SIMLAB - A Simulation Environment for Storage Area Networks *

Petra Berenbrink
University of Warwick
Department of Computer Science
Coventry,CV4 7AL, UK
pebe@dcs.warwick.ac.uk

André Brinkmann
Paderborn University
Department of Electrical Engineering
33102 Paderborn, Germany
brinkman@hni.upb.de

Christian Scheideler
The Johns Hopkins University
Department of Computer Science
Baltimore, MD 21218-2682, USA
scheideler@cs.jhu.edu

Abstract

In this paper, we present a simulation environment for storage area networks called SIMLAB. SIMLAB *is a part of the PRESTO project, which is a joint project of the Electrical Engineering Department and the Computer Science Department of the Paderborn University. The aim of the PRESTO project is to construct a scalable and resource-efficient storage network that can support the real-time delivery of data.* SIMLAB *has been implemented to aid the development and verification of distributed algorithms for this storage network. However, it has been designed in such a way that it can also be used for the simulation of many other types of networking problems.* SIMLAB *is based on C++ and common libraries and input/output formats, which ensures that* SIMLAB *can be used on many different platforms. We therefore expect* SIMLAB *to be useful also for other people working on similar problems.*

1 Introduction

In the last couple of years, a dramatic increase in the need of storing huge amounts of data can be observed. The introduction of enterprise resource planning, on-line transaction processing, e-business, data warehousing, and especially the increasingly media-rich content found in intranets and the Internet are heavily contributing to the data load. This is overwhelming traditional storage architectures. A common approach to handle this huge amount of data is to combine storage devices into a dedicated network, called *storage area network (SAN)*, that is connected to several LANs and/or servers. Major requirements for storage networks are scalability, reliability, availability and serviceability.

In the PRESTO (**P**aderborn **re**al-time **sto**rage network) project, a joint project of the Electrical Engineering and Computer Science Department of the Paderborn University, we focus on the development of such a storage network. Our storage network is intended to be based on a dedicated network formed by a single type of hardware component, called *active router*. In contrast to standard switching elements in SANs active routers incorporate a microprocessor core, which enables an active behavior and the transfer of functionality from the server to the network. Our active routers can be connected to several LANs and/or servers (see Fig. 1.a). An example of such an architecture is given in Fig. 2. From the view of the outside world, our storage network behaves like a single and very large disk by simply providing a huge *virtual data space* (see Fig. 1.b).

In order to ensure a high scalability and availability of our concept, we have to identify efficient strategies for the data management, the communication between the active routers, and for the scheduling of requests. Of course, all the strategies have to be completely distributed, resource-efficient, and self-reconfigurable. Many of our algorithms have been analyzed theoretically in simplified and clean models. The aim of our simulation tool SIMLAB is to fill the gap between our theoretical models and the "real world", i. e. to study the behavior of the developed algorithms in practice. SIMLAB has been implemented to aid the development and verification of distributed algorithms for the

*The research has been done while the authors stayed at the Paderborn University and was supported by the DFG-Sonderforschungsbereich 376, Project C5. The first author is supported by the EPSRC Research Grant GR/M96940 and by the ESPRIT Projects RAND-APX and ALCOM-FT.

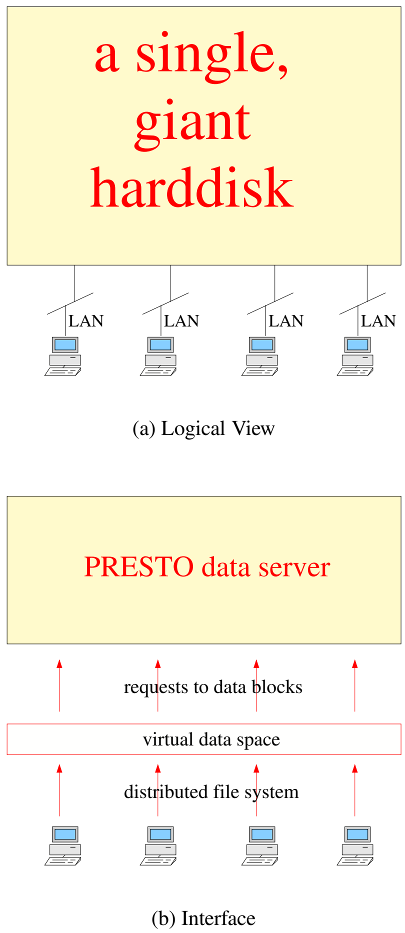

Figure 1. Concept of a virtual parallel disk.

PRESTO storage network and to aid the decision which of the algorithms are best suitable and most efficient in order to be realized in the scope of the PRESTO Project.

SIMLAB works as follows. It requires an ASCII file as input that contains the configuration parameters, and produces an ASCII file as output which stores the simulation results. At the beginning of the simulation, the user can setup an arbitrary network architecture with nodes representing our active routers. As a next step, the set of routers can be specified that have external connections or connections to storage devices. For example, the user can have pre-defined networks like the butterfly network with external connections/ disk connections only at nodes of the highest/lowest level (see Figure 2), or he may choose a network with external connections/ disk connections on every node. Furthermore, the user can specify the type of stored data (films, ...), together with several probability distributions specifying the request behavior of the SAN users. If the physical layout and the data profile is fixed, the user can select between different communication and scheduling strategies. SIMLAB allows to evaluate many different parameters, e. g. waiting time distributions and network/disk utilization. Furthermore, SIMLAB provides scripts and programs that allow the users to efficiently set up a configuration file, and to efficiently evaluate the simulation results that are output by SIMLAB. This includes, for instance, a graphical user interface in order to specify network architectures.

SIMLAB is a sequentially implemented simulation tool and is based on a library of objects that are implemented as classes in C++. It has been written in a way that it ensures a fast incorporation of new algorithms and hardware models. Since the interfaces of SIMLAB are simple ASCII files, our simulator is not limited to any specific computing environment. In its current form, the class library of SIMLAB contains precise models for hard disks, routing nodes, and network interfaces. Each of the models can be adjusted to adapt to new hardware or software constraints. Thus, in addition to using it for the simulation of storage area networks, it is possible to use SIMLAB also for the simulation of standard networks and/or parallel computers. However, in this paper we will restrict ourselves to presenting only those parts of SIMLAB that are necessary for the simulation of algorithms for SANs that are based on a network of active routers.

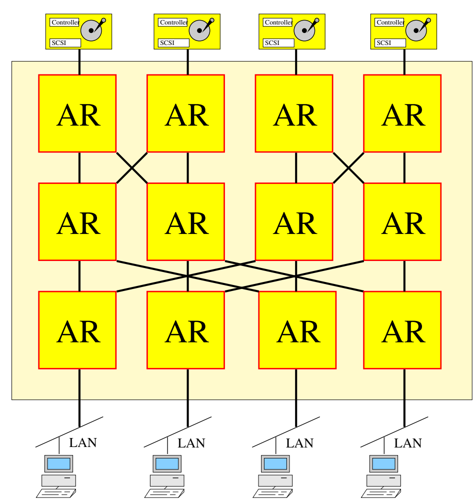

Figure 2. A parallel storage network ("AR" means "active router").

2 Functionality of the SimLab Environment

The SIMLAB environment is based on the assumption that the SAN consists solely of active routers. The SAN has to be able to organize itself in such a way that it is able to fulfill the requests in an efficient and reliable way, no matter what topology the active routers form, or where the external connections/ disk connections are. To achieve this goal, we divided the functionality of the routers into five main units (see Figure 3): the disk unit (short “D”), the user unit (short “U”), the scheduling unit (short “S”), the routing unit (short “R”), and the topology unit (short “T”). Figure 3 shows the logical structure of the functional units and the relationships between them.

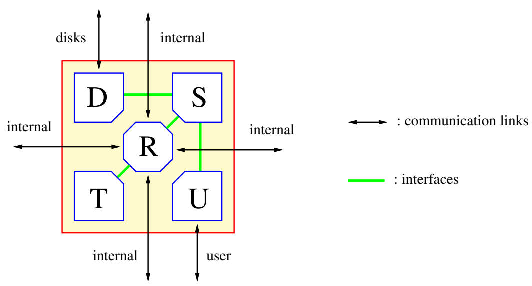

Figure 3. Logical structure of an active router node.

The functional units have the following tasks.

- The **user unit** manages the communication between the outside world (users or servers) and our system. It receives data requests and sends the requested data back to the users. Within the simulation environment, the user unit can also generate data requests by itself.
- The **disk unit** is responsible for managing the storage devices attached to the router. It forwards data requests to the storage devices and receives the requested data from the disks. Furthermore, it is responsible for the *local data placement*, i.e. it determines the mapping of the virtual data blocks to the local disks. Within the simulation environment, the disk unit also simulates the behavior of the connected storage devices.
- The **scheduling** unit determines *when* the data has to be read from *which* storage device. Furthermore, it is responsible for the *global data management*, i.e. the distribution of the virtual data blocks among the disks in the system.
- The **topology unit** determines the paths packets have to take in order to get from some source router to some destination router.
- Finally, the **routing unit** handles the communication between neighbored routers. It determines which packets to prefer if several contend to use the same link at the same time.

In the case of the PRESTO project, these functional units can be used in the following way to process requests (see Figure 3,5). Upon arrival of a new request at the user unit, the user unit forwards the request to the scheduling unit of the same active router. The scheduling unit determines the devices storing the requested data block and the routers connected to these devices. For each of these routers, the scheduling unit generates a request packet and forwards it to the routing unit on its router. With the help of the topology units, these packets will be sent along a sequence of routing units until they reach their destinations. Upon arrival at its destination router, a request packet is forwarded to the disk unit of that router. The disk unit submits a request to the corresponding storage device (or devices) and receives the requested data. This data is sent back to the origin of its request. From there it is forwarded to the user unit that sends it to the user that issued the request.

3 Implementation

SimLab has been implemented in C++. It uses various data structures and graph libraries [15, 11], and different libraries for distributed computation and communication (MPI, PVM)[14, 13].

3.1 The class concept

The class hierarchy is based on a parent class for nodes called *SimNode*. This class provides the basic data structures to build up a parallel interconnection network. This includes input and output queues for packets and other data structures for the information exchange with adjacent nodes like their addresses and the port number of the adjacent link. Furthermore the SimNode class offers a number of useful mechanisms for the simulation itself, like random number generators and functions for the synchronization of the network.

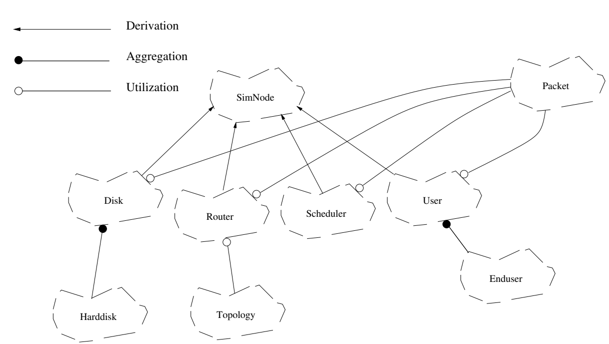

Figure 4. SimLab objects and their relationships.

Derived from this parent class, the disk, router, scheduler, and user class represent the main functional units of an active router node. The class corresponding to the topology unit has an exceptional position in our simulation environment. If it would have been derived from the SimNode class, every router object in the network needed its own topology object. Since the topology object requires a large amount of memory, this would lead to an enormous waste of storage capacity, especially for large networks. Hence, we implemented only one global instance of the topology class which can be used by every instance of the router class. At the beginning of the simulation, the global topology class determines a system of paths. During the simulation, every router can access the topology class in order to request valid paths.

The communication between the disk, router, scheduler, and user unit takes place via data and control packets, derived from the packet class. We do not make any difference between the inter- and intra-node communication of active routers. Therefore, it is possible to relax the model of the active router node and to model the scheduler or disk controller as, e.g. a workstation or PC. To enable the simulation environment to work without an external request generator we added objects that create data requests to the user unit. The object representing the disk unit has been designed as a general purpose local memory management unit. In order to consider the characteristics of hard disks in our simulations, we added objects to the disk unit that simulate the behavior of hard disks. The routing object is designed in a way that the routing rules and hardware characteristics can be exchanged easily. Also the topology and scheduling objects allow for implementing a large variety of different strategies. Since the topology and the scheduling units perform only internal processes, no additional objects representing hardware characteristics have to be added to them.

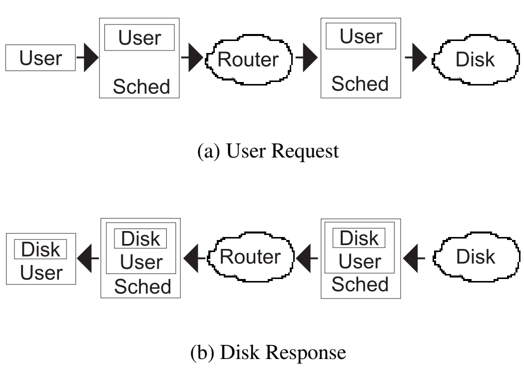

Figure 5. Packet data flow.

Technically, a typical packet flow for a storage area network is processed as follows (see Figure 5). The user poses a request to the respective scheduler. This user request includes a pointer to/or place for a data block, the user address, and the virtual address of the data block. The scheduler processes the request and inserts the user packet into a scheduler packet, which is moved to the router module. In order to enable the active routers to pass the packet to the destination address, they ask the topology unit for valid output ports. At the destination, the router packet is unwrapped and given to the scheduler. The scheduler recognizes the user packet and passes it to the disk controller, where it is unwrapped again. The remaining user packet is filled with the data information and sent back to the requesting user. Other possible situations are that the scheduler requests information about the link load of a router or the queue length of a disk. For this purposes, additional classes are derived from the parent packet class.

3.2 SimLab interfaces

As already mentioned, SIMLAB is embedded into a tool chain, which enables the user to specify network topologies, to set up simulation parameters, and to evaluate the simulation results (see Fig. 6).

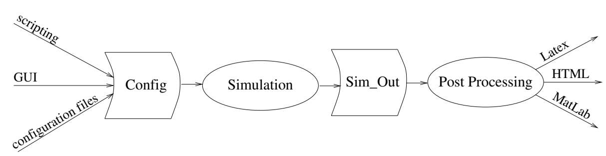

Figure 6. Simulation flow.

SIMLAB can run on every PC/workstation which includes a C/C++ compiler and the Leda library. Access to the simulation parameters is given through different configuration files. Each of the objects of the files can be configured in different ways. To have an easy entry into the simulation environment, we offer access to the SIMLAB-parameters via a graphical user interface (GUI). E.g., it is possible to specify the desired network infrastructure with the help of a graph editor and to translate its output into a valid net-list format. Furthermore it is possible to set up the parameters of the node objects via a Java script. This script enables the user to easily manipulate the parameters of the environment, to start the simulation, and to evaluate the simulation results. For users that would like to use all capabilities of the simulation environment, we suggest a script-based access to SIMLAB. A demo version of SIMLAB and the configuration tools can be accessed and executed via the Internet. This version is limited to the simulation of small networks for a limited amount of time.

SimLab produces two kinds of textual output. We implemented a trace mode for single step evaluation. At each time step, every active router writes its internal statistics, like queue lengths and packet transmissions, into an output file. This enables the algorithm designer to prove the correct implementation of his algorithm and to gain further insights into his algorithm. The output files can be linked with a visualization tool, which documents the packet flow in the network. Of course, this mode results in a huge memory consumption, which limits this approach to small networks or/and a small number of simulation steps. Therefore, we developed a second mode where each object keeps statistics about its most important parameters like maximum or average utilization together with the corresponding standard deviations. The respective output files can be connect with AWK-scripts in order to extract a subset of the generated data. The AWK-scripts offer interfaces to Matlab and Latex for the graphical and/or tabular representation of the simulation results.

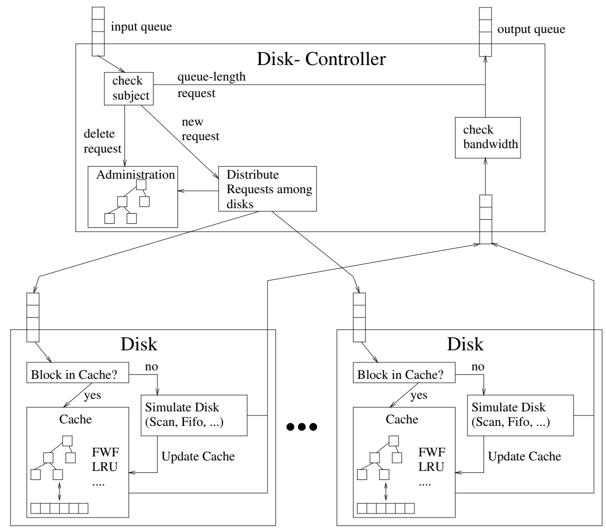

Figure 7. Implementation of the disk-controller.

4 Algorithms

In this section we describe the algorithms we have implemented in the SimLab environment for the scheduler unit, disk unit, topology unit, routing unit, and the user unit. Note that these algorithms can be regarded as exemplary implementations. The simulation environment can easily be extended by other strategies. Furthermore, note that the user can choose any one algorithm from each unit in order to make a simulation. Hence, the algorithms can be mixed arbitrarily. The selections can be done, for instance, via the WWW user interface mentioned in Section 3.

4.1 Disk unit

The implementation of the disk unit is divided into two sections: a model of the disk controller and a model of the disks themselves (see Figure 7). This ensures that arbitrary local load balancing strategies like the RAID strategies can be mixed with different kinds of disk models. We assume that the disks are connected to the controller via a SCSI bus. Our bus model only simulates the bandwidth limitations of the SCSI bus. The protocol itself is not incorporated.

The disk controller receives three different kinds of requests from the corresponding scheduler: queue length requests, delete requests, and data requests. The requests are buffered in an input queue and processed in FIFO order. The queue length request returns the actual input buffer length of the disk controller. The delete request can be used to erase formerly given orders from the disk controller. These kinds of requests can be used to implement global load balancing strategies to distribute requests evenly among the disks. They can be immediately processed by the controller and do not require any disk access.

Data requests to a disk controller require a distribution strategy among the connected disks. In the case of the standard strategy, every data block is stored only on one disk and every request is submitted directly to this disk. Furthermore, we have implemented different RAID strategies like RAID 0 (Striping), RAID 1 (Mirroring) and RAID 5 (Striping with Parity). The parity block in RAID 5 can be used for an additional level of load balancing [9, 6]. In this case, only n of $n+1$ disks are required to process a block request. The remaining disk can be used for another request. In this case it is important to notice that the use of the parity block requires extra work to rebuild the data block, which has to be done in the disk controller itself or can be transferred to the requesting server.

The disks: In order to simulate disk behavior we distinguish between seek time (the time which is used in order to find the requested block on the disk) and read time (the time used to read the data from the disk) [18]. To calculate the seek time, we use a linear model which depends on the actual position of the disk head and the time the head requires for a full-stroke. To determine the order in which a disk fulfills its requests we have implemented the FIFO and the SCAN strategy. Additionally, we provide a model of a RAM disk which is able to process any request in a single time step. This can be especially useful to compare different higher level strategies without the influence of the real hardware limitations. Furthermore, we have implemented several standard caching strategies, such as FWF (flush when full) and LRU (last recently used).

4.2 User unit

The user unit does not only serve as an interface (to the users), it additionally simulates the behavior of SAN users. We assume a fixed number of users in every user unit that request the data following a fixed rate, i.e. the request injection rate is fixed and does not vary over the time. In order to determine the requested data items (e. g. the film that is chosen by the user) we use binomial distribution, zipf distribution, or the normal distribution. Due to this distribution, the user chooses a file and accesses sequentially the data blocks stored in the file, starting from the first block, or from a randomly chosen block of the file. The file length can be fixed in advance or also be randomly chosen. This approach is able to cover different possible scenarios. For example, to study tele-teaching applications, we can chose a request distribution where every user accesses the same block from the same file at a very similar time-step. In order to study Video-on-Demand servers, we assume a zipf distribution with randomly chosen access points in the files. This ensures that a constant number of users in the system can be simulated without overestimating the influence of the caching strategy.

4.3 Scheduling unit

As mentioned earlier, this unit is responsible for the placement of the data and for the scheduling of the requests. We assume that the virtual data space consists of data blocks of uniform size.

Data placement. We realized, for instance, the k of $k+1$-strategy where every data block is divided into k sub-blocks. The $k+1$th sub-block stores the parity function of the first k sub-blocks. Each of the $k+1$ sub-blocks is then allocated to a different node in the storage network. Note that for $k = 1$ the strategy simply stores every data block as a whole on two disks. Furthermore, note that it is sufficient for a user to get arbitrary k out of the $k + 1$ sub-blocks. In the case that the parity block is sent to the user it is easy to construct the contents of the missing sub-block.

Obviously, the strategy with $k = 1$ is superior if the virtual data blocks are relatively small (see Figure 8). In such a case it will not be profitable to divide a block because this will result in long seek times at the disks. With increasing block size, the better load balancing capabilities of the strategies with a greater number of sub-blocks outperform the disadvantage of the smaller sub-block size. The storage overhead caused by the k of $k + 1$-strategy is $(100/k)\%$.

In order to distribute the virtual data space among the disks of the system we developed a special, randomized hash function [10]. The virtual data space can be far greater than the capacity of the system. However, only the used

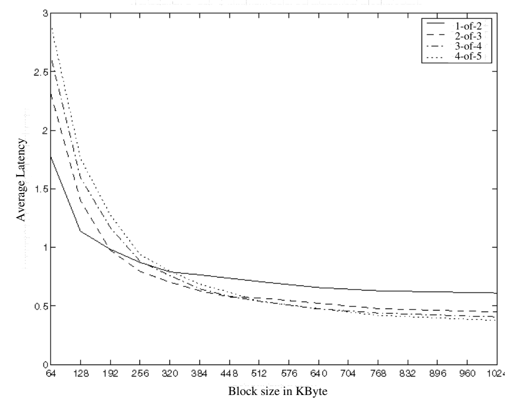

Figure 8. Relationship between block size, data placement, and latency.

data blocks will actually be assigned to physical blocks on the disks. The developed hash function is very flexible and allows to add and to remove some disks with a minimum number of virtual blocks that have to be replaced.

Scheduling. We have implemented several strategies which are motivated by so-called balls-into-bins games (see [5], [17], [8]). These strategies are able to exploit the redundancy generated by our placement strategies.

- Simple minimum game: The requests are fulfilled from the k of the least loaded $k + 1$ disks storing the requested data. The scheduler sends a *load request* to each of the $k + 1$ disks holding a copy of the requested virtual data block. Each disk receiving such a load request answers with its *request queue length* (the number of jobs which already wait at the disk in order to be fulfilled). In turn, the scheduler sends *data requests* to the k disks with the minimum queue length. These data requests will be fulfilled by the disks.

- Minimum game with pre-placement [1]. If the request queue of a disk is empty and the active router controlling this disk receives a load request, then it does not respond with the queue length, but instead immediately asks the disk to provide the requested data block and sends the data back to the user. If the the request queue of a disk is not empty, the disk responds with the request queue length, as in the case of the simple minimum game. In the case that k' disks respond with directly sending the requested data, the scheduler sends data requests to the $k - k'$ disks with minimum queue length.

  This strategy has the advantage that disks are not idle even though there is a data request which can be fulfilled by it. A disadvantage is that some requests may be processed by more disks than necessary to reconstruct the requested data block.

- The next strategy is similar to the minimum game, but in addition to the request queue length, the scheduler takes the link congestion of the interconnection network into account. More precisely, for each involved disk the scheduler computes a function depending on the maximum link congestion (number of packets that have to be routed over a link) of the links on the path to the disk and the request queue length of the disk. Then the request is fulfilled from the k disks having the minimum function value.

4.4 Routing unit

We have implemented four different packet switching schemes.

- FIFO (first-in-first-out). The routers forward the packets in the order of their arrivals.
- LIS (longest-in-system). The routers always forward the packet with the earliest generation time.
- Growing rank protocol (see [16]). In this case every packet is marked with a rank depending on its generation time. The routers always forward the packet with the smallest rank. Furthermore, every time a packet traverses a link, its rank will be increased.
- ELIS (enforced longest in system, see [19]). This strategy is a restricted variant of the LIS strategy. The strategy uses control packets in order to enforce that the generation times of the packets which are routed across any link are increasing with time.

Figure 9 shows a sample output of simulations with different switching strategies for a butterfly network of dimension 4 and different injection rates. It can be seen that the ELIS strategy is inferior to the other two strategies in terms of average latency in this kind of network topology. This is due to the fact that ELIS requires some control packets, which decrease the total amount of available bandwidth for the data packets.

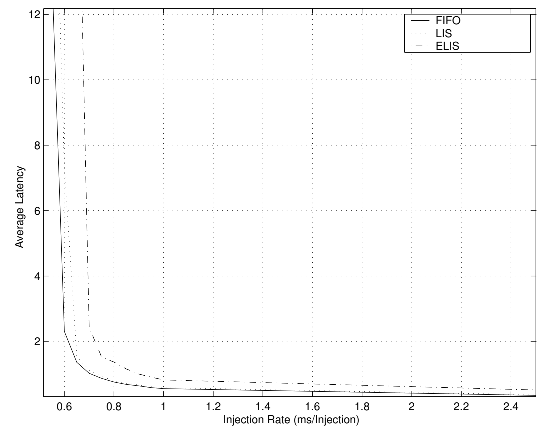

Figure 9. Relationship between switching scheme and average latency in a butterfly network of dimension 4.

4.5 Topology unit

The task of the topology unit is to initialize the server and to determine path systems which distribute the routing packets as evenly as possible over the links of the whole network. Our topology unit is able to generate many standard networks like mesh networks, butterflies, hypercubes, and DeBrujin networks. But it also offers an interface to build own networks or random networks. Furthermore, the user can distribute external user interfaces and storage devices arbitrarily over the active routers of the networks. For example, in the case of the two dimensional mesh one can chose between (1) every router of the mesh is connected to an external interface and a disk interface, (2) only the routers of the first row are connected to an external interface, and only the routers of the column are connected to disk interfaces.

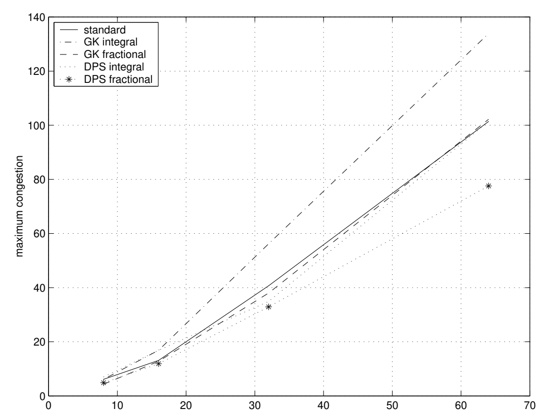

Figure 10. Max. congestion of router links.

We have implemented several algorithms to determine path systems. Some of them can only be used for fixed networks, like the simple X-Y path system that can only be used for meshes. For the case of arbitrary networks we have implemented three general strategies. In addition to a standard shortest path algorithm, we implemented the algorithm by Garg and Könemann ([12]). This algorithm offers an interesting new approach in order to find good approximate solutions to multi-commodity flow problems solely through an iterative use of shortest path calculations. The basic approach of our *DPS* algorithm [7] is based on approximation algorithms for the multi-commodity flow problem [3, 4] and on a routing algorithm presented in [2]. The algorithm finds paths for each source/destination pair so that the capacity constraints on the links in the network are kept and, therefore, requests and data streams do not overload the network. It is based on a simple load balancing approach, called *diffusion*. Figure 10 shows the influence of the path selection strategy onto the congestion of the router links for DeBruijn networks of different degree.

5 Conclusions

In this paper we presented the sequential version of the simulation environment SIMLAB for storage area networks. SIMLAB has been developed to aid the development and verification of distributed algorithms for storage area networks and is based on the assumption, that the SAN consists of active routers. Due to the programmability of these active router nodes, the designer of storage area networks has the possibility to transfer functionality from the connected servers into the SAN.

In addition to the sequential version of SIMLAB we have implemented a first distributed version which is based on PVM. Besides the possible speed-up one major advantage is the much larger amount of available memory. This enables the algorithm designer to instantiate an instance of the topology unit in every active router and to study dynamic properties of the used path selection algorithms (i.e., their behavior when the network topology changes).

Acknowledgments

We thank Ralf Hunstock for helpful and stimulating discussions. Also, we would like to thank the members of the software project *PGMUDAS* which has been held at the Paderborn University during the years 1999/2000. The members of that group implemented the simulation environment SIMLAB for our PRESTO server and performed the simulations.

References

[1] M. Adler, P. Berenbrink, and K. Schröder. Analyzing an infinite parallel job allocation process. In *Proc. European Symp. on Algorithms (ESA'98)*, pages 417–428, 1998.

[2] W. Aiello, E. Kushilevitz, R. Ostrovsky, and A. Rosén. Adaptiv packet routing for bursty adversarial traffic. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 359–368, 1998.

[3] B. Awerbuch and F. Leighton. A simple local-control approximation algorithm for multicommodity flow. In *Proc. of the 34th Annual Symposium on Foundations of Computer Science*, pages 459–468, 1993.

[4] B. Awerbuch and F. Leighton. Improved approximation algorithms for the multi-commodity flow problem and local competitive routing in dynamic networks. In *Proc. of the 26th Annual ACM Symposium on Theory of Computing*, pages 487–496, 1994.

[5] Y. Azar, A. Broder, A. Karlin, and E. Upfal. Balanced allocation. In *Proceedings of the 26th Symposium on Theory of Computing (Stoc)*, pages 593–602, 1994.

[6] P. Berenbrink, A. Brinkmann, and C. Scheideler. Design of the PRESTO multimedia data storage network. In *Proceedings of the Workshop on Communication and Data Management in Large Networks (INFORMATIK 99)*, Oct. 1999.

[7] P. Berenbrink, A. Brinkmann, and C. Scheideler. Distributed path selection for storage networks. In *Proceedings of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2000), Las Vegas, USA*, June 2000.

[8] P. Berenbrink, F. Meyer auf der Heide, and K. Schröder. Allocating weighted jobs in parallel. *In Journal of Theory of Computing Systems*, 32:281–300, 1998.

[9] Y. Birk. Random RAIDs with selective exploitation for high performance video servers. In *Proceedings NOSSDAV '97*, May 1997.

[10] A. Brinkmann, K. Salzwedel, and C. Scheideler. Efficient, distributed data placement for storage area networks. In *Proceedings of the 12th ACM Symposium on Parallel Algorithms and Architectures (SPAA'2000)*, 2000.

[11] P. Davids. Atlas - analysis tool for local area network simulation version 5. Technical report, Lehrstuhl für Informatik IV, RWTH Aachen, 1993.

[12] N. Garg and J. Knemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS '98)*, Nov. 1998.

[13] A. Geist. Pvm 3 user's guide and reference manual. Technical report, Oak Ridge Laboratory, 1994.

[14] W. Gropp, E. Lusk, and A. Skjellum. *MPI, Portable Parallel Programming with the Message-Passing Interface*. The MIT Press, 1994.

[15] K. Mehlhorn and S. Näher. *LEDA, A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[16] F. Meyer auf der Heide and B. Vöcking. A packet routing protocol for arbitrary networks. In *Proceedings of the 12th Symposium on Theoretical Aspects of Computer Science*, pages 291–302, 1995.

[17] M. Mitzenmacher. Density dependent jump markov processes and applications to load balancing. In *Proceedings of the 37th Symposium on Foundations of Computer Science (FOCS'96)*, pages 213–222, 1996.

[18] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *Computer*, 27(3):17–28, Mar. 1994.

[19] C. Scheideler and B. Vöcking. From static to dynamic routing: Efficient transformations of store-and-forward protocols. In *Proceedings of the 31st Symposium on Theory of Computing (STOC)*, pages 215–224, 1999.