On Fractional Dynamic Faults with Threshold*

Stefan Dobrev[1], Rastislav Královič[2], Richard Královič[2], and Nicola Santoro[3]

[1] School of Information Technology and Engineering, University of Ottawa, Ottawa, K1N 6N5, Canada.
[2] Dept. of Computer Science, Comenius University, Mlynská dolina, 84248 Bratislava, Slovakia.
[3] School of Computer Science, Carleton University, Ottawa, K1S 5B6, Canada.

Abstract. Unlike *localized* communication failures that occur on a fixed (although a priori unknown) set of links, *dynamic* faults can occur on any link. Known also as mobile or ubiquitous faults, their presence makes many tasks difficult if not impossible to solve even in synchronous systems. Their analysis and the development of fault-tolerant protocols have been carried out under two main models. In this paper, we introduce a new model for dynamic faults in synchronous distributed systems. This model includes as special cases the existing settings studied in the literature. We focus on the hardest setting of this model, called *simple threshold*, where to be guaranteed that at least one message is delivered in a time step, the total number of transmitted messages in that time step must reach a threshold $T \leq c(G)$, where $c(G)$ is the edge connectivity of the network. We investigate the problem of *broadcasting* under this model for the worst threshold $T = c(G)$ in several classes of graphs as well as in arbitrary networks. We design solution protocols, proving that broadcast is possible even in this harsh environment. We analyze the time costs showing that broadcast can be completed in (low) polynomial time for several networks including rings (with or without knowledge of n), complete graphs (with or without chordal sense of direction), hypercubes (with or without orientation), and constant-degree networks (with or without full topological knowledge).

1 Introduction

1.1 Dynamic Faults

In a message-passing distributed computing environment, entities communicate by sending messages to their neighbors in the underlying communication network. However, during transmission, messages might be lost.

The presence of communication faults renders the solution of problems difficult if not impossible. In particular, in *asynchronous* settings, the mere possibility of faults renders unsolvable almost all non trivial tasks, even if the faults are *localized* to (i.e., restricted to occur on the links of) a single entity [11]. Due to this

* Partially supported by VEGA 1/3106/06, NSERC, and TECSIS Co.

inherent difficulty connected with asynchrony, the focus is on *synchronous* environments, both from the point of view of theoretical investigation, and industrial application (e.g. communication protocols for wireless networks).

Since synchrony provides a perfect omission detection mechanism [2], localized faults are easily dealt with in these systems; indeed, any number of faulty links can be tolerated provided they do not disconnect the network. The immediate question is then whether synchrony allows to tolerate also *dynamic* communication faults; that is, faults that are not restricted to a fixed (but a priori unknown) set of links, but can occur between any two neighbors [17]. These types of faults, also called *mobile* or *ubiquitous*, are clearly more difficult to handle.

In this regard, the investigations have focused mostly on the basic problem of *broadcast*: an entity has some information that must communicate to all other entities in the network. Indeed, the ability or impossibility of performing this task has immediate consequence for many other tasks. Not surprisingly, a large research effort has been on the analysis of broadcasting in the presence of dynamic communication faults.

Clearly no computation, including broadcast, is possible if the amount of faults that can occur per time unit and the modality of their occurrence is unrestricted. The research quest has thus been on determining under what conditions on the faults non-trivial computations can be performed in spite of those faults. Constructively, the effort is on designing protocols that can correctly solve a problem provided some restrictions on the occurrence of faults hold.

A first large group of investigations have considered the so-called *cumulative* model; that is, there is a (known) limit L on the number[4] of messages that can be lost at each time unit. If the limit is less than the edge connectivity of the network, $L < c(G)$, then broadcast can be achieved by simply flooding and repeating transmissions for an appropriate amount of time. The research has been on determining what is the smallest amount of time in general or for specific topologies [3–6, 8–10, 12, 14, 15], as well as on how to use broadcast for efficiently computing functions and achieving other tasks [7, 18, 19].

The advantage of the cumulative model is that solutions designed for it are L-tolerant; that is they tolerate up to L communication faults per time units. The disadvantage of this approach is that it neglects the fact that in real systems the number of lost messages is generally a function of the number of all message transitions. This feature leads to an anomaly of the cumulative model, where solutions that flood the network with large amounts of messages tend to work well, while their behavior in real faulty environments is often quite poor.

A setting that takes into account the interplay between amount of transmissions and number of losses is the *probabilistic* model: there is no a priori upper bound on the total number of faults per time unit, but each transmission has a (known) probability $p < 1$ to fail. The investigations in this model have focused on designing broadcasting algorithms with low time complexity and high proba-

[4] since the faults are dynamic, no restriction is clearly posed on their location

bility of success [1, 16]. The drawback of this model is that the solutions derived for it have no deterministic guarantee of correctness.

The drawbacks of these two models have been the motivation behind the introduction of the so called *fractional* model, a deterministic setting that explicitly takes into account the interaction between the number of omissions and the number of messages. In the fractional model, the amount of faults that can occur at time t is not fixed but rather a linear fraction $\lfloor \alpha\ m_t \rfloor$ of the total number m_t of messages sent at time t, where $0 \leq \alpha < 1$ is a (known) constant. The advantage of the fractional model is that solutions designed for it tolerate the loss of up to a fraction of all transmitted messages [13]. The anomaly of the fractional model is that, in this setting, transmitting a single message per communication round ensures its delivery; thus, the model leads to very counterintuitive algorithms which do not behave well in real faulty environments.

Summarizing, to obtain optimal solutions, message redundancy must be avoided in the fractional model, while massive redundancy of messages must be used in the cumulative model; in real systems, both solutions might not fare well. In many ways, the two models are opposite extremes. The lesson to be learned from their anomalies is that on one hand there is need to use redundant communication, but on the other hand brute force algorithms based on repeatedly flooding the network do not necessarily solve the problem.

In this paper we propose a deterministic model that combines the cumulative and fractional models in a way that might better reflect reality. This model is actually more general, in that it includes those models as particular, extreme cases. It also defines a spectrum of settings that avoid the anomalies of both extreme cases.

1.2 Fractional Threshold and Broadcast

The failure model we consider, and that we shall call *fractional dynamic faults with threshold* or simply *fractional threshold* model, is a combination of the fractional model with the cumulative model. Both fractional and cumulative models can be described as a game between the algorithm and an adversary: in a time step t, the algorithm tries to send m_t messages, and the adversary may destroy up to $F(m_t)$ of them. While in the cumulative model, the *dependency function* F is a constant function, in fractional model $F(m_t) = \lfloor \alpha m_t \rfloor$. The dependency function of the fractional threshold model is the maximum of those two:

$$F(m_t) = \max\{T - 1, \lfloor \alpha\ m_t \rfloor\}$$

where $T \leq c(G)$ is a constant at most equal to the edge connectivity of the graph, and α is a constant $0 \leq \alpha < 1$. The name "fractional threshold" comes from the fact that it is the fractional model with the additional requirement that the algorithm has to send at least T messages in a time step t in order to have any guarantees about the number of faults.

Note that both the cumulative and the fractional models are particular, extreme instances of this model. In fact, $\alpha = 0$ yields the *cumulative* setting: at

most $T-1$ faults occur at each time step. On the other hand, the case $T=1$ results in the *fractional* setting. In between, it defines a spectrum of new settings never explored before, which avoid the anomalies of both extreme cases.

From this spectrum, the settings that give the maximum power to the adversary, thus making the broadcasting most difficult, are what will be called a *simple threshold* model defined by $T = c(G)$ and $\alpha = 1 - \varepsilon$ with ε infinitely close to 0. In this model, if less than $c(G)$ messages are sent in a step, none of them is guaranteed to arrive (i.e., they all may be lost); on the other hand, if at least $c(G)$ messages are transmitted, at least one message is guaranteed to be delivered.

In this paper we start the analysis of fault-tolerant computing in the fractional threshold model, focusing on the simple threshold setting. In this draconian setting the tricks from cumulative and fractional models fail: if the algorithm uses simple flooding the adversary can deliver only one message between the same pair of vertices over and over. If, on the other hand, the algorithm sends too few messages, they all may be lost.

1.3 The Results

The network is represented by a simple graph G of n vertices representing the entities and m edges representing the links. The vertices are *anonymous*, i.e. they are without distinct IDs. The communication is by means of synchronous message passing (i.e. in globally synchronized communication rounds), local computation is performed between the communication rounds and is considered instantaneous. The communication failures are dynamic omissions in the simple threshold model.

We consider the problem of *broadcasting*: At the beginning, there is a single initiator v containing the information to be disseminated. Upon algorithm termination, all entities must have learned this information. We consider *explicit* termination, i.e. when the algorithm terminates at an entity, it will not process any more messages (and, in fact, no messages should be arriving anyway).

The complexity measure of interest is *time* (i.e., number of communication rounds). We consider various levels of topological knowledge about the network (knowing network size n, being aware of the network topology, having *Sense of Direction* or having full topological knowledge).

In this paper, we focus on the hardest setting, the *simple threshold*, where to be guaranteed that at least one message is delivered in a time step, the total amount of transmitted messages in that time step must be at least $c(G)$, i.e. the edge connectivity of the network.

By definition, it is sufficient to ensure that $c(G)$ or more messages are transmitted at each time unit to guarantee that at least one of these messages is delivered. The problem however is that an entity does not know which other entities are transmitting at the same time and in general does not know which of its neighbors has already received its messages. Indeed the problem, in spite of synchrony and of the simplicity of its statement, is not simple.

We investigate the problem of *broadcasting* under this model in several classes of graphs as well as in arbitrary networks. We design solution protocols, proving that broadcast is possible even under the worst threshold $c(G)$. We analyze the time costs showing the surprising result that broadcast can be completed in (low) polynomial time for several networks including rings (with or without knowledge of n), complete graphs (with or without chordal sense of direction), hypercubes (with or without orientation), and constant-degree networks (with or without full topological knowledge). In addition to the upper bounds, we also establish a lower bound in the case of complete graphs without sense of direction. The results are summarized in the Table 1. Due to space constraints some technical parts have been omitted.

Topology	Condition	Time complexity
ring	n not necessarily known	$\Theta(n)$
complete graph	with chordal sense of direction	$O(n^2)$
complete graph	unoriented	$\Omega(n^2)$, $O(n^3)$
hypercube	oriented	$O(n^2 \log n)$
hypercube	unoriented	$O(n^4 \log^2 n)$
arbitrary network	full topological knowledge	$O(2^{c(G)}nm)$
arbitrary network	no topological knowledge except $c(G)$, n, m	$O(2^{c(G)}m^2n)$

Table 1. Summary of results presented in this paper.

2 Ring

The ring is a 2-connected network, i.e. $T = c(G) = 2$. Hence, at least two messages must be sent in a round to ensure that not all of them are lost.

We first present the algorithm assuming the ring size n is known, and then show how it can be extended to the case n unknown.

At any moment of time, the vertices can be either *informed* or *uninformed*. Since the information is spreading from the single initiator vertex s, informed vertices form a connected component. The initiator splits this component into the left part and the right part. Each informed vertex v can easily determine whether it is on the left part or on the right part of the informed component – this information is delivered in the message that informs the vertex v.

Each informed vertex can be further classified as either *active* or *passive*. A vertex is active if and only if it has received a message from only one of its neighbor. A passive vertex has received a message from both neighbors. This implies that, as long as the broadcast has not yet finished, there is at least one active vertex in both left and right part of the informed component (the left-most and the right-most informed vertices must be active; note, however, that also the intermediate vertices might be active).

The computation consists of $n-1$ phases, with each phase taking four communication rounds. The goal of a phase is to ensure that at least one active vertex becomes passive.

Each phase consists of the following four steps:

1. Each active vertex sends a message to its possibly uninformed neighbor.
2. Each active vertex in the right part sends a message to its possibly uninformed neighbor. Each vertex in the left part that received a message in step 1 replies to this message.
3. Same as step 2, but left and right parts are reversed.
4. Each vertex that received a non-reply message in steps 1–3 replies to that message.

To avoid corner cases at the initiator of the broadcast, the initiator is split into two virtual vertices such that each of them starts in active state (i.e. the initiator acts as if it belongs both to the left and to the right part).

Lemma 1. *At least one reply message passes during the phase.*

Initially, there are two active (virtual) vertices (the left- and right- part of the initiator). Lemma 1 ensures that during each of the subsequent phases, at least one previously active vertex becomes passive. Since passive vertices never become active again, it follows that after at most $n-1$ phases, there are $n-1$ passive vertices. Once there are $n-1$ passive vertices, the remaining two must be informed (both are neighbors of a passive vertex), i.e. $n-1$ phases are sufficient to complete the broadcast.

Note also that the algorithm does not require distinct IDs or ring orientation (it can compute them, though, as it is initiated by a single vertex).

Theorem 1. *There is $4(n-1)$-time fault-tolerant broadcasting algorithm for (anonymous, unoriented) rings of known size.*

If n is unknown, the above algorithm cannot be directly used, as it does not know when to terminate. This is not a serious obstacle, though. Assume that the algorithm is run without a time bound, and each discover message also contains a counter how far is the vertex from the initiator. After at most n phases there will be a vertex v that has received discover messages from both directions. From the counters in those messages v can compute the ring size n. In the second part of the algorithm v broadcasts n (and the time since the start of the second broadcast) using the algorithm for known n; when that broadcast is finished, the whole algorithm terminates. In order to make this work, we have to ensure that there is no interaction between the execution of the first broadcast and the second broadcast. That can be easily accomplished by scheduling the communication steps of the first broadcast in odd time slots and the second broadcast in even time slots.

Theorem 2. *There is an $O(n)$-time fault-tolerant broadcasting algorithm for (anonymous, unoriented) rings of unknown size.*

3 Complete Graphs

As the connectivity of complete graphs is $n-1$, we assume that least $n-1$ messages must be sent to ensure that at least one passes through.

3.1 Complete Graphs with Chordal Sense of Direction

Chordal Sense of Direction in a complete graphs means that vertices are numbered $0, 1, \ldots, n-1$ and the link from a vertex u to a vertex v is labelled $v - u \bmod n$. [5]

The algorithm consists of two parts. The purpose of the first part is to make sure that at least $\lceil n/2 \rceil$ vertices are informed; the second part uses these vertices to inform the remaining ones. The algorithm is executed by informed vertices. Each message contains a time counter, so a newly informed vertex can learn the time and join the computation at the right place.

The first part of the algorithm consists of phases $0, 1, \ldots, \lceil n/2 \rceil - 2$. During phase 0 the initiator sends messages to all its neighbors. The goal of phase k is to ensure that there are at least $k+1$ informed vertices distinct from the initiator; this ensures that after the first part, there are at least $\lceil n/2 \rceil$ informed vertices.

Consider a phase k and suppose that there are exactly k informed vertices distinct from the initiator at the beginning of phase k. Let $d = \left\lfloor \frac{n-1}{k+1} \right\rfloor$, and consider $k+1$ disjoint intervals $I_0, \ldots, I_k$ each of size d, consisting of non-initiator vertices. The phase will consist of $k+1$ steps. The idea is that during the i-th step, the informed vertices (including initiator) try to inform an additional vertex in the interval I_i by sending messages to all vertices in I_i. If I_i does not contain any informed vertices, and at least one message is delivered, then a new vertex must be informed. The problem is, however, that only $d(k+1)$ messages are sent, which may not be sufficient to guarantee delivery. To remedy this, the i-th step will span over d rounds. In a j-th round, all informed vertices send messages to all vertices in I_i and to the j-th vertex of $I_{i \oplus 1}$ (the addition is taken modulo $k+1$). Now, in each step there are $(k+1)(d+1)$ messages sent, so at least one must be delivered. Hence we can argue that, during phase k, a new vertex is informed if there is an interval I_i that does not contain any informed vertices, followed by interval $I_{i \oplus 1}$ that contains at least one non-informed vertex. However, the existence of such I_i follows readily from the fact that there are only k informed vertices distinct from initiator and $d \geq 2$.

Lemma 2. *After phase k there are at least $k+1$ informed vertices distinct from the initiator.*

[5] Strictly speaking, the vertices do not necessarily need to know their ID, the link labels are sufficient: The initiator may assume ID 0 and each message will also carry the link label it travels on and the ID of the sender, allowing the receiver to compute its ID.

Each phase k consists of $k+1$ steps with d rounds each, therefore every phase takes $O(n)$ time steps. Since there are $O(n)$ phases, the first part of the algorithm finishes in $O(n^2)$ time.

The second part of the algorithm starts with at least $\lceil n/2 \rceil$ informed vertices and informs all remaining ones. The algorithm is as follows: consider all pairs $[i,j]$ such that $1 \leq i,j \leq n-1$, sorted in lexicographic order. In each step, all informed vertices consider one pair and send messages to vertices i and j. Since at least $2\lceil n/2 \rceil \geq n-1$ messages are sent, at least one of them is delivered. This ensures that a new vertex is informed whenever both i and j were uninformed. In this manner, all but one vertex can be informed (at any moment the two smallest unexplored vertices form a pair that has not been considered yet).

To inform the last vertex, all $n-1$ informed vertices send in turn messages to vertices $1, 2, \ldots, n-1$.

Theorem 3. *There is a $O(n^2)$ time fault-tolerant broadcasting algorithm for complete graphs with chordal sense of direction.*

Proof. The first part consist of $\lceil n/2 \rceil - 2$ phases, with each phase taking $O(n)$ steps. The second part consists of $n(n-1)/2$ steps and informing the last vertex takes $n-1$ steps.

Note that the algorithm did not exploit all properties of the chordal sense of direction, it is sufficient for the informed vertices to agree on the IDs of the vertices, and to be able to determine the ID of the vertex on the other side of a link. Therefore, we get:

Corollary 1. *There is a $O(n^2)$ time broadcasting algorithm for complete graphs with neighboring (Abelian group based) sense of direction.*

3.2 Unoriented Complete Graphs

The algorithm in the previous section strongly relied on the fact that the vertices know the IDs of the vertices on the other side of the links. In this section, we use very different techniques to develop an algorithm that works for unoriented complete graphs (i.e. the only structural information available is the knowledge that the graph is complete; of course, local orientation – being able to distinguish incident ports – is also required).

We will view the flow of messages as tokens traveling through the network (and possibly spawning new tokens). A message (token) arriving to a vertex may cause the vertex to transmit some messages (either immediately, or in some of the subsequent steps). We will view those new messages as child tokens of the parent token. This means the tokens form a tree structure, and each token can be assigned unique identifier (corresponding to a path in the tree structure). Note that each vertex can also be given unique identifier (the ID of the token that first informed it). Each token carries all information about itself and its ancestors (i.e. IDs of its ancestors, traversed vertices and traversed ports).

Each token may be of two types: green and red. The intuition is that a token is green if it is "exploring", i.e. trying to traverse a port that has never been explored by its ancestors. When every port has been explored by the token's ancestors, the broadcast is finished, and no new tokens are sent. Ideally, if a token arrives to a vertex v, it would be spawned as a green token along all links that have not yet been explored by its ancestors. However, there is usually not enough unexplored ports in v[6]. In this case red tokens are sent along some already explored links. The meaning is that a red token carries a "request for help" to already explored vertices that are not yet engaged in helping. This request triggers new tokens to be sent from those vertices, and eventually a situation occurs when only green tokens are sent and at least one of them is delivered.

Let T be any token. The *green ancestor* of T is the closest green ancestor of T, if T is red, and T itself, if T is green. The *red tail* of token T is the path (sequence of tokens) between the green ancestor of T and T itself. Note that all tokens on the red tail are red except the first one.

We present a fault-tolerant broadcast algorithm that satisfies the following invariants:

- *I1* Let T be a token that is sent over an oriented edge $\langle a, b\rangle$. If T is green, then it holds that no ancestor of T has been sent over $\langle a, b\rangle$. Conversely, if T is red, there exists some ancestor of T that has been sent over $\langle a, b\rangle$.
- *I2* Let T be a red token. Then the red tail of T contains at most n vertices.
- *I3* Let T be a red token. Then T is sent exactly one round later than the parent of T.
- *I4* Let T be a green token. Then T is sent at most $n+1$ rounds later than the last green ancestor of the parent of T.
- *I5* Let T be a green token delivered in round t. If the broadcast is not finished yet, at least one green token is delivered in some of the rounds $t+1, \ldots t+n$.

These invariants imply that the broadcast completes in $O(n^3)$ time: the invariant *I5* ensures that the algorithm can not stop before the broadcast is finished. Consider a path from root to a leaf in the tree of tokens. Invariant *I1* ensures that the leaf is green and that there are at most $O(n^2)$ green tokens on this path. Invariant *I4* implies that there are at most $n+1$ consecutive red tokens on the path. Hence the overall time of the broadcasting algorithm is $O(n^3)$.

The algorithm works as follows. In the first round of the algorithm, the initiator sends green tokens through all its ports. All these tokens are children of some virtual root token. In each subsequent round t, each vertex gathers all received tokens in this round and processes them in parallel using procedure PROCESS described in Algorithm 1.

[6] recall that at least $n-1$ messages must be sent in every step to make sure that at least one is delivered

Algorithm 1 Complete graphs without sense of direction

```
 1: procedure PROCESS(T)    // process token T
 2:     Let P be the set of all ports
 3:     Let A be the set of ports that have never been traversed by any ancestor of T
 4:     If A = ∅, the broadcast is finished.
 5:     Let S be the set of vertices acting as a source of a red token in the red tail of T.
 6:     Let B ⊆ P − A be the set of ports that lead to a vertex in S.
 7:     Let C = P − (A ∪ B)
    // Note that since only ports already traversed by (an ancestor of) T
    // are considered, the vertex processing T can indeed compute B and C.

 8:     for the first round of processing T do
 9:         Send new red tokens with parent T to all ports in C
10:         Send new green tokens with parent T to all ports in A
11:     end for

12:     Let l be the length of the red tail of T.
13:     for subsequent n − l rounds of processing T do
14:         Send new green token with parent T to all ports in A
15:     end for
16: end procedure
```

If some processor should send more than one token through a port in one round, it (arbitrarily) chooses single one of them to send and discards the remaining ones.

Lemma 3. *The presented algorithm satisfies invariants I1, I2, . . . , I5.*

Combining Lemma 3 with the discussion about the invariants we get

Theorem 4. *There exist a $O(n^3)$ fault-tolerant broadcasting algorithm for unoriented complete networks.*

3.3 Lower Bound for Unoriented Complete Networks

The $O(n)$ algorithm for rings is obviously asymptotically optimal. An interesting question is: How far from optimal are our algorithms for oriented and unoriented complete networks? In this section we show that

Theorem 5. *Any fault-tolerant broadcasting algorithm on unoriented complete networks must spend $\Omega(n^2)$ time.*

Proof. In the course of the computation there are two kinds of ports: the ports that have never been traversed by any message in any direction are called "*free*", the ports that are not free are called "*bound*". The lower bound proof is based on the following simple fact:

Let p be a free port of vertex u in time t. Let v be any vertex such that no bound port of u leads to v. Then it is possible that port p leads to vertex v.

Indeed, if p would lead to v, the first t rounds of computation would be the same. Hence, the computation can be viewed as a game of two players: the algorithm chooses a set of ports through which messages are to be sent. The adversary chooses one port through which the requested message passes. If this port is free, it chooses also the vertex to which this port will be bound.

We show now that it is possible for the adversary to keep the vertex n uninformed for $\frac{(n-1)(n-2)}{2} = \Omega(n^2)$ communication rounds. The idea is that some message has to traverse through all edges between vertices $1 \dots n-1$ before any message arrives to the vertex n.

Consider the time step $i < \frac{(n-1)(n-2)}{2}$ and assume that the vertex n is not informed yet. There exist at most $2i$ bound ports, since in each time step at most one edge, i.e two ports are bounded. This means that at least $(n-1)(n-1)-2i \geq n$ ports of vertices $1 \dots n-1$ are free.

The following cases can occur:

1. The algorithm sends some message through some bound port. The adversary passes this message, hence the vertex n stays uninformed.
2. The algorithm sends messages only through free ports.
   (a) The algorithm does not send messages from all vertices $1 \dots n-1$. Then there have to be at least 2 messages sent from one vertex. The adversary delivers one of these messages and binds the corresponding port to any vertex other than n. (Since there are at least two free ports, it is possible for the adversary to do so.)
   (b) The algorithm sends messages from all vertices $1 \dots n-1$. Since at least n ports of vertices $1 \dots n-1$ are free, at least one vertex w from $1 \dots n-1$, has 2 free ports. The adversary delivers the message sent from w, and binds corresponding port to any vertex other than n. (Again, since there are at least two free ports, it is possible for the adversary to do so.)

Hence it is possible for the adversary to keep the vertex n uninformed for the first $\Omega(n^2)$ time steps.

Now assume a stronger computation model: each vertex immediately learns for any message it has sent whether this message has been delivered or not. It is interesting to note that our lower bound is valid also in this model. Furthermore, it is easy to see that the lower bound is tight in this model.

4 Arbitrary k-connected graphs

In this section we consider k-edge-connected graphs and we assume the threshold is k, i.e. at least k messages must be sent to ensure that a message is delivered.

4.1 With full topological knowledge

The algorithm runs in $n-1$ phases. Each phase has an initiator vertex u (informed) and a destination vertex v (uninformed), with the source s being the initiator of the first phase. The goal of a phase is to inform vertex v, which then becomes the initiator of the next phase; the process is repeated until all vertices are informed.

The basic idea is a generalization of the idea from rings. The ring algorithm tried to "push" the information simultaneously along the left and right part of the ring. Here, the initiator u chooses k edge-disjoint paths[7] $\mathcal{P} = \{P_1 \dots P_k\}$ from itself to v and then pushes the information through all the paths simultaneously. Let $P_i = (u_0 = u, u_1, \dots u_{l_i} = v)$; consider an oriented edge $e = \langle u_j, u_{j+1} \rangle$. This edge can be either *sleeping*, *active* or *passive*:

1. The edge e is *passive* if and only if a message has been received over both e and the edge opposite to e, i.e. $\langle u_{j+1}, u_j \rangle$.
2. The edge e is *active* if and only if it is not passive and a message has been received over the edge $\langle u_{j-1}, u_j \rangle$. In case $j = 0$ the edge e is active whenever it is not passive.
3. The edge e is *sleeping* if and only if it is not active nor passive.

One phase consists of several rounds, each round spanning over many communication steps. The goal of one round is to ensure that a progress over at least one edge has been made: at least one active edge becomes passive, at least one sleeping edge becomes active or the vertex v becomes informed.

The procedure ROUND() defined in Algorithm 2 is the core of the algorithm; it is performed in each round by every vertex $w \in \mathcal{P}$.

It is easy to see that the uninformed vertices never send any messages and that at any time each vertex can determine all active edges incident to it. Synchronous communication and full topological knowledge ensure that all procedures (phases/rounds/subrounds) are started and executed simultaneously by all participating vertices.

Lemma 4. *During one round at least one active edge becomes passive, or a sleeping edge becomes active, or v is informed.*

Proof. By contradiction. Assume the contrary, we show that in such case, at the beginning of the i-th subround there will be at least i paths $\mathcal{P}' \subseteq \mathcal{P}$ such that on any path $P_j \in \mathcal{P}'$ there is an edge through which an activating message has been delivered in the current round. This would mean that in the k-th subround there are at least k deactivating messages sent and therefore at least one of them will be delivered and an active edge will become passive, a contradiction.

We prove that above statement about subrounds by induction on i. The statement trivially holds for $i = 0$, as there is nothing to prove. Assume (by induction hypothesis) that at the beginning of the i-th subround there are exactly

[7] since the graph is k-edge-connected and the vertices have full topological knowledge, the initiator can always find these paths

Algorithm 2 k-connected graphs

```
1: procedure ROUND(vertex w)
2:     Let A be the set of active edges incident to w at the beginning of the round
3:     for i:=0 to k do    // One subround:
4:         for B ⊆ {1...k} such that |B| = i do    // one iteration per time step
5:             Let C be the set of edges incident to w via which an activating message
6:             has been received in the current round    // not in the current time step
7:             for e ∈ C do
8:                 send deactivating message through e    // all in one time step
9:             end for
10:            for e ∈ A such that e ∈ P_z ∧ z ∉ B do
11:                send activating message through e  // all in the same time step as
   in 8
12:            end for
13:        end for
14:    end for
15: end procedure
```

i paths $\mathcal{P}'$ with an edge over which an activating message has been delivered in the current round(if there are more, the hypothesis is already true for $i+1$). From the definition of an active edge and from construction it follows that unless the vertex v is informed, there is at least one active edge on each path P_j. Let us focus on the time step in the i-th subround when B contains exactly the numbers of paths from $\mathcal{P}'$ (i.e. $B = \{j | P_j \in \mathcal{P}'\}$). In this time step, at least $k-i$ activating and at least i deactivating messages are sent, therefore at least one of them must be delivered. As no activating message is sent over an edge $e \in \mathcal{P}'$ and no deactivating message is delivered (by assumption that no active edge becomes passive), an activating message must be delivered on a path not in $\mathcal{P}'$. Hence, the invariant is ensured for the subround $i+1$, too.

Theorem 6. *There is a fault-tolerant broadcasting algorithm on k-connected graphs with full topology knowledge that uses $O(2^k nm)$ time, where n is the number of vertices and m is the number of edges in the graph.*

Proof. The correctness follows straightforwardly from construction and Lemma 4.

The time complexity of one round is 2^k, as it spends one time step for each subset of $\{1, 2, \ldots, k\}$. The number of rounds per phase is[8] $2m$, as all paths in $\mathcal{P}$ together cannot contain more than all m edges and each edge can change its state at most twice (from sleeping to active to passive). Finally, the number of phases is $n-1$ as $n-1$ vertices need to be informed. Multiplying we get $O(2^k mn)$.

Theorem 6 can be successfully applied to many commonly used interconnection topologies. However, better results can usually be obtained by carefully choosing the order in which the vertices should be informed, allowing for short

[8] some topology-specific optimization is possible here

paths in $\mathcal{P}$. One such example is oriented hypercubes (i.e. each link is marked by the dimension it lies in):

Theorem 7. *There is a fault-tolerant broadcasting algorithm for oriented d-dimensional hypercubes that uses $O(n^2 \log n)$ time, where $n = 2^d$ is the number of vertices of the hypercube.*

Proof. The basic idea is to use the algorithm for k-connected graphs, with the initiator of a phase choosing as the next vertex to inform its successor in (a fixed) Hamiltonian path of the hypercube.

The algorithm for one phase is the same as in the case of k-connected graphs with the following exception: it is possible to choose d edge-disjoint paths from vertex u to its neighbor vertex v such that each of these paths has length at most 3. This results in $\mathcal{P}$ containing only $O(d)$ edges instead of $O(n \log n)$, thus reducing the cost of one phase from $O(n^2 \log n)$ to $O(nd) = O(n \log n)$. The resulting time complexity is therefore $O(n^2 \log n)$.

4.2 Without topological knowledge

Finally, we show that the broadcasting on a k-connected graph with n vertices and m edges can be performed in time $O(2^k m^2 n)$ even in the case when the only known information about the graph are the values of n, m, and k. To achieve this, we combine the ideas used for complete graphs with those using full topology knowledge. In particular, the vertices accumulate topology information (using local identifiers) in a fashion similar to the algorithm for complete graphs. The algorithm works in phases, where each phase is performed within one informed component, and uses the topology knowledge of that component. However, since there may be many phases active at the same moment, great care must be given to avoid unwanted interference. The detailed result has been omitted due to space constraints.

Applying this result to the case of d-dimensional hypercube without sense of direction yields an algorithm that uses $O(n^4 \log^2 n)$ time.

5 Conclusions

We have introduced a new model for dynamic faults in synchronous distributed systems. This model includes as special cases the existing settings studied in the literature. We have focused on the *simple threshold* setting where, to be guaranteed that at least one message is delivered in a time step, the total amount of transmitted messages in that time step must be above the threshold T. We have investigated broadcasting in rings and complete graphs, as well as arbitrary networks, and we have designed solution protocols, proving that broadcast is possible also under the worst threshold (i.e., equal to the connectivity). The perhaps surprising result is that the time costs are (low) polynomial for several networks including rings, complete graphs, hypercubes, and constant-degree networks.

This investigation is the first step in the analysis of distributed computing in spite of fractional dynamic faults with threshold.

References

1. P. Berman, K. Diks, and A. Pelc, "Reliable broadcasting in logarithmic time with Byzantine link failures". *Journal of Algorithms*, 22 (2), 199–211, 1997.
2. T. Chandra, V. Hadzilacos, and S. Toueg, "The weakest failure detector for solving consensus". *Journal of ACM*, 43(4), 685–722, 1996.
3. B.S. Chlebus, K. Diks, and A. Pelc, "Broadcasting in synchronous networks with dynamic faults". *Networks* 27, 309–318, 1996.
4. G. De Marco and A. Rescigno, "Tighter time bounds on broadcasting in torus networks in presence of dynamic faults". *Parallel Processing Letters* 10 (1), 39–50, 2000.
5. G. De Marco and U. Vaccaro, "Broadcasting in hypercubes and star graphs with dynamic faults". *Information Processing Letters* 66, 309–318, 1998.
6. S. Dobrev, "Communication-efficient broadcasting in complete networks with dynamic faults". *Theory of Computing Systems* 36(6), 695–709, 2003.
7. S. Dobrev, "Computing input multiplicity in anonymous synchronous networks with dynamic faults". *Journal of Discrete Algorithms* 2, 425–438, 2004.
8. S. Dobrev and I. Vrťo, "Optimal broadcasting in hypercubes with dynamic faults". *Information Processing Letters* 71, 81–85, 1999.
9. S. Dobrev and I. Vrťo, "Optimal broadcasting in even tori with dynamic faults". *Parallel Processing Letters* 12, 17–22, 2002.
10. S. Dobrev and I. Vrťo, "Dynamic faults have small effect on broadcasting in hypercubes". *Discrete Applied Mathematics* 137(2), 155–158, 2004.
11. M. J. Fischer, N.A. Lynch, and M.S. Paterson, "Impossibility of distributed consensus with one faulty process", *Journal of the ACM* 32 (2), 1985.
12. P. Fraigniaud and C. Peyrat, "Broadcasting in a hypercube when some calls fail", *Information Processing Letters* 39, 115–119, 1991.
13. R. Královič, R. Královič, and P. Ružička, "Broadcasting with many faulty links". In *Proc. 10th Colloquium on Structural Information and Communication complexity* (SIROCCO'03), 211–222, 2003.
14. Z. Liptak and A. Nickelsen, "Broadcasting in complete networks with dynamic edge faults", In *Proc. 4th International Conference on Principles of Distributed Systems* (OPODIS 00), Paris, 123–142, 2000.
15. Tz. Ostromsky and Z. Nedev, "Broadcasting a Message in a Hypercube with Possible Link Faults". In *Parallel and Distributed Processing '91* (K. Boyanov, editor), Elsevier, 231–240, 1992.
16. A. Pelc and D. Peleg, "Feasibility and complexity of broadcasting with random transmission failures". In *Proc. 24th ACM Symposium on Principles of Distributed Computing* (PODC 05), 334–341, 2005.
17. N. Santoro and P. Widmayer, "Time is not a healer". In *Proc. 6th Ann. Symposium on Theoretical Aspects of Computer Science* (STACS 89), LNCS 349, 304–313, 1989.
18. N. Santoro and P. Widmayer, "Distributed function evaluation in the presence of transmission faults". In *Proc. International Symposium on Algorithms* (SIGAL 90), Tokyo, LNCS 450, 358–367, 1990.
19. N. Santoro and P. Widmayer, "Agreement in synchronous networks with ubiquitous faults". In *Theoretical Computer Science*, 2006, to appear; preliminary version in *Proc. 12th Colloquium on Structural Information and Communication Complexity* (SIROCCO'05), LNCS, 2005.