Distributed Algorithms Adapted to Knows-Based Systems

Thomas Eirich

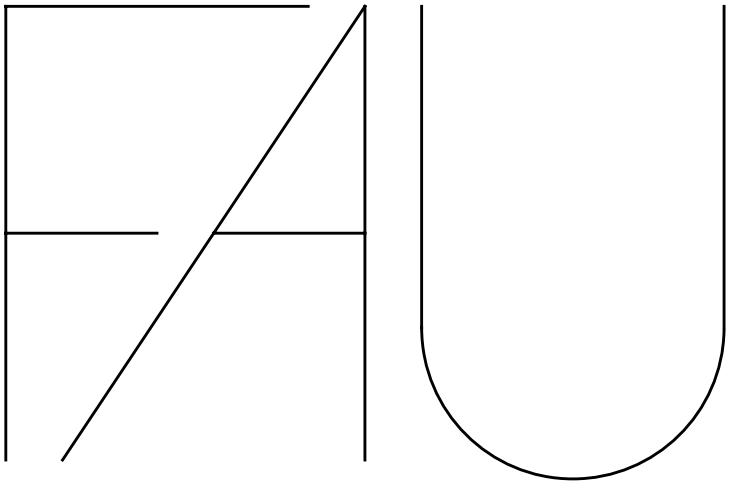
Dec. 1994 TR-I4-94-25

Technical Report

Computer
Science Department

Operating Systems — IMMD IV

Friedrich-Alexander-University
Erlangen-Nürnberg, Germany



Distributed Algorithms Adapted to Knows-Based Systems

Thomas Eirich

eirich@informatik.uni-erlangen.de
University of Erlangen-Nürnberg, IMMD 4
Martensstraße 1
D-91058 Erlangen, Germany

Abstract. This paper presents distributed algorithms for a computational model based on knows relations. Usually, a distributed system is modelled by a graph. The edges of the graph represent static communication links and messages between nodes has to follow a path in the graph. This model is called the topology model.
In contrast to this model we assume that any to nodes can exchanged messages directly provided the sender knows the identification of the destination. A network layer abstracts from the underlying communication topology and provides efficient delivery of messages. This model seems to be more convenient with respect to today's software layering. Objects or processes communicate via messages, RPCs or DSM without any knowledge of the physical network.
We present three classes of algorithms, simple propagation of information, echo algorithms, and election algorithms tailored to this model. Each class is based on the algorithms of the previous one. The time complexity of the algorithms is only half of those designed for network topologies.

1 Introduction

This paper presents basic algorithms for distributed systems based on knows-graphs. Usually, the distributed system is modelled by a graph. The nodes of this graph represent processors or processes, and the edges are communication links. Nodes communicate by exchanging messages over links. The details of the computational models are varying and concern whether links allow unidirectional or bidirectional communication, whether communication is synchronous, asynchronous, atomic or FIFO, or whether the algorithms consider links to be possibly down. But all these models have in common that messages between two nodes have to follow a path in the graph. Therefore, we call these models throughout topology based. The graph describes the physical topology of a distributed system which is assumed to be invariant to the distributed algorithm. There are algorithms dedicated to specific topologies because there is relevant hardware based on it (e.g. rings).

This is the essential difference between a knows-based and a topology based computational model is the ability of any to nodes to communicate. In the knows-based model any two nodes can exchange messages provided the sender *knows* the identification of the destination. A graph models the current system state where directed edges represent the *knows-relations* among nodes. Edges are subject to modifications as the distributed computation proceeds. A node can stand for a host, a process, or an object. A directed edge from *A* to *B* corresponds to the fact that node *A* stores the system-wide unique identification of node *B*. Such an identification could be an object identifier or a network address.

A knows based model abstracts from the physical topology and assumes that a network layer provides efficient delivery of messages across physical media. Such a model fits better into the layering of network software than an topology based approach. Of course, topology based algorithms could be employed in a knows based model but their performance would be inferior to adapted solutions. We will show that time and message complexities can be halfened.

We will start with a very simple algorithm which achieves propagation of information. Based on this algorithm we then present *echo* and *election* algorithms. These are basic algorithms and their operation principles are building blocks in solutions for several problems of distributed systems. Echo algorithms [4, 10, 12] are used to disseminate and collect information. They are used to create spanning trees for efficient broadcast delivery or to check connectivity of nodes [4, 12]. Election algorithms are based on echo algorithms and are employed to achieve mutual exclusion [4], to take snapshots [5, 8], to detect termination of computations [10], or to detect deadlocks [3].

The next section describes the knows based computational model in detail. Section three presents a simple algorithm to propagate information. Section four enhances this algorithm leading to two version of an echo

algorithm. Then, in section five election algorithms are presented based on the echo algorithms of the previous section.

2 Computational Model

The current state of the distributed system is modelled by a graph. The nodes represent computing entities. Each node has a system-wide unique identification. Node identifications are totally ordered. A node has local storage and receives messages destined to it. It can send messages to any node, including itself, provided it knows the identification of the destination.

A directed edge reflects the knowledge of one node about the other. New outgoing edges are created if a node stores identifications of other nodes either obtained by creating new instances or extracted from received messages. Edges are deleted if identifications are removed from a node's state.

The nodes run some distributed computation called the basic computation. The presented algorithms implement a control computation which can be initiated from the basic computation. The control computation communicates by exchanging control messages and maintains state information separate from the basic computation. This means, that the control computation can create a control graph by storing node identifications in its own state without disturbing the basic computation.

The communication layer available to the control computation provides asynchronous sending of messages. The identification of the destination must be specified. The message is reliably delivered after a finite but unpredictable delay. The communication layer does not guarantee FIFO order of messages.

All nodes execute the same control algorithm which can be activated either by the local basic computation or by incoming control messages. The control computation performs its processing atomically. Once, one of its procedures has been started it completes uninterrupted. The processing of a received message is first finished before the processing of the next is started or the basic computation is continued.

In all the presented algorithms the variable σ is the identification of current node. Zero is not a valid value for σ. Λ is a set containing the node identifications currently known to the basic computation. The only restriction to be fulfilled by the basic computation is that the knows-graph has to be complete. That is, there is a path between any pair of nodes. The algorithms will work

```
   procedure start(X):
       u ← u+1
3      send ⟨explorer: σ, u, {}, {}, X⟩ to σ

4  on receipt of ⟨explorer: i, u, V, P, X⟩:
5      return if engaged_i > u
6      P ← P – {σ}
7      V ← V + {σ}
8      if engaged_i < u then
           engaged_i ← u
10         P ← P+Λ–V
       end
12     if P ≠ {} then
13         foreach (Q, q) in Π(P)
14             send ⟨explorer: i, u, V, Q, X⟩ to q
           end
       end
```

Fig. 1 Propagation of information (PI_Π)

even without this restriction but then different nodes have a different view of the system and presentation is unnecessarily complicated. Uninitialized variables return either zero or an empty set depending on their usage. Messages are denoted in the form $\langle t\text{: } d_1, ..., d_n\rangle$ where t is the message type and d_i the data carried within the message.

3 Propagation of Information

The first basic algorithm achieves propagation of information (PI). Figure 1 shows the algorithm. The basic computation initiates the algorithm PI by calling the procedure *start* and supplying some data. The procedure immediately returns and the supplied information eventually will reach all the other nodes. Information is disseminated by explorer messages $\langle explorer\text{: } i, u, V, P, X\rangle$. The i denotes the initiator of the information X. The component u is a sequence number and the sets V and P are used to control the spreading of explorer messages.

A sequence number is necessary because a node can restart the algorithm even before the previous information has been completely disseminated. Since control communication does not provide FIFO channels, explorer messages with newer information may arrive before those with older information. Line 5 guarantees, that explorers carrying obsolete information will be extinct. In line 8 the sequence number is used to prevent cycles in the traversal of a particular propagation run. If an explorer arrives at a node which already has been informed its links are not considered again by late explorers.

A node maintains a variable u counting its own initiations of PI and also a set of variables $engaged_i$ recording the most recent sequence numbers of received explor-

ers. Consequently, several nodes can propagate their information without interference.

The presented algorithm PI is parameterized with a function Π which controls the degree of concurrency and the selection of successor nodes. The function

$$\Pi: M \rightarrow \{(M_1, m_1), ..., (M_n, m_n)\}$$

creates a partition of a set of nodes *M* and selects from every partition M_i an element m_i. There are two special functions

$$\Pi_{min}: M \rightarrow \{(M, m)\}$$

and

$$\Pi_{max}: M \rightarrow \{(\{m_1\}, m_1), ..., (\{m_{|M|}\}, m_{|M|})\}$$

which lead to the extrema of concurrent behavior. If Π_{min} is chosen then the algorithm visits all nodes sequentially while Π_{max} lets PI perform a depth parallel traversal. Note that Π need not to be invariant. It may be different at every individual propagation step. This enables the underlying communication system to influence the performance of the algorithm and to control induced system degradation. The number of partitions created by Π can be viewed as the propagation fan-out and induces network load. The grouping of nodes into partitions and the elected nodes from every partition can be chosen with network metrics in mind.

In order to be able to efficiently vary the degree of concurrency, an explorer contains two sets of node identifications. *V* records the nodes which already have been visited by ancestors of this explorer and *P* contains those nodes which should be addressed by descendants of this explorer. The set *V* prevents that already visited nodes are reconsidered by descendants. During a relay step the set *P* is possibly enlarged with new nodes (line 10) and is then divided into non overlapping sets (line 13). Each part of *P* is passed along with a successor explorer to a node out of this set.

Now, we proof that PI is correct by showing that two properties are fulfilled: explorer activity will eventually cease and every node will be visited by at least one explorer. That is, PI terminates and the information will reach all nodes.

Property 1: Explorer activity will eventually cease. Explorers will be extinct either because their information has become obsolete (line 5, $engaged_i > u$) or because they will not find any further uninformed nodes (line 12, P={}). Clearly, obsolete explorers cannot survive because subsequent runs of PI will reach all nodes and surviving obsolete explorers will be discarded everywhere.

Evidently, the interesting part is to show that explorer activity ceases in the absence of a subsequent initiation of PI. Let *N* be a finite set containing all potential nodes of the system while the PI algorithm lasts. The life times of the descendants of an explorer ⟨*explorer:* i, u, V, P, X⟩ are bounded by |*N*–V|. At every propagation step the visited node is added to *V* and removed from *P*. At least, after |*N*–V| steps *P* must be empty and propagation ceases according to line 12. *P* will be empty because line 10 ensures that already visited nodes are not added to *P*. This means, any node in *N* is added at most once to *P* and *P* is continuously lessened.

Property 2: Every node will be visited by at least one explorer. Of course, this is only guaranteed if there is no interference with a subsequent run of PI. We assume that some set of nodes *M* hasn't yet been visited by explorers. The knows-graph of the basic computation is connected and, hence, there must be an already visited node *n* which has a link to a node $m \in M$. The first explorer ⟨*explorer:* i, u, V, P, X⟩ arriving at *n* has created a descendant explorer with $m \in P$ because $engaged_i < u$, $m \notin V$, and $m \in \Lambda$. This means *m* will be addressed by a descendant of this explorer. According to property 1 explorer activity will cease. That is, the set P will become empty. The node *m* must have been removed from P which only happens (line 6) when node *m* is visited by an explorer.

The PI algorithm can be somewhat simplified if the parameter Π is invariant and one of Π_{max} or Π_{min}. Some components of explorer messages can be saved and some processing steps can be omitted.

4 Propagation of Information with Feedback

A user of the PI algorithm does not know when all other nodes have received the information. But PI can be easily enhanced to provide some feedback about this fact. Topology-based echo algorithms consist of two phases [4, 12]: the first phase disseminates data in concentric waves from the originator towards the *ends* of the graph. This first phase is like the PI(Π_{max}) algorithm. The second phase carries back echo messages towards the originator. The originator knows that every node has received its data if it has obtained echoes from all its incident links.

In a knows-based graph the second phase can be improved. Echoes need not to hop back over several nodes towards the initiator. We can send then directly back to the originator since its identification is included in explorer messages. But then, we have a termination prob-

```
   procedure Start:
2      f+ ← 0
       u ← u+1
4      send ⟨explorer: σ, u, {}, {}, 1⟩ to σ
       wait for f+= 1

   on receipt of ⟨echo: w, f⟩:
       return if w≠u
8      f+ ← f+ + f

   on receipt of ⟨explorer: i, u, V, P, f⟩:
       ...
       if P ≠ {} then
12         foreach (Q, q, g) in Π(P, f)
               send ⟨explorer: i, u, V, Q, g⟩ to q
           end
       else
16         send ⟨echo: u, f⟩ to i
       end
```

Fig. 2 PI with feedback (PIF-1$_\Pi$)

```
   procedure Start:
       K+ ← {}
       V+ ← {}
       u ← (u+1) mod 2
5      send ⟨explorer: σ, u, {}, {}, {}⟩ to σ
       wait for V+= K+

   on receipt of ⟨echo: V, K⟩:
       K+ ← K+ + K
       V+ ← V+ + V

10 on receipt of ⟨explorer: i, u, V, P, K⟩:
       ...
12     K ← K + Λ
       if engaged_i ≠ u then ... end
       if P ≠ {} then ...
       else
           send ⟨echo: V, K⟩ to i
17     end
```

Fig. 3 PI with feedback (PIF-2$_\Pi$)

lem: the originator must be able to determine if all outstanding echoes have been received. We present two solution to this termination problem which goes well together with the PI algorithm. Certain termination detection techniques, as message counting schemes [10, 7, 6], are not appropriate because they require two waves, two rounds over a Hamiltonian circle, or two traversals of a spanning tree. The time saved by directly sending back echoes would be wasted by expenses of termination detection.

Figures 2 and 3 show the extensions to PI in black while original code of PI is printed gray. Some unchanged lines have been omitted for brevity. The handling of the information variable X supplied by the basic computation to PIF has been left out for simplicity.

The sequence number u part of explorer messages can be reduced to a single bit if PIF is not restarted before the previous run has been completed. The single bit is necessary because it is compared with the state of the variables $engaged_i$ to determine if a node already has been visited by explorers of the current PIF run. PIF-1 allows initiations even if the previous run is in progress while PIF-2 allows a new run only after the previous one has reported completion. This behavior is interchangeable between both algorithms. They can be easily adapted to meet the application requirements.

Message activity of both algorithms will eventually cease because they are derived from PI. The difference between PI and PIF with respect to message creation is that terminal explorers cause one additional message. Explorer are called terminal if they do not create descendants in a relay step. They do not disappear as in PI instead they send an echo message back to the initiator. Echo messages do not cause further message traffic.

Algorithm PIF-1 detects termination by maintaining an invariant. Explorer and echo messages have a weight (component *f*). The sum of the weights of all explorers, echoes and the variable f^+ at the initiating node yield one. The invariant is initially true (see fig. 2 lines 2, 4) and it is maintained at every relay step. The function Π receives the current weight *f* of the arrived explorer and splits up this value so that each successor gets a fraction of *f* (line 12). The sum of the successor's weights equals *f*. And finally, before an explorer disappears its weight is sent back in an echo message to the initiator (line 16) and added there to f^+ (line 8). If f^+ has value one then explorer of this particular initiation do not exist anymore. From the properties of PI we can follow that every node has been visited.

The initial weight and the splitting of weights can be chosen so that it fits well in the binary representation of numbers. Floating point numbers can not be used because of rounding errors. This termination principle is used for instance in garbage collection algorithms [1, 2].

The second algorithm uses two sets of node identifications to determine if all nodes have been informed. The two sets V^+ and K^+ maintained at the initiator strive towards the fixed-point $V^+=K^+=N$ where N is the set of all informed nodes. V^+ contains all the nodes which have certainly been visited by explorer messages. An explorer while moving through the knows graph and records the visited nodes and the identifications of all other nodes it gained knowledge of (fig. 3 line 12). These sets

are stored in the components *V* and *K* of explorers and are passed on to echoes and are finally accumulated at the initiating node.

The termination criterion $V^+=K^+$ means that all the nodes have been visited which explorers have got knowledge of. PIF-2 maintains the following invariants in its explorer and echo messages: $P \subseteq K$ and $V \subseteq K$. The former is true because it is initially true (line 5) and because it is propagated to successor explorers ($P^{10} \subseteq K^{10} \Rightarrow P^{16}=(P^{10}+\Lambda-\{\sigma\}-V^{10}) \subseteq (K^{10}+\Lambda)=K^{16}$). Variables with subscript or superscript indices denote the state of the variable before or after the specific line. The latter is also proofed by induction. It is initially true (line 5). If the identification of a node σ is added to *V* (fig. 1 line 7) due to the arrival of an explorer then $\sigma \in K$ because at the sending node $\sigma=q \subseteq Q \subseteq P \subseteq K$. Fig. 1 line 7 is the only place where *V* is enlarged and we showed that the added element is already in *K*.

We show that the termination criterion is correct by contradiction. We assume $V^+ \subseteq K^+$ and no more explorers being in transit. There must be a node $m \in (K^+-V^+)$ and a node $n \in V^+$ with $m \in \Lambda_n$. Otherwise the graph is not connected. From $n \in V^+$ we can follow that there has been an explorer visiting *n* for the first time. This explorer must have created a descendant explorer *x* with $m \in P_x$ addressed to a node $m \notin V$. Since PI's propagation scheme is implemented correctly and since we assumed that no more explorers are in transit *x* must have turned into an echo message. This can have happened only if $P=\{\}$. A node *e* is only removed from *P* if it has been visited by an explorer. That is, node *m* must have been visited and should be in the set *V*.

Note, that PIF-2 also returns in V^+ the set of nodes which have been visited by explorer messages as a result of termination detection. Often, this is useful for the basic computation in case it wants to address all these nodes again [5]. PIF-1 can be easily modified to compute this set, too. The main advantage of these echo algorithms compared to traditional algorithms [4, 7, 8, 10, 12] is the improved time and message complexity. We compare PIF(Π_{min}) and PIF(Π_{max}) to possible solutions for topology graphs. Message exchange is considered to take one time unit in average. A node's computing time is neglected.

Time complexity equals message complexity for sequential traversal of graphs. Topology algorithms have complexity 2(|N|-1) while PIF(Π_{min}) has only complexity |N|. N is the set of nodes in the graph G representing the distributed system. In topology algorithms every node except the initiator receives an explorer and replies to this explorer with an echo message. PIF(Π_{min}) instead addresses every node except the initiator with an explorer message and only the last node sends back an echo message. The message size of PIF(Π_{min}) is bigger than for topology solutions but this is not considered harmful due to the high bandwidth of today's communication media. Time efficiency is considered the prior to message complexity [11].

Time complexity for depth parallel traversals is $2R_i(G)$ for topology algorithms vs. $2R_i(G)+1$ for PIF(Π_{max}). $R_i(G)$ denotes the radius of the graph G with the node i being the center. The radius is defined similar to the diameter of a graph. It is the longest of the shortest paths from i to any node in G. In the topology approach explorers first have to travel to farthest node before the echo message can start their way back towards the initiator. PIF(Π_{max}) abbreviates the way back from the farthest node to the initiator by directly sending to the initiator.

For both knows and topology based algorithms the determining factor for message complexity is the number of edges |E| in the graph G. They both induce a spanning tree on G. An edge (*n*, *m*) is part of the spanning tree if an explorer was sent from *n* to *m* and it was the first explorer arriving at *m*. Let *T* be the edges which are part of the spanning tree. All other edges are in $S=E-T$. On every edge in *S* two explorers cross each other and cause the sending of echo messages. For edges in *T* they behave slightly different. Topology based algorithms sent an explorer in one direction and an echo message in the other. For PIF(Π_{max}) there is only an explorer message per edge in *T* and some of these explorers turn into echo messages at the leaves of the tree. Thus, the message complexity is $O(|E|)-O(|N|)$ for both.

The crucial advantage of the adapted algorithms is their reduced time complexity though messages are larger and the number of created messages is only slightly less compared to topology based algorithms. We consider reduced time complexity more important because bandwidth of communication media has increased to an extent where processors have problems to keep pace with incoming data. This means that the number and sizes of messages have less influence on communication latency than communication setup per message.

5 Election

Now, we enhance the PIF algorithms of the previous section in order to obtain election algorithms. We will

consider two enhancements. The first is very simple. The second is more complicated but has a better performance. In order to elect a specific node, we have to break symmetry in the distributed system. The key property to accomplish this are the node identifications. We assume that they are totally ordered. To prevent that a node dominates we can combine the identifications with random numbers and use the identifications to solve a tie. Such totally ordered identifications are common place in distributed systems and do not imply any additional costs.

Each node wishing to become a leader starts an echo algorithm similar to the PIF algorithms presented in the previous section. Several nodes may do this concurrently. The explorers are ranked according to their initiator component *i*. If an explorer arrives at a node which already has been visited by a stronger explorer then it is simply discarded. This guarantees that only the echo algorithm of the node with the highest rank completes. This solution has the disadvantage that the work of other initiators is ignored. Their expenses to explore part of the knows-relationships are wasted. Mattern [8] proposes two solutions for a topology based computational model to improve this simple approach. We present another solution which is better than these two but which is only possible in knows-based computational model.

As before, every node wishing to become a leader starts an PIF echo algorithm. Figure 4 shows the resulting election algorithm if PIF-1 is used. The modifications to PIF-1 are: explorer and echo messages have some additional components and there is only one variable *engaged*. This variable is used to mark ownership of nodes. Each candidate tries to explore the knows-relationships and marks every visited node with its identification. This is used on the one hand to prevent cycles in the traversal of a single initiator and on the other hand to check if other candidates already have explored some parts of the graph. The traversal work of other candidates is not wasted because explorers are not discarded. If explorers arrive at foreign nodes the late explorer ignores the incident links (line 26-29 are not executed) because these links are handled the first explorer.

The effect is that each candidate conquers part of the graph. Explorers register in the component *O* the owners of neighboring regions (line 30) and finally report this information via echoes back to their originator (line 19, 38). After each candidate has completed its echo algorithms we view (C, O_C^+) as a graph. C is the set of candidates and O_C^+ are the edges connecting candidates. The echo algorithms is restarted on this graph by only a few of the original candidates (line 10). That is, we have the same problem as before but with a smaller graph and a reduced set of candidates. In order to distinguish explorer messages from adjacent rounds they are equipped with a bit *a*. The regions discovered by different initiators may have different sizes. Thus, some echo algorithms complete before others and send explorer messages out to neighboring candidates. Explores and the current and of the next round may arrive at some. Line 25 uses the local variable *a* and the information in the explorer to defer the processing of early explorers. Note that the echo algorithms will work properly since the variable *engaged* is reset at the end of every round.

```
   procedure Start:
2      do
           O+ ← {σ}
           f+ ← 0
           send ⟨explorer: σ, {}, {}, {}, a, 1⟩ to σ
           wait for f+= 1
           a ← (a+1) mod 2
           Λ ← Λ– {σ}
           engaged ← 0
10     while O+≠ {σ} ∧ σ = max(O+)

       if O+={σ} then
           foreach n in V+
               send ⟨winner: σ⟩ to n
           end
       end

   on receipt of ⟨echo: f, V, O⟩:
       f+ ← f+ + f
       V+ ← V+ + V
19     O+ ← O+ + O

   on receipt of ⟨winner: i⟩:
       engaged ← 0
       V+ ← {}
       a ← 0

24 on receipt of ⟨explorer: i, V, P, f, O, b⟩ ∧ a = b:
25     if engaged = 0 then
26         engaged ← i
           P ← P+Λ–V
28     end

29     O ← O + {engaged}
       P ← P – {σ}
       V ← V + {σ} + V+

       if P ≠ {} then
           foreach (Q, q, g) in Π(P, f)
               send ⟨explorer: i, V, Q, g, O, b⟩ to q
           end
       else
37         send ⟨echo: f, V, O⟩ to i
       end
```

Fig. 4 An election algorithm based on repeated execution of an echo algorithm (RE)

The concurrent echo algorithms are repeated until only one candidate survives (O^+={σ}, line 10). At least one

candidate, namely the one with lowest rank among all other candidates, will be dropped from candidacy at each round. This guarantees that the loop 2–10 terminates at exactly one node with $O^+=\{\sigma\}$.

The winner of the election finally notifies all other nodes about its leadership by a *winner* message. This message also triggers the reset of certain variables. After receipt of the winner message a node may restart the election algorithm. As long as a node is engaged in an election it may not restart the election.

The major advance of this solution is that the traversal performed concurrently by several candidates is not wasted. Though, the solution with repeated echo algorithms is not better that the simple solution with respect to worst case behavior we expect that it is much better with respect to the average case. As Mattern pointed out, the average performance of algorithms may significantly deviate from the worst case with high probability [8, 9]. We are currently about to run simulations on graphs which we consider typical for distributed computations.

6 Conclusion

We presented echo and election algorithms for a knows-based computational model. This model relies on a network layer which provides efficient delivery of messages to any node in the system provided the identification of the destination is known. Nodes in such a system have only limited knowledge about other nodes. This knowledge is subject to modification while a distributed computation proceeds. This scenery is more realistic for distributed applications and systems than the assumption of a static graph where edges represent available communication paths. Normally, distributed applications are not aware of the network topology. Distributed operating systems offer at least topology transparent services (e.g. IP) and sometimes location transparent communication (e.g. mobile objects). Even inside of a distributed operating system algorithms should not be designed for certain topologies and efficient routing should separated from algorithms with other concerns.

We started with a simple parameterized algorithm PI which achieves propagation of information. The traversal behavior of this algorithm can range from sequential to depth parallel. A function Π controls the traversal order and the degree of concurrency. This function need not to invariant instead it can be varied at every relay step. Via this function a network layer can hook into the algorithm can tune the efficiency according to current network metrics. Furthermore, it also can control the network load induced by the algorithm.

Then, two echo algorithms based on the PI algorithm have been presented. The time complexity of both algorithms is half of those for topology based models. The reduction comes from the different construction of the information collection phase. Echo messages can be send directly to the initiator of the echo algorithm. But this introduces a termination problem. The initiator must be able to determine when the last echo message has arrived. We proposed two solutions which fit well in the presented algorithms.

Finally, we discussed two election algorithms both based on the echo algorithms. The first used message extinction and the second was based on repeated executions of echo algorithms. The advantage of the latter one was that concurrent traversals of leader candidates are not wasted.

7 References

[1] D. I. Bevan: Distributed Garbage Collection Using Reference Counting. In *Proc. PARLE - Parallel Architectures and Languages Europe*, J. W. de Bakker, et al. (eds), 1987, pp. 176-187

[2] D. I. Bevan: An Efficient Reference Counting Solution to the Distributed Garbage Collection Problem. In *Parallel Computing 9*, 1989, pp. 179-192

[3] G. Bracha, S. Toueg: Distributed deadlock detection. In *Distributed Computing*, vol. 2, 1987, pp. 127-138

[4] E.J.H. Chang: Echo algorithms: Depth parallel operations on general graphs. In *IEEE Trans. on Software Engineering*, vol. SE-8, no. 4, 1982, pp. 391-401

[5] T.Eirich: *Fine-grained checkpointing in distributed object systems*. TR-I4-94-12, Univ. of Erlangen-Nürnberg, IMMD IV, June 1994

[6] D. Kumar: A class of termination detection algorithms for distributed computations. In *Proc. of 5th Conf. on Foundations of Software Technology and Theoretical Computer Science*, N. Maheshwari (ed), LNCS 206, Springer Verlag, 1985, pp. 13-22

[7] F. Mattern; Algorithms for distributed termination detection. In *Distrib. Comput.*, Vol.2, 1987, pp. 161-175

[8] F.Mattern: *Verteilte Basisalogorithmen*, Springer-Verlag, 1989

[9] F. Mattern: Message Complexity of Simple Ring-Based Election Algorithms - An Empirical Analysis. In *9th Int. Conf. on Distributed Computing Systems*, IEEE Computer Soc. Pr., 1989, pp. 94-100

[10] F.Mattern: Asynchronous Distributed Termination – Parallel and Symmetric Solutions with Echo Algorithms. In *Algorithmica*, Vol.5, No.3, Springer Verlag New York, 1990, pp.325-340

[11] D. Peleg: Time-Optimal Leader Election in General Networks. In *Journal of Parallel and Distributed Computing*, January 1990, pp. 96-99

[12] A. Segall: Distributed Network Protocols. In *IEEE Trans. Inform. Theory*, Vol.22, 1983, pp.23-35