Distributed Path Selection for Storage Networks*

Petra Berenbrink
Department of Computer Science
Paderborn University
33095 Paderborn, Germany

André Brinkmann
Department of Electrical Engineering
Paderborn University
33095 Paderborn, Germany

Christian Scheideler
Department of Computer Science
Paderborn University
33095 Paderborn, Germany

Abstract *In the last couple of years a dramatic growth of data storage capacity can be observed. To manage the explosion of data, a common approach is to combine storage devices into a dedicated network, called storage area network. One of the major requirements for these networks is scalability. Existing concepts often lack scalability either because they are based on central components, or because the routing cannot handle large or irregular topologies efficiently.*

In a project at the Paderborn University, we are currently developing concepts for storage area networks ensuring that every function is performed in a completely distributed way. In this paper, we concentrate on our routing concepts. We present a completely distributed path selection algorithm called DPS *that is applicable to* arbitrary *network topologies. Surprisingly, this algorithm does not need any information about the topology of the network and, although very simple, is able to compute paths that provably reach a best possible distribution of paths among the links. To demonstrate the applicability of our algorithm, we compare its performance with several other approaches for standard networks such as meshes and butterflies.*

Keywords: parallel and distributed algorithms, distributed storage networks, path selection strategies

1 Introduction

In the last couple of years a dramatic growth of enterprise data storage capacity can be observed. A common approach is to combine storage devices into a dedicated network, called *storage area network*, that is connected to several LANs and/or servers. One of the major requirements for these networks is scalability. In order to ensure a high scalability it is vital to avoid central instances of any kind, since they cause bottlenecks in the system. Thus, distributed strategies are sought that can handle all aspects of a storage area network.

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 aim to develop intelligent routing switches, called *active routers*, that can be used to manage storage area networks in a completely distributed way. Each active router can be connected to disks via a SCSI interface and to the outside world via a fast Ethernet interface. Furthermore, the router has four dedicated links which can be used to construct a network of active routers of arbitrary size and topology (for an example, see Fig. 1).

We assume that for each Ethernet port of the network connected to the outside world data requests arrive with a fixed maximum injection rate. Each of these requests has to be forwarded to the router that is connected to the storage device storing the requested data item. In the case of read requests, the requested data has to be sent back to the routers connected to

*Research supported by the DFG-Sonderforschungsbereich 376

the corresponding Ethernet ports. Obviously, the *data layout*, i. e. the distribution of the data items among the storage devices of the network, determines the communication pattern in the network. In the PRESTO project, we choose the common approach [1, 2, 3, 4] of distributing the data items randomly among the storage devices. The great advantage of random placement strategies is that they ensure a *statistically fixed communication pattern* which allows us to assign fixed bandwidth demands to each source/destination pair of active routers.

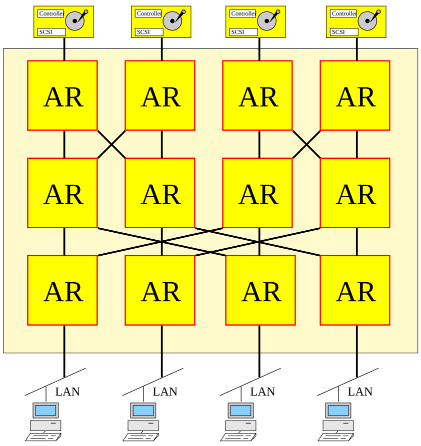


Figure 1: A parallel storage network ("AR" means "active router").

In this paper, we concentrate on routing concepts for storage area networks. Routing involves two basic activities: a *path selection* strategy, and a strategy for sending the information units (usually called packets) along the selected paths. In this paper, we focus on the first part, the path selection. We present a fully distributed path selection algorithm, called *Diffusive Path Selection* (or *DPS*) that exploits the fact that we have a statistically fixed communication pattern. 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. In particular, *DPS* has the property that for *any* set of bandwidth demands for which there exists a path system that keeps the constraints on the link capacities, *DPS* finds such paths. This means that if the bandwidth demands are *maximal* (i. e., any increase of a bandwidth demand results in overloaded links), then *DPS* will converge towards a *best possible* path system. Previous path selection algorithms such as *RIP* and *OSPF* do not ensure this property in general. Like *RIP*, *DPS* works in a completely distributed way and does not require any knowledge about the topology. It applies a simple local load balancing method to find fractional flows through the network for the bandwidth demands.

We believe that our path selection strategy is applicable to many more networking scenarios than just storage networks. Basically, the only condition that has to be fulfilled is that the bandwidth demands are quasi-stable (in a sense that they only change slowly).

1.1 Specification of the problem

Suppose we are given a network $N = (V, E)$ of active routers which connects a set of sources $S = \{s_1, \dots, s_n\}$ to a set of destinations $T = \{t_1, \dots, t_m\}$, $S, T \subseteq V$. We do not require that S and T are disjoint (that is, a node can be both source and destination). The *link capacities* are determined by the function $C : E \rightarrow \mathbb{R}^+$. Furthermore, the *demands* are given by a matrix $D = (d_{i,j})$, $1 \leq i \leq n$ and $1 \leq j \leq m$. $d_{i,j}$ represents the bandwidth demand of the source/destination pair (s_i, t_j). The goal is to find a path system which complies with the edge capacities and which is able to fulfill the bandwidth demands of all source/destination pairs.

In general, solutions to this problem are classified into *fractional* and *integral* ones. In the former case there can be more than one path for a single source/destination pair, i. e. its bandwidth is split over several paths. In the latter case only one path is allowed for each source/destination pair. A standard technique to convert fractional paths into integral paths is to use *randomized rounding* [5].

1.2 Previous results

The problem defined in the previous section is related to multicommodity flow problems. Problems of this kind can be solved through linear programming in polynomial time (see e.g. [6]). There are also a number of fast approximation algorithms [7, 8, 9, 10] for the multicommodity flow problem. In [11] Garg and Könemann present an interesting new approach that finds good approximate solutions to multicommodity flow problems solely through an iterative use of shortest path calculations. We decided to compare this algorithm with our algorithm. However, we note that in contrast to *DPS* the Garg-Könemann algorithm, like *OSPF*, is not a distributed algorithm. In order to use it in a network, every processor has to know the entire topology of the network.

The basic approach of our *DPS* algorithm is based on approximation algorithms for the multicommodity flow problem [9, 10] and on a routing algorithm presented in [12]. In [9] Awerbuch and Leighton present a distributed approximation algorithm for the multicommodity flow problem that runs in a time that is polynomial in the number of nodes of the network. The algorithm is based on a simple load balancing approach, called *diffusion*: in each round, every processor attempts to balance its load with all of its direct neighbors. In [10], the same authors present a related algorithm that has a better running time and that even works in networks where edge capacities can vary in an unpredictable and unknown fashion. In [12], Aiello et al. use a similar diffusion approach in order to design a routing protocol that "discovers" routes which avoid "traffic jams". They assume an adversarial injection model. Their protocol is very simple, distributed, and deterministic and applies to any network topology. Furthermore, it guarantees that for any injection sequence generated by the adversary the number of the packets in the system is bounded. However, the drawback of this protocol is that packets may experience a delay that can be polynomial in the number of nodes in the network.

The difference between these results and our algorithm is that Awerbuch and Leighton use different, more complicated strategies to balance the load and Aiello et al. do not use their diffusion method for the design of path systems, but directly for the routing of packets.

The diffusion approach is well known in areas such as physics and load balancing in networks. For example, in the area of load balancing Cybenko and Boillat [13, 14] were the first to study a simple diffusive load balancing strategy. In [15] Diekmann, Frommer, and Monien design a general mathematical framework to analyze the properties of nearest neighbor balancing algorithms using the diffusion approach.

2 The DPS Algorithm

In this section, we present the distributed path selection algorithm *DPS*. We will restrict our attention to a uniform link bandwidth of $C(e) = k$ for every link e, i. e. in every round, at most k packets can cross any edge in both directions. However, our algorithm can be easily extended to the non-uniform case. We assume that we have n sources and m destinations and that packets can be split into arbitrarily small parts.

In order to compute a path system, *DPS* simulates a fractional flow of data through the network. This flow will be used to obtain a fractional path system. The fractional path system is represented by local routing tables which are stored on every node of the network. In contrast to standard routing tables, the table of node i stores a *weight* $w^e_{i,j}$ for every incident edge e and every destination j. The weight can be regarded as the fraction of packets reaching i with destination j that have to be sent across edge e.

DPS works in rounds. During every round, each node i has to perform the following actions. According to the bandwidth demands, it injects $d_{i,j}$ packets for each destination j. Then, node i calculates $u_{i,j}$, the number of packets stored in it with destination j, and distributes these packets evenly among its outgoing edges, resulting in $p_{i,j}$ packets per edge. Furthermore, it computes for every incident edge $e = (i, w)$ the difference U_j^e between $p_{i,j}$ and $p_{w,j}$. This will determine the number of packets to be sent from i to w. After sending $\bar{\ell}_{i,j}^e$ packets with destination j along link e, node i updates the value of the weight $w_{i,j}^e$ for each incident edge and each destination j, and the next round starts.

Let $\ell_{i,j}$ (resp. $\ell_{i,j}^e$) count the total number of packets with destination j that were sent out of node i (resp. along link e) during all rounds. In the following, we present a detailed description of one round of *DPS* at node i.

1. Inject $d_{i,j}$ packets for every destination j.
2. For each destination j compute $u_{i,j}$ and $p_{i,j} := u_{i,j}/(\text{degree of node } i)$
3. For each outgoing link $e = (i, w)$
   (a) Exchange the variables $p_{i,j}$ with the adjacent neighbor w.
   (b) Compute the total potential difference
   $$U_i^e := \sum_{\substack{j \in T \\ p_{i,j} > p_{w,j}}} p_{i,j} - p_{w,j} \ .$$
   (c) For all destinations j, if $p_{i,j} > p_{w,j}$
      - send
      $$\bar{\ell}_{i,j}^e := \frac{p_{i,j} - p_{w,j}}{2} \cdot \min\left[1, \frac{k}{U_i^e}\right]$$
      of the packets destined for destination j across e
      - $\ell_{i,j}^e := \ell_{i,j}^e + \bar{\ell}_{i,j}^e$
      - $\ell_{i,j} := \ell_{i,j} + \bar{\ell}_{i,j}^e$
      - $w_{i,j}^e := \ell_{i,j}^e / \ell_{i,j}$
4. Receive all incoming packets and remove any packets that have reached their destination.

The algorithm terminates if the change of $w_{i,j}^e$ is smaller than some constant ϵ for every node. In order to convert this path system into an integral one we use the technique of randomized rounding.

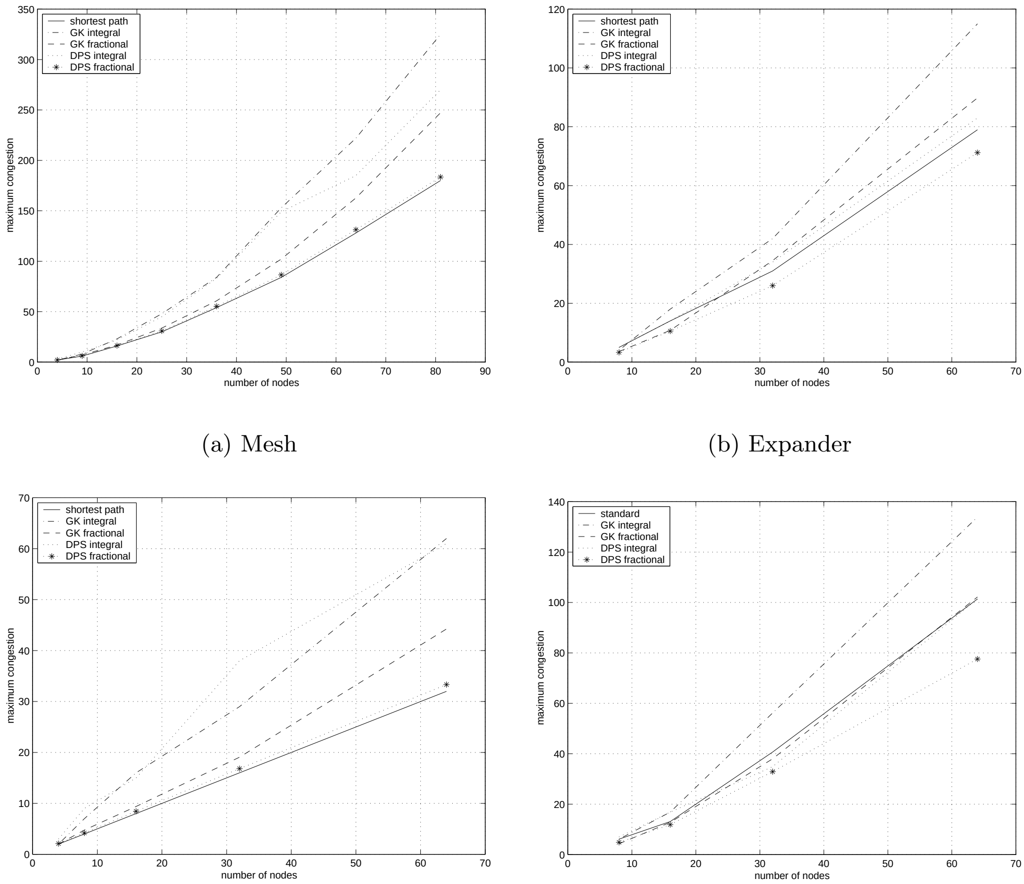
There are two important aspects one has to consider. First of all, the most expensive part of *DPS* is Step 3(a), since it involves exchanging a large amount of data between neighbors. We note, however, that there are strategies (see, for instance, [12]) that allow to significantly reduce this amount without violating the fact that *DPS* converges to a valid solution. Furthermore, we note that at the termination of our algorithm there may be fractional paths for some source/destination pairs that are not connected to the destination. However, if the number of rounds performed by *DPS* is large enough, the weight of such a path is guaranteed to be so small that it can be neglected. Therefore, we simply eliminate such paths.

3 Performance study

We developed an accurate simulation environment for storage area networks to compare *DPS* with different other path selection strategies. In this section, we present the results we obtained for meshes, expanders, hypercubes, DeBruijn networks, and butterfly networks (see [16]).

We compare an integral and a fractional path system computed by *DPS* with three other path systems. In the first case, the path system only consists of shortest paths. For meshes, hypercubes, and butterfly networks we use standard shortest path selection strategies (such as the $X - Y$ routing for meshes). These strategies provide a system of integral paths with an optimal *congestion*. (The congestion is defined as the maximum number of paths crossing a link.) For the expander networks we computed shortest path systems with the help of the Dijkstra algorithm, and for the DeBruijn networks we use a standard path system. Furthermore, we used path systems which are constructed according to fractional and integral solutions of the multicommodity flow algorithm of Garg and Knemann, in the following called *GK*-algorithm (see Section 1.2).

In order to compare the path systems determined by the different strategies, we looked at different criteria. First, we calculated the maximum expected congestion caused by sending



(a) Mesh

(b) Expander

(c) Hypercube

(d) DeBruijn

Figure 2: Max. congestion of router links.

one packet per source/destination pair (that is, the maximum expected number of paths taken by the packets that cross a link), using the given path system. Second, we simulated the situation that packets are continuously injected into the storage network, using the following approach. We assume for every network except of the butterfly networks that every node of the network serves as a source and a destination, i.e. every node is connected to a disk array and a local area network. Each node injects 16.250 packets per second into the system with a fixed size of 8 KBit. Hence, each node injects 128 MBit/s. The destinations of the packets are evenly distributed among the m possible destinations of the network. This results in a bandwidth requirement of $d_{i,j} = \frac{128MBit/s}{m}$ for each source/destination pair (s_i, t_j). The edge capacity is 500 MBit/s. We use the FIFO rule to send the packets along the selected paths. The simulations provide information about the link load (i.e., the maximum expected amount of packets injected into the system within a time unit that intend to cross a particular edge) of the active routers and the packet latency. Finally, we evaluated the number of rounds needed by the *DPS* algorithm to terminate.

Figure 2 deals with the first evaluation criteria and depicts the maximum expected congestion for the different path systems. In order to compute these values, we assume that each source sends exactly one packet to each destination. As noted above, for the hypercube

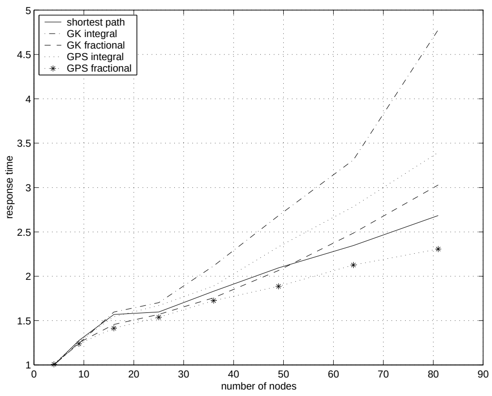


Figure 3: Average response time for a packet request.

and mesh networks the simple shortest path systems have a minimal congestion. Therefore, these path systems serve as a reference for our *DPS* algorithm. It is remarkable that the expected congestion for the fractional path systems calculated by *DPS* is within 2% of the minimum possible congestion. Comparing the fractional solution of the *DPS*-algorithm with the solutions for expander and DeBruijn networks, *DPS* has an up to 30% better congestion. Another observation is that the integral solution of the *DPS*-algorithm is up to 83% worse than the fractional solution. Furthermore, the fractional and integral solutions of the *GK*-algorithm (obtained after running it a reasonable amount of time) do not provide better solutions than the shortest path systems and the path systems calculated by our *DPS*-algorithm.

Now we turn to the second evaluation criteria, the simulation of storage networks. Figure 3 shows the influence of the path selection strategies on the response time of a data request. The response time is defined as the amount of time between the injection of a data request and the delivery of the data packet to the requesting user. We normalized the response time to the response time of a 2x2 network with a shortest path system. Although the congestion obtained by using *DPS* is higher than that of the shortest path systems, the response time under *DPS* is up to 16% shorter. The reason for this is that in our simulations we assumed data blocks to be of size 64KByte. Thus, a requested data block will be sent back in 64 packets of 8KBit each. Since a fractional path system provided by *DPS* allows these packets to follow different paths, whereas for the shortest path systems all of these packets have to follow the same path, the link load is more evenly balanced in the case of *DPS*.

Table 1: Link load of an Expander with 64 nodes.

	Expander with 64 nodes		
Algorithm	avg. load	max. load	min. load
shortest path	1.00	1.43	0.41
int. GK	1.19	1.88	0.47
frac. GK	1.26	1.63	0.82
int. DPS	1.04	1.40	0.53
frac. DPS	1.03	1.30	0.68

Table 2: Link load of an 8×8 mesh.

	8x8 Mesh		
Algorithm	avg. load	max. load	min. load
shortest path	1.00	1.36	0.55
int. GK	1.15	1.89	0.27
frac. GK	1.30	1.60	0.93
int. DPS	1.03	1.79	0.28
frac. DPS	1.03	1.40	0.63

Table 1 and 2 give a detailed picture of the link load caused by the path selection strategies for the 8×8 mesh and an Expander network with 64 nodes. The numbers are normalized to the best possible average link load in these networks. The tables demonstrate that the fractional variants of *GK* and *DPS* always have a lower maximum link load than their integral variants. For the 8×8 mesh, the average and maximum link load of the fractional *DPS* is very close to the values of the optimal shortest path system. For the Expander network with 64 nodes, the fractional *DPS* has the best maximum link load.

Finally we present the results concerning the number of rounds until *DPS* terminates (see Figure 4). This number is even moderate for relatively large networks.

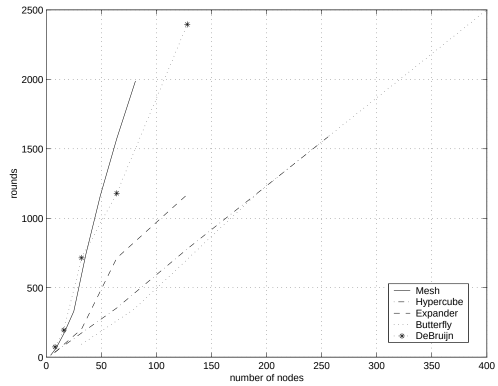


Figure 4: Number of rounds until *DPS* terminates for different network topologies.

Acknowledgments

We would like to thank the members of the software project "PGMUDAS" which has been held at Paderborn University during the year 1999. The members of that group implemented a simulation environment for our PRESTO server. Especially, we would like to thank Ralf Große Börger, Michael Heidebuer, and Guido Schütte. They implemented the path selection strategies and performed the simulations.

References

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

[2] J. Santos and R. Muntz. Performance analysis of the RIO multimedia storage system with heterogeneous disk configurations. In *Proc. of the 6th ACM Multimedia*, 1998.

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

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

[5] P. Raghavan and C.D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, 1987.

[6] S. Kapoor and P.M. Vaidya. Fast algorithms for convex quadratic programming and multicommodity flows. In *Proc. of the 20th Ann. Symp. on Theory of Computing*, 1986.

[7] P. Klein, S.A. Plotkin, C. Stein, and E. Tardos. Faster approximation algorithms for the unit capacity flow problem with applications to routing and finding sparse cuts. In *Proc. of the 22nd Ann. ACM Symp. on Theory of Computing*, 1990.

[8] S. Plotkin and E. Tardos. Improved bounds on the max-flow min-cut ratio for multicommodity flows. In *Proceedings of the 25th Ann. ACM Symp. on Theory of Computing*, 1993.

[9] B. Awerbuch and F.T. 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.

[10] B. Awerbuch and F.T. 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.

[11] Naveen Garg and Jochen Knemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proc. of the 39th Annual Symposium on Foundations of Computer Science (FOCS '98)*, November 1998.

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

[13] J.E. Boillat. Load balancing and poisson equation in a graph. *Concurrency: Practice and Experience*, 2(4):289–313, 1990.

[14] G. Cybenko. Load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing*, pages 279–301, 1989.

[15] Ralf Diekmann, Andreas Frommer, and Burkhard Monien. Nearest neighbor load balancing on graphs. In *Proc. of the 6th European Symposium on Algorithms (ESA'98)*, pages 429–440, 1998.

[16] F.T. Leighton. *Introduction to parallel algorithms and architectures : arrays, trees and hypercubes.* Morgan Kaufmann, 1992.