



PEPITO
IST-2001-33234
PEer-toPeer Implementation and TheOry

Deliverable no: D2.9

Final Report on Distributed Algorithms

(covering 2002.01.01 – 2005.04.30)

REPORT VERSION: first

REPORT PREPARATION DATE: 11th March 2005

CLASSIFICATION: Private

DELIVERABLE NO: D2.9 DUE DATE: Month 38 DELIVERY DATE: Month 38

PROJECT START DATE: 2002.01.01 PROJECT DURATION: 38 months

RESPONSIBLE PARTNER: KTH Stockholm

PARTICIPATING PARTNERS: KTH Stockholm, SICS Sweden, EPFL Lausanne, UCL Louvain

PROJECT COORDINATOR: Swedish Institute of Computer Science AB

PROJECT PARTNERS: EPFL Lausanne, INRIA Rocquencourt, KTH Stockholm, UCL Louvain,
University of Cambridge UK



**Project funded by the European Community under
the 'Information Society Technologies' Programme
(1998–2002)**

Contents

1	Introduction	3
1.1	Objectives	3
1.2	Achievements during 2002-2004	3
2	A framework for DHT-based overlay networks	4
3	Distributed k-ary System	5
3.1	Topology Maintenance	5
3.2	Replication in DKS	6
3.3	Network structure	6
3.4	Stack embedded messages for robust recursive lookup	7
3.5	Novel algorithm for the join operation	7
3.6	Novel algorithm for the leave operation	8
3.7	Novel algorithms for handling failures	8
3.7.1	Failure semantics	8
3.7.2	The novel approach for handling failures	9
3.8	One-to-many communication primitives in DKS	9
3.8.1	Broadcast	9
3.8.2	Multicast	10
4	Simulation environment for peer-to-peer algorithms	10
5	Formal specification and verification of DKS's algorithms	11
6	Symmetric overlay networks	11
7	Probabilistic algorithms	12
8	Proof of concepts	12
8.1	Integration of DKS's algorithms in the DSS	12
8.2	DKS-based WebLog application	12
8.3	DKS-Based file system	13
8.4	A peer-to-peer library within the Mozart/Oz platform	13
9	Information dissemination	13
9.1	Published papers	13
10	Attached papers	14

1 Introduction

This report, D2.9, constitutes the final report on distributed algorithms and services developed in workpackage 2 in the PEPITO project.

We proceed as follows. In Section 1.1, we recall the objectives of workpackage 2. In Section 1.2, we give a list of the major achievements. Then, from Section 2 to Section 7, we elaborate on each of the achievements listed. In Section 8, we summarize some of the work performed to proof the applicability of our designs. In Section 9, we show the results of our efforts in disseminating information. Finally, a list of attached papers is given in Section 10.

1.1 Objectives

As it became apparent in the first progress report, the main objective of WP2 was to provide a robust, low communication and scalable application-independent infrastructure that eases the construction of peer-to-peer applications. This entails design of a substantial number of distributed algorithms capable of supporting such an infrastructure. These include at least:

- ▷ Distributed algorithms for constructing overlay networks with desirable properties such as scalability, fault-tolerance and providing location-independent one-to-one and one-to-many communication primitives.
- ▷ Distributed algorithms for replication.
- ▷ Distributed algorithms for a large-scale directory services.
- ▷ Probabilistic distributed algorithms that might prove powerful in environments with high dynamism.

1.2 Achievements during 2002-2004

The work of WP2 can be summarized as follows:

1. The construction of a general theoretical framework for DHT-based overlay networks, describing how structured peer-to-peer systems can be constructed.
2. The development of algorithms for the Distributed k-ary System (DKS) which is based on the general theoretical framework. This includes algorithms for routing, topology maintenance, group communication, replication, and fault-tolerance.
3. A simulation environment for peer-to-peer algorithms. The simulation environment is used to simulate all the different peer-to-peer systems which can be constructed from the general theoretical framework.
4. A formal specification and verification of DKS's algorithms.
5. Symmetric overlay networks with high scalability based on the DKS system.
6. Proof of concepts
 - ▷ A middleware written in JAVA implementing the DKS.
 - ▷ Integration of DKS's algorithms in the DSS.

- ▷ A DKS-based WebLog application.
- ▷ A DKS-based file system.
- ▷ A peer-to-peer library within the Mozart/Oz platform.

In the rest of this document, we shall elaborate on each of these points in turn. Though the document is intended to be less technical, we opted to provide some technical aspects in order to help the reader appreciate the contributions. We shall, for most of the items mentioned above, give the motivations of the problem addressed and a brief description of the proposed solution.

2 A framework for DHT-based overlay networks

We have provided a general framework for understanding, analyzing and deriving structured peer-to-peer overlay networks. The framework has changed over the course of the PEPITO project, and has become more general and simpler over time.

The framework is formalized, as reported in [?], and now includes constant-degree networks, where nodes contain routing tables of a constant size, while the lookup efficiency is logarithmically bounded in the number of nodes in the system.

We showed that a wide-range of peer-to-peer systems, including [?, ?, ?, ?, ?, ?], can be formalized using a “metric” space with a distance function, d , on it. We defined a structured peer-to-peer system as a peer-to-peer system where nodes have identifiers in the metric space, and their routing pointers point to nodes which have been constrained by the distance function on the metric space.

Given this, we defined that the system needs to embed a graph into the metric space to facilitate efficient searching. We showed that structured P2P systems are logically embedding two k -ary trees in the metric space, an I-DAG, specifying intervals in the metric space, and a R-DAG, specifying responsible nodes for the corresponding node in the I-DAG.

We showed how search can be done in these systems by embedding either *relatively space divided* k -ary trees or *fixed space divided* k -ary trees, or a hybrid of them, each one giving different properties. The two ways to divide the space can be summarized as follows:

- ▷ *Fixed division of the space.* In the fixed division of the space, there is a *virtual global* partitioning that is made for all potential nodes of the system. A node that joins the system will obtain a routing table by picking some nodes from some of the portions of the global partition. This is the case for systems such as Pastry[?], Tapestry[?], P-Grid[?] and Kademia[?].
- ▷ *Relative division of the space.* In the relative division of the space, for each node, there is a logical division of the virtual space that is specific to the node. Furthermore, any node n by knowing the identifier of any other node n' , can compute the logical division of the space relative to node n' . This is the case for systems such as Chord[?], DKS[?] and Koorde[?].

Different space divisions have different inherent properties. For instance, using fixed-space division, proximity information could be used when constructing routing tables. Relative space

division, on the other hand, is more deterministic and allows for more rigid topology maintenance techniques (refer to Section 3.1). The hybrid of the two can be used to achieve constant-degree systems, where the number of entries in the routing table are constant, while the number of hops needed to perform lookups is logarithmic in the number of nodes in the system.

3 Distributed k-ary System

Using the results that were summarized in the previous section, algorithms were designed for a structured peer-to-peer system called Distributed k-ary System (DKS). We next summarize the topology maintenance algorithms designed and implemented in the DKS system.

3.1 Topology Maintenance

Effective maintenance of overlay networks is perhaps the main challenge in peer-to-peer systems, where one typically assumes high dynamism. In the first and second progress reports of WP2, we summarized different techniques to overcome this challenge from a fundamental stand-point.

The DKS system now supports the following topology maintenance techniques:

1. *Local atomic action*: To account for the high dynamism assumed in peer-to-peer systems, the framework proposes the use of local atomic action[?] for inserting new nodes and for managing the departure of nodes that wish to cooperate while leaving the system. The local atomic action is important to ensure strong guarantees for locating any item inserted into the system.
2. *Correction-on-use*: Correction-on-use[?] lazily corrects outdated routing entries by piggybacking local topology parameters in routing messages such that the receiver of a routing message can detect inconsistencies in the current topology. One of the main advantages of correction-on-use is that its bandwidth consumption is low and it can therefore be applied to systems already deploying other types of topology maintenance. Another advantage is that some topology maintenance strategies, such as periodic stabilization, can lead to queries reaching the wrong destination. Using correction-on-use together with such topology maintenance strategies ensures that queries always end up at the right destination. One disadvantage of correction-on-use is that it builds on local atomic operations, which might be complicated to implement, depending on the degree of guarantees to be achieved. Another disadvantage with correction-on-use is that it assumes there are enough routing messages to keep the topology information up-to-date. If there is not enough routing messages, the DKS system relies on correction-on-change for topology maintenance.
3. *Correction-on-change*: Correction-on-change[?] eagerly identifies and notifies all nodes that have outdated routing information as a result of a node joining, leaving, or failing. Effective failure handling is simplified as the detection of a failure triggers a correction-on-change which updates all the nodes that have a pointer to the failed node. The resulting system has increased robustness as nodes with stale routing information are immediately updated. Each event triggers a correction-on-change event which needs $O(\log_k^2(n))$ messages, which also is the lower bound for the number of messages needed in periodic stabilization, which is used by Chord-alike systems.

3.2 Replication in DKS

We devised a symmetric replication scheme for increased security and performance in structured overlay networks [?].

Structured peer-to-peer networks rely on replication as a basic means to provide fault-tolerance in presence of high churn. Most of the existing systems replicate on either the so-called successor-list [?] or on the leaf-set [?]. We show that both these schemes to varying degree have limitations. First, they both require at least $\Omega(f)$ messages for each join and leave event to maintain a replication degree of f . Often, however, an epidemic algorithm is used with a cost of $\Theta(f^2)$ messages sent in each round. Second, in both schemes, the requesting peer cannot directly route to a specific replica. Instead, the request is routed to the first replica that is encountered, which then can forward the request to the desired replica. The first peer storing the replica is thus a bottleneck, which can fail, decelerate the operation, or launch a security attack. The successor-list scheme is worse in this respect, as the first replica met is always the same for a given item. The possibility to route to a specific replica is useful if an insertion or update is desired, or if several replicas need to be looked up concurrently.

Our generic replication scheme, called *symmetric replication*, only needs $O(1)$ message for every join and leave operation to maintain any replication degree. The scheme is applicable to all existing structured peer-to-peer systems, and can be implemented in an end-to-end fashion on-top of any DHT. The scheme has been implemented in the DKS system, and is used to do load-balancing, end-to-end fault-tolerance, and to increase the security by using distributed voting.

Furthermore, the scheme can be used to implement proximity neighbor selection in the routing process, such that routing is done to nodes that have proximity with the current node.

3.3 Network structure

One of the main characteristics of the DKS overlay networks is that the outdated routing entries are corrected on-the-fly, to avoid unnecessary bandwidth consumption that results from the use of periodic stabilization as it is the case in most of the previous peer-to-peer infrastructures such as Chord, CAN and Pastry.

In a DKS overlay network, a node n updates its routing table in each of the following situations.

1. *Upon receipt of a regular message.* when a node n receives a “regular” message, for example a lookup request, from another node n' , node n adapts its routing table to account for the existence of n' if necessary. This will be the case when node n should have, but does not have node n' in its routing table.
2. *Upon receipt of a notification.* In a DKS overlay network, when a node n sends a regular message to another node n' , node n' first determines whether node n used a correct routing entry at the time of the sending operation. If node n' finds out that node n did not use a correct routing entry, node n' sends a notification to node n . This notification message serves to inform node n about the fact that it has used an outdated routing entry. In order to help node n correcting itself, node n' embeds a candidate node c that node n uses to replace n' in its routing table.

In our previous design, the candidate node sent by n' was the predecessor of n' . This has the drawback that the convergence of the overlay network towards the perfect state, in

which every node has correct routing entries, might be slow. To overcome this problem, we have introduced a *back-list* component at each node. In the back-list, a node maintains information about a predefined number of its predecessors on the circular identifier space (a ring in which for a given node n , the successor of n is the first node met in clockwise direction and the predecessor of n is the first node met in the counter-clockwise direction). Exploiting the back-list component, the convergence of a DKS overlay network towards a perfect state is made faster, as the candidate node used for correction is now selected from a set of candidates. Though this acceleration of the convergence of the overlay network towards a perfect state can be proven analytically, we validated it by means of simulations.

With the addition of the back-list component, special care had to be taken for managing joins, leaves and failures. This led us to design novel algorithms for handling joins, leaves and failures. We elaborate on the novel algorithm for the join operation in subsection 3.5. In subsection 3.6, we sketch the novel algorithm for leave. Then in subsection 3.7 we explain the core ideas behind the novel algorithms for handling failures.

3.4 Stack embedded messages for robust recursive lookup

Depending on the application, lookup in a DKS overlay network can be done in one of the following styles: Recursive, Iterative or Transitive. These are well-known strategies borrowed from name resolution in distributed systems [?].

In the recursive lookup approach, upon receipt of a lookup request, a DKS node, if it cannot resolve the target key, will forward the request to another DKS node that is closer to the successor of the target key. This forwarding process is essentially an *interval routing* process, that continues until the successor of the target key identifier is met. From that point the result (successful or not) is returned on the reverse of the path, along which the request was forwarded, up to the originating node.

The problem with this approach is that when forwarding the result backward, the failure of one link/node of the reverse path will prevent the originating node from receiving the result of its request. To overcome this problem, we proposed a novel recursive lookup algorithm in which each request message carries a stack that serves to trace the path followed by the request during the key resolution process. Therefore, during the backward forwarding phase of the result, the stack is popped to determine to which node the result must be sent back. With this improvement, as long as the originating node remains in the system, it will receive the result of its request. In addition, the information carried by the stack is used for correction-on-use.

The price paid by doing this is an increase in the size of the messages. Since the maximum length of a lookup path is logarithmic, it follows that this increase does not impair the scalability of our design.

3.5 Novel algorithm for the join operation

In our preliminary design of the DKS overlay networks, each node had in addition to its routing table, a predecessor pointer and a list of its f successors. We call this list of successors at a node n , the *front-list* component of n . One of the purposes of the front-list, initially, was to increase the guarantees, as this list was used to achieve effective replication. We note that we have improved our replication strategy with the use of symmetric replication previously discussed in Section 3.2.

Given that in a DKS overlay network, each node now has a routing table, a back-list and a front-list and that the system entirely relies on *correction-on-use* for maintaining routing information, we decided to search for a new algorithm for the join operation. As a result of our investigations, we developed a stabilizing algorithm for managing the join operation. The novel algorithm is stabilizing in the sense that when a new node n_j join the system, the vicinity (union of nodes in the back-list and in the front-list of the node), $\mathcal{V}(n_j)$, of n_j converges quickly towards the legitimate state if ever disturbances cease within $\mathcal{V}(n_j)$. However, the insertion of the node itself is done through a local atomic action, that involves only a few nodes (typically, the joining node, its current successor and its current predecessor) on the ring. Local atomicity of the join is important, as it avoids *false lookup failures*.

3.6 Novel algorithm for the leave operation

Work has been done to provide a novel algorithm for the leave operation. This, to incorporate the maintenance of both the back-list and the front-list of the nodes in the vicinity of the departing node.

The novel algorithm for the leave operation is stabilizing. Indeed, when a node n wishes to leave the system, the departure of n is coordinated by the successor of n . This coordination mainly consists in ensuring that each node in the vicinity of the departing node quickly gets a legitimate back-list and/or a front-list, if ever disturbances cease in the vicinity of the departing node. To achieve this, the coordinator triggers a *local diffusion* mechanism that stops as soon as each node in the vicinity of the coordinator has a legitimate back-list and/or a front-list.

Remote nodes not in the vicinity of the n , but that are pointing to n , will realize (actually, suspect) that node n has left the system when they try to communicate with n , and this will trigger fault-tolerance mechanism that we discuss in the next sub-section.

3.7 Novel algorithms for handling failures

Fault-tolerance is probably one of the most challenging tasks in distributed computing in general and in exponential overlay networks in particular.

A significant step has been taken to develop novel effective mechanisms for handling failures. Currently, we are preparing a publication for the developed algorithms. In this section, we shall only sketch the ideas. Before, doing so, we will first present the failure semantics.

3.7.1 Failure semantics

The first step in designing fault-tolerant algorithms is to set up the failure semantics. To introduce our failure semantics, let us consider a simple example. Let n and n' be two nodes (processes) that are connected by a communication channel (e.g. a TCP connection). Assume that node n sends a message MSG to node n' and that node n expects to receive a reply RPLY from n' within T time units. The following two scenarios can be observed when node n does not receive a timely RPLY:

1. node n receives within T time units, a notification message saying that node n' has failed. Such a notification will typically be sent by the OS above which node n' was running. In this case, from the node n perspective, there is an *accurate failure* situation with respect to n' . The failure situation from the node n stand-point is accurate, because node n has received information it can rely on.

2. node n does not receive within T time units, any message from n' nor from the OS above which n' is (or was) running. In this case, node n knows for sure that there is a failure situation. However, node n cannot say for sure what is happening. In this case, we say that from the node n perspective, there is an *inaccurate failure* situation with respect to n' (a timeout event).

From the above two scenarios, we distinguish two types of failures: *accurate failure* and *inaccurate failure*. We chose these names to embody the important fact that a node observing a failure situation either has accurate or inaccurate information about the observed failure situation.

As far as failure detection is concerned, a node n detects an accurate failure when it receives a reliable notification. Though this type of failures is rare, they do occur. A node n detects an inaccurate failure when an expected event does not occur in a timely fashion, i.e. a timeout occurs.

3.7.2 The novel approach for handling failures

In our earlier design of the DKS, we provided a simple approach to handle accurate failures only. The proposed approach worked fine. However, the convergence of the network towards perfect state was slow.

In our current design, we handle both accurate and inaccurate failures. The principle used for handling both types of failures is as follows. When a node n detects the (accurate/inaccurate) failure of a node n' , node n triggers a local stabilizing diffusion mechanism within the vicinity of node n' . This diffusion process has as effect to correct all nodes in the vicinity of n' , if the failure was accurate. In the case the failure was inaccurate, the local stabilizing diffusion process enables to gather enough information in order to take an appropriate action. The local diffusion is stabilizing in the sense that it stops as soon as all nodes in the vicinity of n are in "correct" state.

It is worth noticing that with the development of our correction-on-change technique, we have made the failure handling simpler. With the local stabilizing diffusion, one can ensure quick convergence of the vicinity of a node to a legitimate state. Using correction-on-change, all nodes that need to be corrected when a change occurs are determined and notified using a restricted version of our correcting-broadcast algorithm.

3.8 One-to-many communication primitives in DKS

We have enhanced the API provided by the DKS family of overlay networks. One-to-many communication primitives are important for various distributed applications. We started to investigate the broadcast primitive in structured overlay networks during the year 2002. The result of our first attempt was published in [?].

3.8.1 Broadcast

The broadcast algorithms designed in the beginning of the PEPITO project have been extended to be able to cope with outdated routing entries, that are common in structured overlay networks that build upon interval routing and correction-on-use, as it is the case in DKS overlay networks.

We developed two broadcast algorithms that integrate our correction-on-use philosophy. The proposed algorithms have the particularity that they correct outdated routing entries while disseminating broadcast messages to all nodes in the system without redundancy. We call these broadcast algorithms, correcting broadcast algorithms [?].

Most importantly, the latest broadcast algorithm does not induce any redundant messages, and only requires N messages to broadcast to N nodes. Many other broadcast systems based on structured peer-to-peer systems, such as multicasting in Content-Addressable Networks (CAN)[?], suffer from the sending of redundant messages, and thus need more than N messages to broadcast to a system of N nodes.

3.8.2 Multicast

Effort has been put in understanding facets of multicast in structured peer-to-peer overlay networks. After a careful analysis of the existing approaches to provide multicast in such overlay networks, we were able to come up with an interesting design that builds upon our work on broadcast.

Briefly, the idea consists of creating for each multicast group, an instance of DKS overlay network. Then, dissemination of multicast messages within a given group can be achieved using any of our correcting broadcasts. Details are given in [?].

Unlike previous attempts to multicast in DHTs, our system does not build a single source tree for each multicast group and does not require non-group members to participate in the dissemination of multicast messages, as in [?]. Furthermore, our multicast approach does not send redundant messages as in [?]. Our multicast design is flexible, as each multicast group can be tailored to fit specific requirements.

4 Simulation environment for peer-to-peer algorithms

Designing correct, robust and scalable distributed algorithms is a challenging task, that often requires substantial amount of time. A common approach to increase the designer's confidence is to experiment with simulations, though that is not enough to ensure correctness.

We built a simulation environment for distributed algorithms in large, and for fault-tolerant large-scale peer-to-peer algorithms in particular.

During the first year of the project, inspired by the initial simulation environment from MIT, we urgently developed a working simulation environment. Unfortunately, when we started to perform experiments with large number of nodes, the limitations of our earlier simulation environment became really undesirable. The major problems we had to address were the following:

1. The lack of an efficient mechanism to deal with the selection of identifiers. For instance, how to quickly find a "free" identifier to be assigned to a joining node. Recall, that in peer-to-peer systems, one typically assumes that the assignment of identifiers is achieved by means of a hash function (typically, SHA 1). Simulating this in face of high dynamism is not a trivial task.
2. The lack of flexibility in generating events in the system. In the previous simulation environment, events were pre-generated using lottery scheduling. This has significant impact on the performance as well.

We addressed these problems and provided effective solutions. For the first problem mentioned, we were able to re-discover the power of the shuffling algorithm (see for instance in [?], page 318), which we carefully exploited to achieve selection of identifiers in constant time, which is a great improvement when compared to the $O(N)$ time complexity algorithm that was used in the MIT simulation environment.

In addition to the lottery scheduling previously used for generating events, we now provide implementation of mechanisms that allow generation of events according to the Poisson distribution. Interestingly, the simulation environment is now flexible, as it can be instructed to generate events on-the-fly.

5 Formal specification and verification of DKS's algorithms

Although not strictly part of WP2, we started work on formal specification and verification of some of the DKS's algorithms. This work started with a tight collaboration with our colleagues in WP1. We report it here to make the document self-contained.

As a first step in the formal verification of peer-to-peer protocols, we studied the DKS lookup algorithm in a static setting, i.e., where no processes may join or leave the system. We expressed the specification of lookup as well as appropriate portions of the DKS lookup algorithm in a value-passing process calculus, and related the specification and the implementation by weak bisimulation [?]. Interestingly, even though weak bisimilarity is a priori not sensitive to the number of internal messages (or even divergent behavior), the performance-driven design of the algorithm required us to prove performance properties (maximum number of messages per lookup) in order to obtain the bisimulation.

6 Symmetric overlay networks

As shown by the first progress report on distributed algorithms and services, we started work on symmetric structured overlay networks. Symmetry is a useful property in structured peer-to-peer overlay networks, as it can for instance serve to achieve notification in place when the overlay network changes, due to peers voluntarily joining and leaving the system.

Our preliminary investigation of the symmetric exponential overlay networks led us to S-Chord[?]. The S-Chord design has two main limitations. First is its rigidity. Indeed, S-Chord is restricted to the base 2. Therefore, it is not possible to specify an arbitrary base $k \geq 2$, according to which routing intervals are going to be built. Second, the scalability of S-Chord is limited, as we shall argue in the next subsection. To overcome these limitations, we have studied how to build parametrized logarithmic overlay networks along the line of the embedding of k -ary trees assuming relative space division. By parameterizing the base of the exponential growth of the system size, one has the possibility to tune the infrastructure according to the application domain. As a result of these investigations, we were able to come up with a design we call Tango. An interesting question that come in mind concerns the optimality of the network size for a given routing table size in the context of exponential overlay networks. We looked at this problem in the case of overlay networks where each node maintains a routing table whose size is logarithmic of the number of nodes in the system.

By considering systems such as Chord and DKS, in which the relative division of the space is exploited, we have observed that there is an unused path redundancy. Based on this observation, we designed Tango in a way that avoids this problem, thereby improving the scalability

of the system by a constant factor.

7 Probabilistic algorithms

Probabilistic algorithms were one of our main tasks in WP 2. In Deliverable D2.5 we give a more elaborate description of our achievements in the area of probabilistic algorithms. In this report, we only present a high-level overview of our work on probabilistic algorithms in order to make this document self-contained.

In peer-to-peer context, high-dynamism is typically assumed. In such a context, probabilistic algorithms seem to be suitable. We therefore explore the design of probabilistic algorithms for a number of primitive services including broadcast. The question considered in this setting is how can we disseminate information in a reliable manner among a large set of processes, possibly varying in time, and possibly connected by unreliable channels.

To solve the above mentioned problem, we provided a new probabilistic specification of reliable broadcast communication primitives, called Δ -Reliable Broadcast. Our goal here was to precisely capture the reliability of practical broadcast algorithms. To demonstrate the applicability of our specification, we used it to measure and compare the reliability of two popular broadcast algorithms, namely Bimodal Multicast and IP Multicast. The details of this work appear in [?].

Building on our precise specification of the reliable broadcast problem we were able to come up with an algorithm to implement a probabilistic broadcast in a completely decentralized manner: no process is supposed to know all processes of the system[?].

Another important contribution we made in the context of probabilistic algorithms is the development of a *data-aware* multicast algorithm for publish/subscribe systems [?]. The proposed algorithm overcomes various limitations of existing approaches. In our design, published events are propagated within small groups in a gossip-based manner, and dissemination between groups is achieved in a bottom-up approach imposed by the topic hierarchy.

8 Proof of concepts

To demonstrate the potential of our designs, efforts have been put into the investigation of some example applications. This appeared to be necessary to enhance the API that is going to be provided by the peer-to-peer infrastructure that will result from the PEPITO project.

8.1 Integration of DKS's algorithms in the DSS

Work was done for the integration of DKS's algorithms into the DSS (reported in WP 3). We mainly studied the DSS to investigate the possibility of separating the communication layer of the DSS from its consistency protocols. The goal of this study meant for the preparation of the integration of the DKS's algorithms in the DSS.

8.2 DKS-based WebLog application

In an internal project at SICS, an implementation of the DKS within the DSS is used for the development of a DKS-based WebLog application.

8.3 DKS-Based file system

Work on a distributed secure read/write file-system based on the DKS was finished. The work was mainly done by two master students at KTH. The resulting system can support reads and writes in a secure manner by using self-certifying data. At the same time, updates are efficient, and do not require moving whole directories as a novel technique is used to encrypt blocks with their content hashes[?].

We note that the work done for developing Keso, was introduced to replace the directory service promised initially in the technical annex.

8.4 A peer-to-peer library within the Mozart/Oz platform

To experiment with the ideas developed in our attempt to incorporate symmetry and provide high scalability in logarithmic exponential overlay networks, we have started work in the implementation of a peer-to-peer library, named P2PS [?], within the Mozart/Oz platform.

9 Information dissemination

The work done in WP 2 has led to a number of documents. Some of them have been published and some are under preparation for publication.

9.1 Published papers

1. A. Ghodsi, L. O. Alima and S. Haridi. “Low-Bandwidth Topology Maintenance for Robustness in Structured Overlay Networks”, *Proceedings of the 38th HICSS Conference*, 2005. We note that this paper got the best paper award in the software track.
2. Luc Onana Alima, Ali Ghodsi, Per Brand and Seif Haridi. “Multicast in DKS(N, k, f) Overlay Networks”. Regular paper in the *Proceedings of the 7th International Conference on Principles of Distributed Systems (OPODIS 2003)*, December 10-13, La Martinique, France.

This paper is an elaborated version of the poster paper.
3. Ali Ghodsi, Luc Onana Alima, Sameh El-Ansary, Per Brand and Seif Haridi. “Self-Correcting Broadcast in Distributed Hash Tables”. *Proceedings of the 15th IASTED International Conference Parallel and Distributed Computing and Systems (PDCS)*, November 3-5, 2003 Marina del Rey, CA, USA.

In the first progress report, we mentioned the result of our first attempt to do broadcast in DHT-based overlay networks. It has become apparent that our first broadcast algorithm that appear in[?], was not suitable to cope with outdated routing information that are common in DKS overlay networks.

This paper presents two broadcast algorithms that are suitable for DHT-based overlay networks designed with correction-on-use in mind. The presented algorithms are very effective in that they allow to disseminate message in an optimal way while correcting outdated routing information on-the-fly.

4. Luc Onana Alima, Ali Ghodsi, Sameh El-Ansary, Per Brand and Seif Haridi. "Multicast in DKS(N, k, f) Overlay Networks". Poster paper in the *Proceedings of the Third International Conference on Peer-to-Peer Computing (P2P2003)*, September 1-3, Linköping, Sweden.

This is a poster paper that presents our preliminary idea on how to achieve multicast in DKS overlay networks.

5. P. Eugster, R. Guerraoui, and P. Kouznetsov, "Delta-Reliable Broadcast", IEEE International Conference on Distributed Computing (ICDCS'04).

This paper is a revised version of a technical report that was mentioned in the first progress report. It gives a precise specification of the probabilistic broadcast problem through the notion of Delta-reliable broadcast. Furthermore, it provides a comparison of various broadcast algorithms.

6. P. Eugster, R. Guerraoui, S. Handurukande, A-M Kermarec and P. Kouznetsov, "Lightweight Probabilistic Broadcast", ACM Transactions on Computer Systems.

This paper gives an algorithm to implement a probabilistic broadcast in a completely decentralized manner: no process is supposed to know all the processes in the system.

10 Attached papers

The following papers are attached to this report.

1. A. Ghodsi, L. O. Alima and S. Haridi. "Low-Bandwidth Topology Maintenance for Robustness in Structured Overlay Networks", *Proceedings of the 38th HICSS Conference*, 2005. We note that this paper got the best paper award in the software track.
2. Ali Ghodsi, Luc Onana Alima, Sameh El-Ansary, Per Brand and Seif Haridi. "Self-Correcting Broadcast in Distributed Hash Tables". *Proceedings of the 15th IASTED International Conference Parallel and Distributed Computing and Systems (PDCS)*, November 3-5, 2003 Marina del Rey, CA, USA.
3. Luc Onana Alima, Ali Ghodsi, Per Brand and Seif Haridi. "Multicast in DKS(N, k, f) Overlay Networks". Regular paper in the *Proceedings of the 7th International Conference on Principles of Distributed Systems (OPODIS 2003)*, December 10-13, La Martinique, France.
4. P. Eugster, R. Guerraoui, and P. Kouznetsov, "Delta-Reliable Broadcast", IEEE International Conference on Distributed Computing (ICDCS'04), to appear.
5. P. Eugster, R. Guerraoui, S. Handurukande, A-M Kermarec and P. Kouznetsov, "Lightweight Probabilistic Broadcast", ACM Transactions on Computer Systems, Vol. 21, NO. 4, November 2003, pages 341-374.