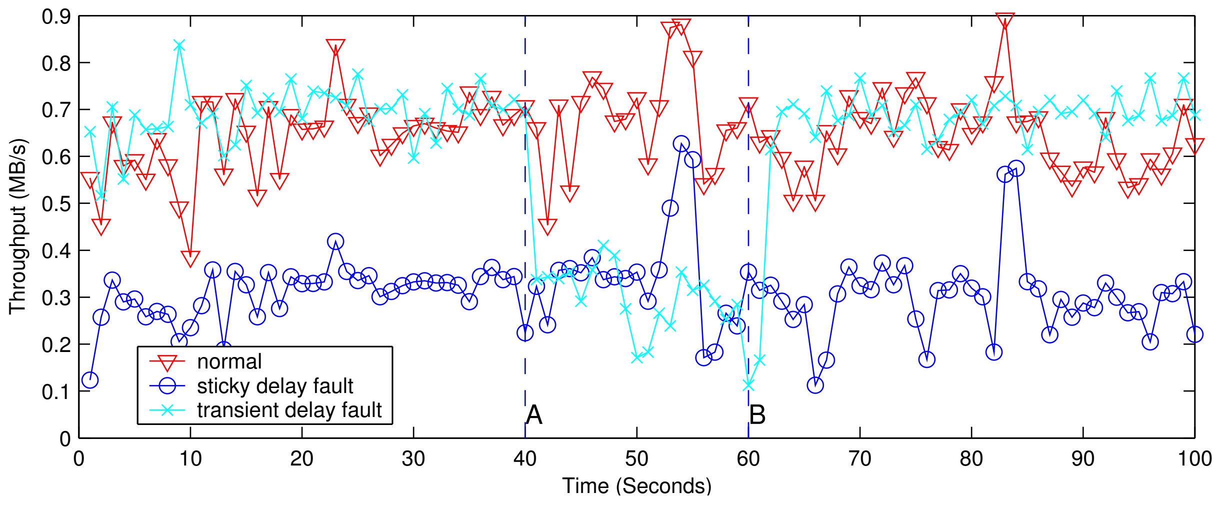

Fig. 4. iSCSI throughput with the EMC trace under network delay faults.

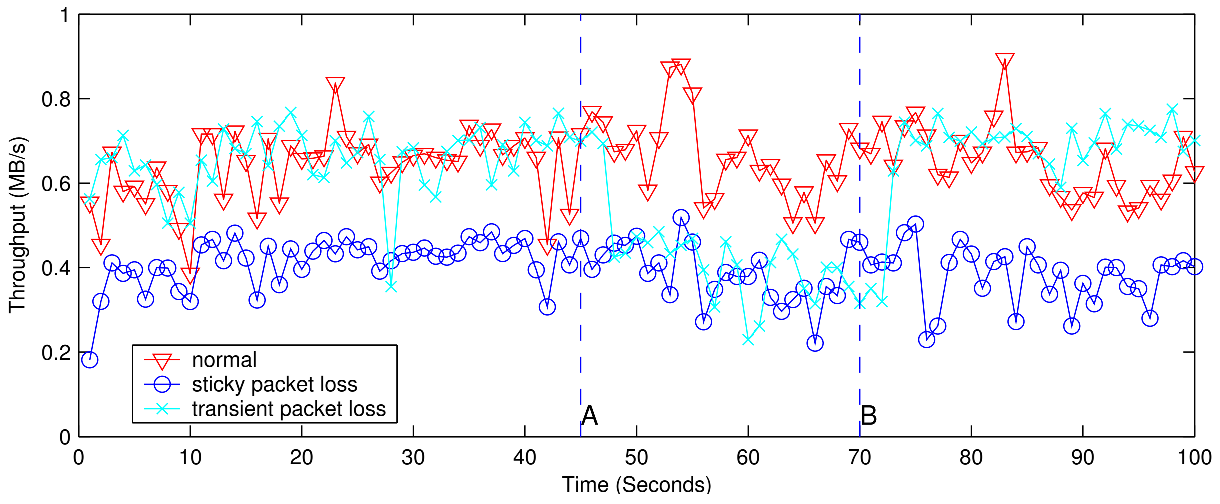

Fig. 5. iSCSI throughput with the EMC trace under network packet loss faults.

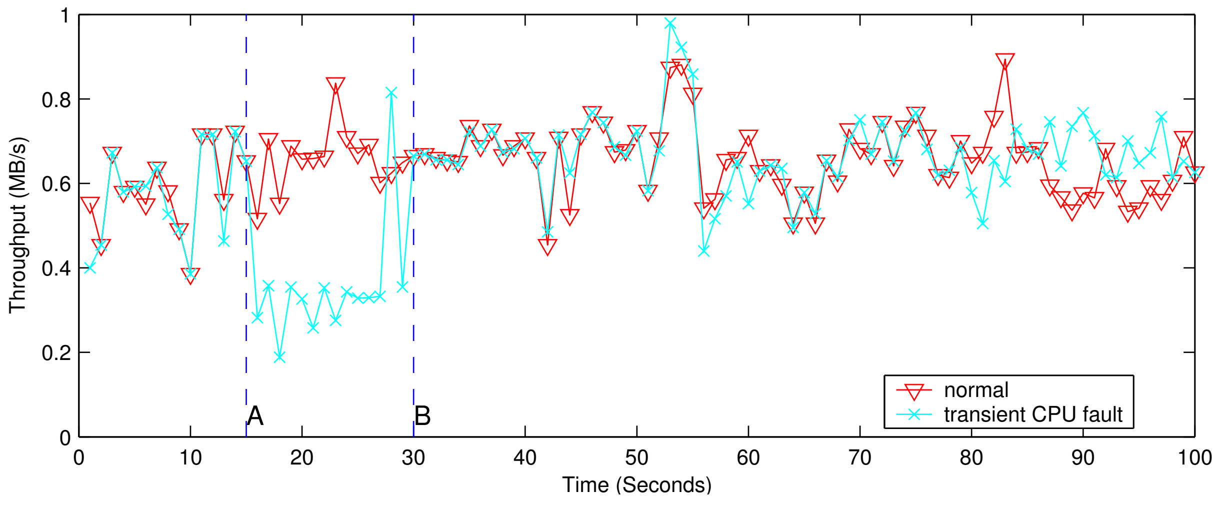

Fig. 6. iSCSI throughput with the EMC trace under transient CPU faults.

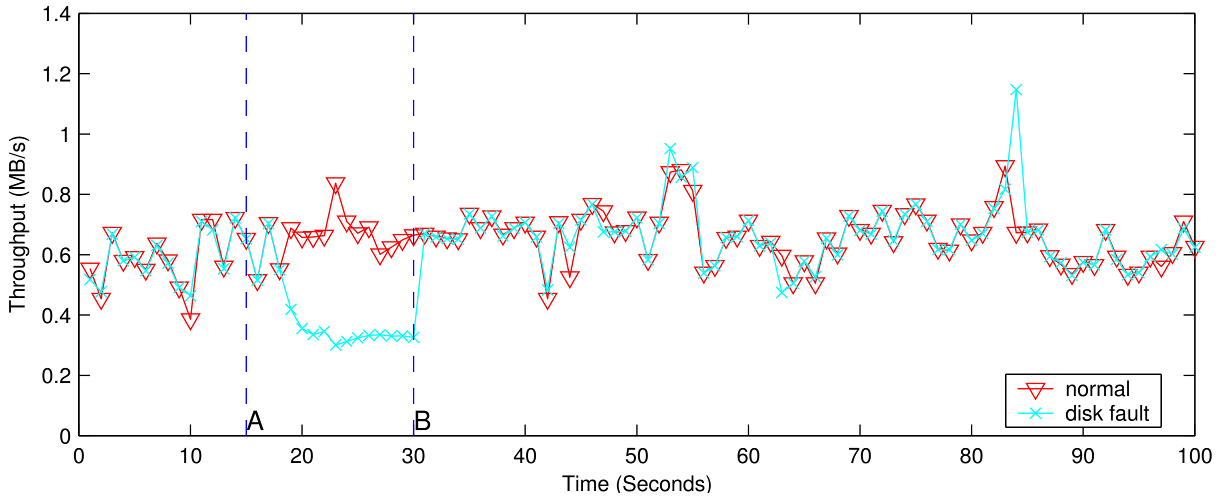

Fig. 7. iSCSI throughput with the EMC trace under disk faults in RAID.

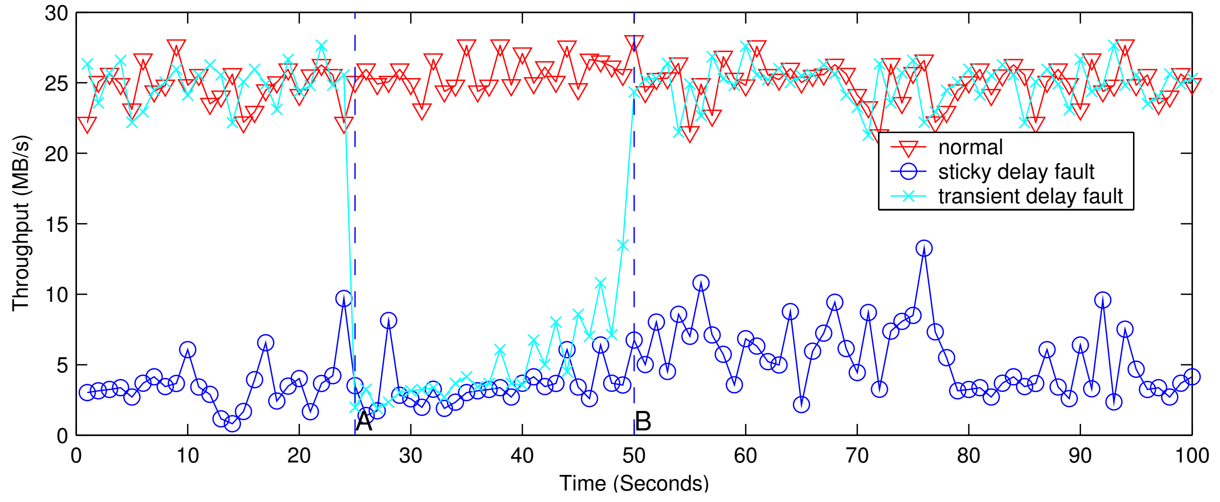

Fig. 8. iSCSI throughput with the TPC-C trace under network delay faults.

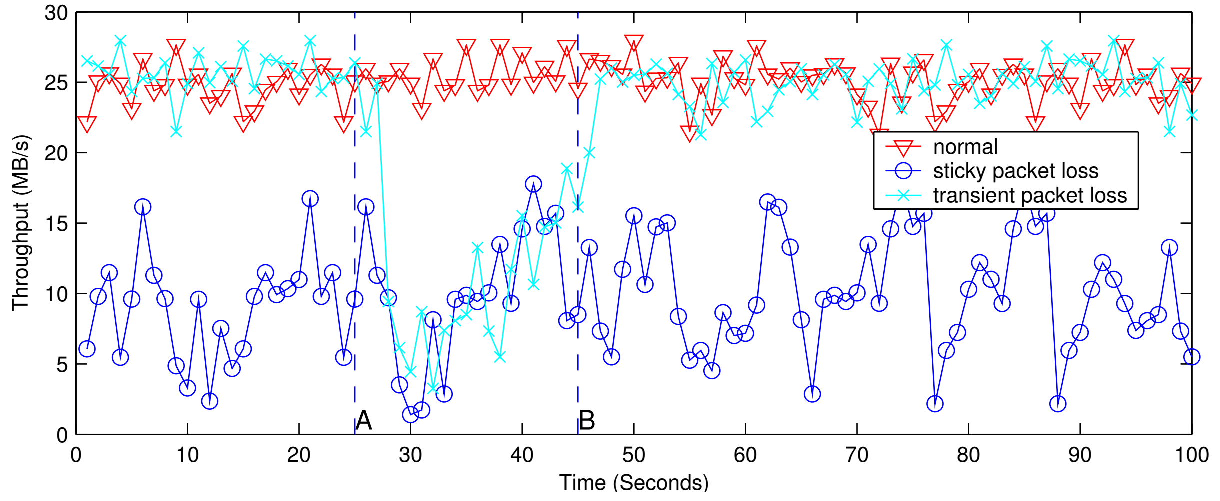

Fig. 9. iSCSI throughput with the TPC-C trace under network packet loss faults.

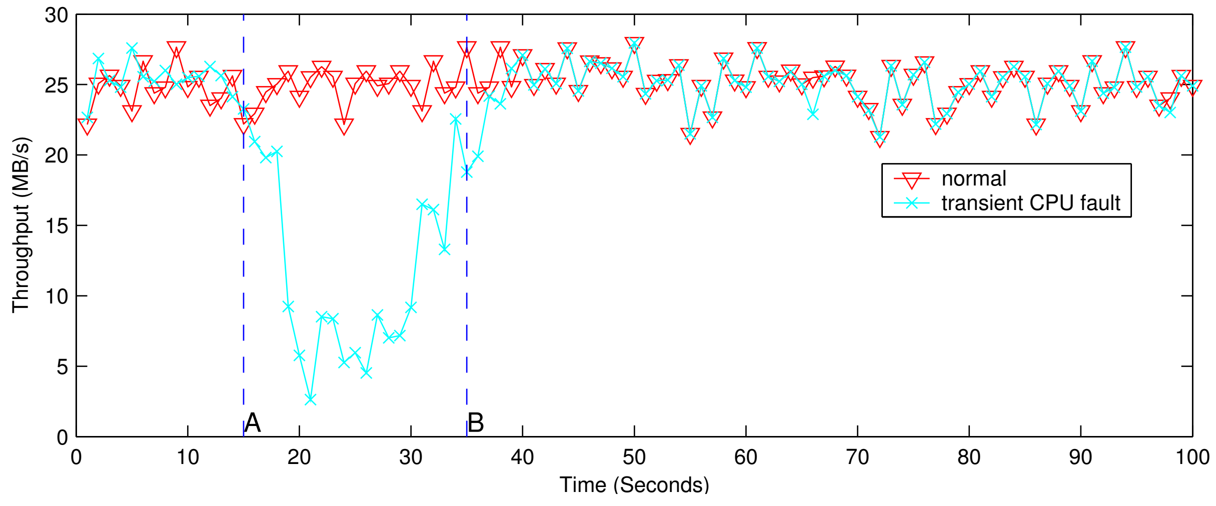

Fig. 10. iSCSI throughput with the TPC-C trace under transient CPU faults.

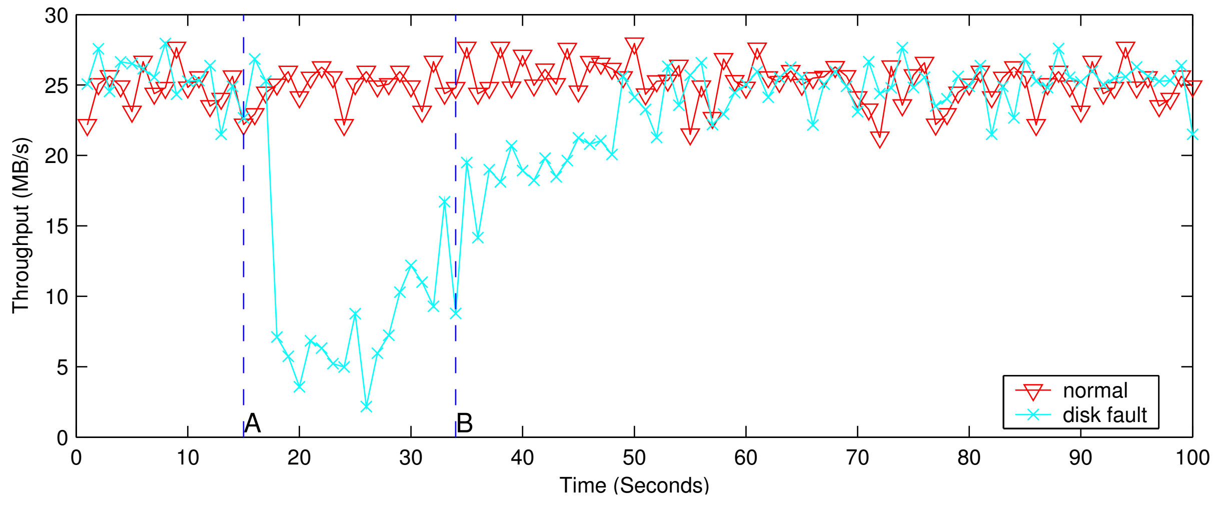

Fig. 11. iSCSI throughput with the TPC-C trace under disk faults in RAID

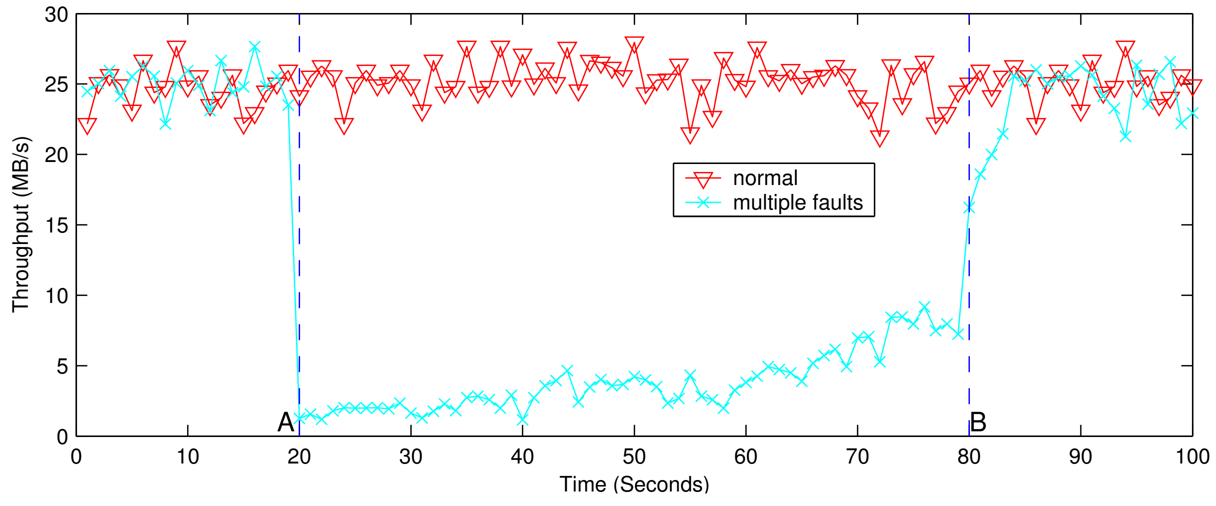

Fig. 12. iSCSI throughput with the TPC-C trace under multiple faults injected. At the 20th second, network delay faults, CPU faults, and disk faults are injected. At the 40th second, the network delay faults are recovered; at the 60th second, the CPU faults are recovered; and the disk faults are recovered at the 80th second.

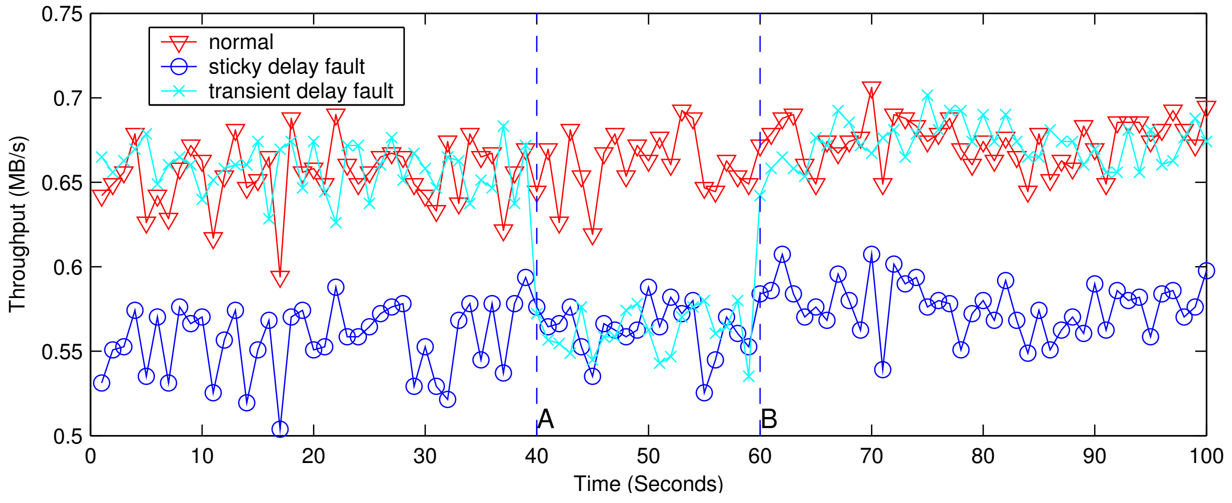

Fig. 13. STICS throughput with the EMC trace under network delay faults.

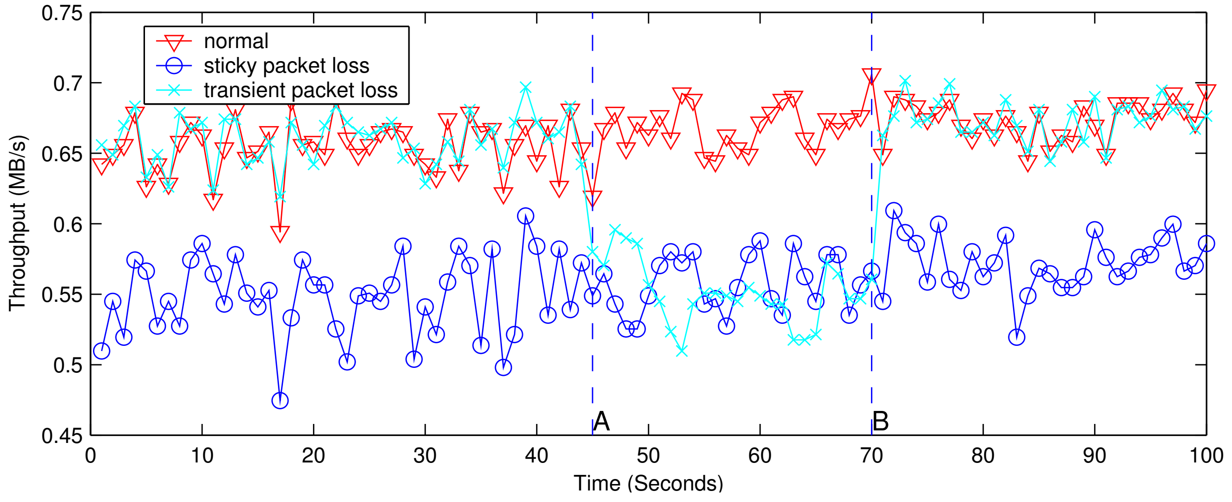

Fig. 14. STICS throughput with the EMC trace under network packet loss faults.

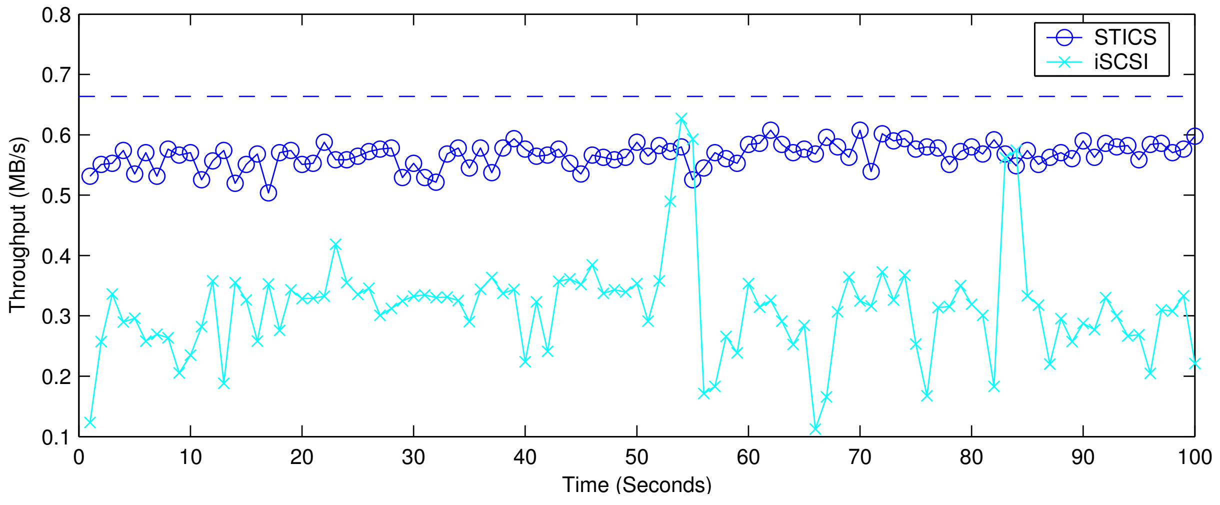

Fig. 15. STICS and iSCSI comparison with the EMC trace under sticky network delay faults.

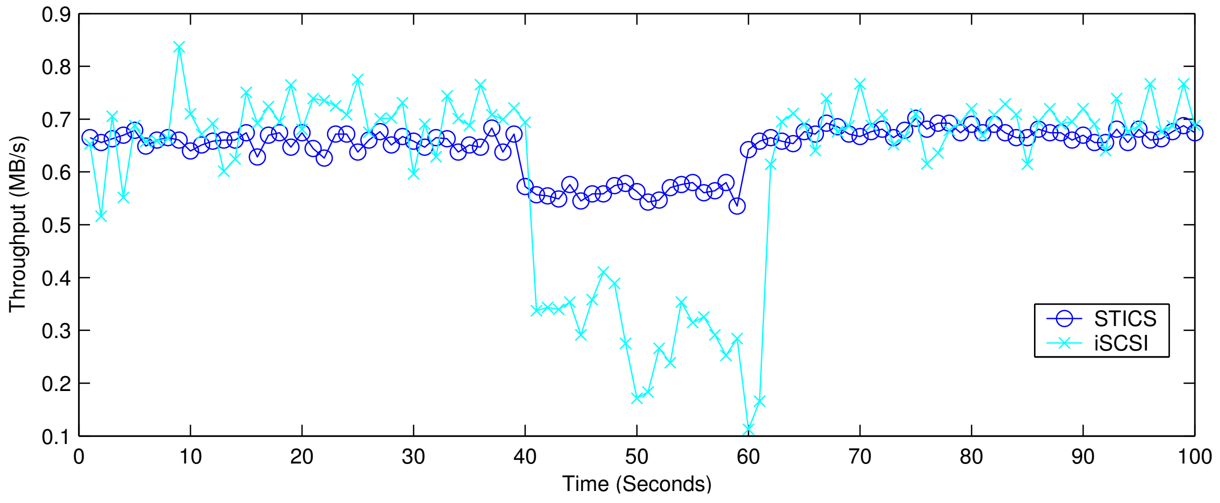

Fig. 16. STICS and iSCSI comparison with the EMC trace under transient network delay faults.

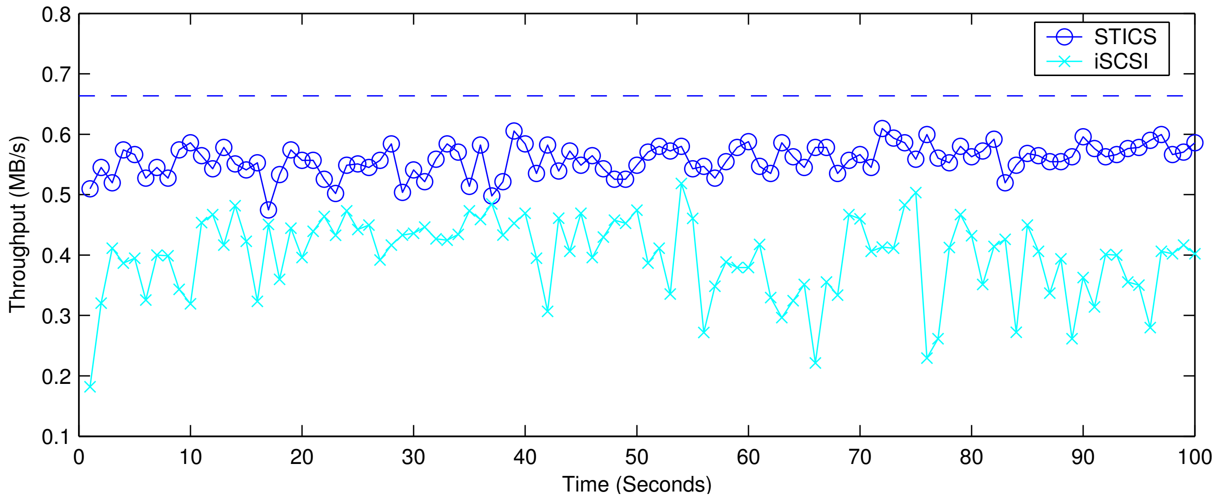

Fig. 17. STICS and iSCSI comparison with the EMC trace under sticky network packet loss faults.

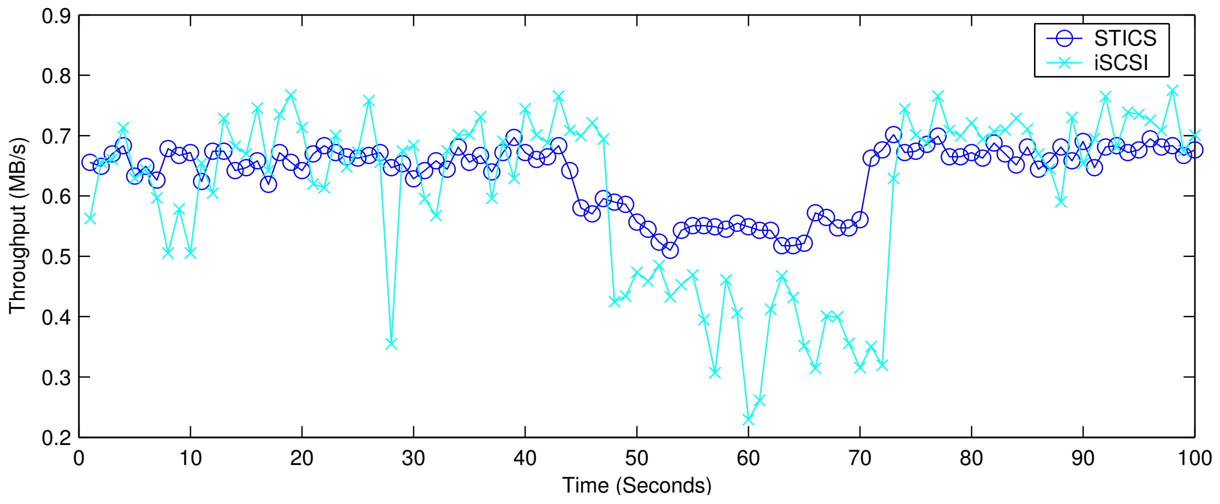

Fig. 18. STICS and iSCSI comparison with the EMC trace under transient network packet loss faults.