On Fractional Dynamic Faults with Threshold ⋆

Stefan Dobrev [a] Rastislav Královič [b,*] Richard Královič [b]
Nicola Santoro [c]

[a] *School of Information Technology and Engineering, University of Ottawa, Ottawa, K1N 6N5, Canada.*

[b] *Dept. of Computer Science, Comenius University, Mlynská dolina, 84248 Bratislava, Slovakia.*

[c] *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. 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).

⋆ Partially supported by APVT 0433-06, NSERC, and TECSIS Co.

* Corresponding author. Address: Department of Computer Science, Comenius University, Mlynská dolina, 84248 Bratislava, Slovakia.

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 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).

Clearly no computation 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. The approaches to describe the restrictions needed can be broadly divided into probabilistic and deterministic.

In 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 probability of success [2,18]. The drawback of this model is that the solutions derived for it have no deterministic guarantee of correctness.

In our work we follow the deterministic approach in which the worst combination of faults satisfying given restrictions is studied. The most basic model of deterministic faults is the model of static (or localized) faults, in which faults can occur only on a fixed (but a priori unknown) set of links [1,14]. This restriction is well suited for modeling permanent faults but is inappropriate to deal with transient faults, which are very common in practice. Indeed, most of the errors occurring in a network, from a packet loss in transmission medium to links turned off during network reconfiguration, can be viewed as transient failures: they are repaired after some time but their location can be arbitrary. Hence, a natural extension in the modeling of network failures is to bound not the location but the number of faults.

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.

A first large group of investigations have considered the so-called *cumulative* model; that is, there is a (known) limit L on the number 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 [3–6,8–10,13,16,17], as well as on how to use broadcast for efficiently computing functions and achieving other tasks [7,20,21].

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.

In order to eliminate this unwanted feature from the model, the so called *fractional* model has been introduced in [15]. This deterministic setting explicitly takes into account the interaction between the number of faults and the number of messages, bounding the amount of faults that can occur at time t not by a fixed constant but rather by a linear fraction $\lfloor \alpha\ m_t \rfloor$ of the total number m_t of messages sent at time t, for some (fixed, known) constant $0 \leq \alpha < 1$. The advantage of the fractional model is that solutions designed for it tolerate the loss of up to a fraction of all transmitted messages. 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 guarantee about the number of messages delivered.

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, the model offers 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 modelled 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. Each vertex can, however, locally distinguish its neighbors; the local identifiers are called *ports*. The communication is by means of synchronous message passing, i.e. there are globally synchronized communication steps, such that in each step, each vertex can send messages to any subset of its neighbors. Local computation is performed between the communication steps and is considered instantaneous. The communication failures are dynamic omissions. 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 number of transmitted messages in that time step must be at least $c(G)$, i.e. the edge connectivity of the network.

Given the non-acknowledged message-passing communication, and the fact that vertices can use only ports to distinguish neighbors, it is often convenient to consider a directed graph G' obtained from G by replacing each edge by two opposite arcs (i.e. oriented edges). We shall sometimes abuse the notation and use G and G' interchangably, and speak about sending a message along an arc.

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).

Although we consider anonymous networks, all our algortihms construct vertex identifiers. The initiator is used to break symmetry, and vertices are assigned names based on the messages they receive. One possible way to do so is to use the sequence of ports used to reach a vertex v from the initiator as the identifier of v. There are usually many such sequences, but it is possible to assign one of them as the identifier of v at the moment v is informed.

Once the vertex identifiers are available, it is easy to construct arc identifiers: an arc is described by the vertex it is outgoing from, and the corresponding port. In the sequel we shall sometimes abuse notation and identify vertices (arcs) with their identifiers.

The complexity measure of interest is *time* (i.e., number of communication steps). 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).

Recall that by the definition of the simple thereshold model, 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.

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 time step to ensure that at least one of them is delivered.

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.

The computation is organized in phases where each phase takes four communication rounds. In each phase, an informed vertex can be either *active* or *passive*. A vertex is active if and only if it has received, in previous phases, messages from only one of its neighbors. 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. The goal of a phase is to ensure that at least one active vertex becomes passive. Each phase consists of the following four steps, which are formally defined in Algorithm 1:

(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. [1]
(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 is delivered during each phase.*

Proof: Since there is at least one active vertex in both the left and the right part of the informed component, at least two messages are transmitted in step 1 and thus at least one of them is delivered. Assume that this message passed in the left part. Then there will be at least two messages transmitted in step 2 leading to different vertices (the reply message in the left part, and the discover message in the right part) and therefore at least one message passes in step 2 (or step 3 if the delivered message of step 1 was in the right part). If no reply message passed in step 2 (or 3), a discover message must have passed in the right (left) part. Therefore, at least two replies will be sent in step 4

[1] Note that passive vertices reply to such message, too.

Algorithm 1 Rings

```
1: state ∈ {uninformed, active, passive}
2: location ∈ {left, right}
3: dir-to-init ∈ Neigh

4: procedure PHASE
5:     time t:
6:     if state = active then
7:         send (discover, location) to opposite of dir-to-init
8:     end if

9:     time t + 1:
10:    if location = right then
11:        if state = active then
12:            send (discover, location) to opposite of dir-to-init
13:        end if
14:    else
15:        if received (discover) in time t then reply with (ack)
16:        end if
17:    end if

18:    time t + 2:
19:    if location = left then
20:        if state = active then
21:            send (discover, location) to opposite of dir-to-init
22:        end if
23:    else
24:        if received (discover) in time t then reply with (ack)
25:        end if
26:    end if

27:    time t + 3:
28:    if received (discover) in time t ... t + 2 then reply with (ack)
29:    end if

30:    if state = uninformed and received (discover, l) via dir in time t ... t+3
       then
31:        state := active;
32:        location := l;
33:        dir-to-init := dir;
34:    end if
35:    if state = active and received (ack) in time t ... t + 3 then
36:        state := passive;
37:    end if
38: end procedure
```

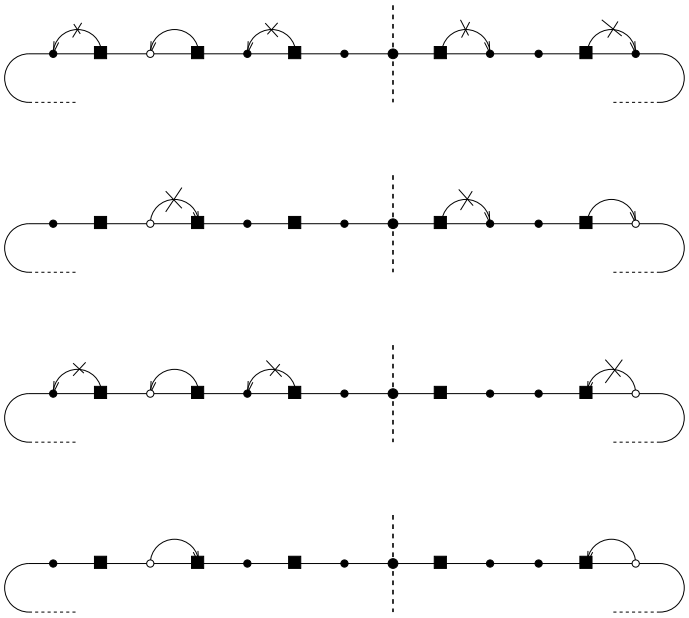

Fig. 1. A sample phase of Algorithm 1. The initially active vertices are squares. In the first step, the only surviving message is delivered to the white vertex on the left, which then replies in the next step. At the end of the phase, two white vertices are sending acknowledgement, so at least one acknowledgement is delivered during that phase.

and at least one of them will pass. □

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). Hence, we have proven the following theorem:

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

If n is unknown, Algorithm 1 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 steps and the second broadcast in even time steps. Furthermore, we need to consider a special case when there are *two* vertices that receive discover messages from both directions in one time step. In this case, the second broadcast has two initiators. However, it is not difficult to see that our broadcasting algorithm works in this case, too.

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 is a form of topological knowledge where vertices are numbered $0, 1, \ldots, n-1$ and the link from a vertex u to a vertex v is labelled $v-u \bmod n$.[2] An example of such labeling is depicted at Figure 2. With this sense of direction there is a natural notion of vertex identifiers: the initiator has label 0, and all other vertices are labelled by the edge label of the link by which they can be reached from the initiator. Note that each vertex, upon receiving a message, is able to compute its identifier even if the message was not sent from the initiator. Moreover, every vertex can compute the identifiers of all its neighbors.

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.

[2] 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.

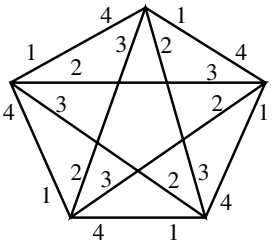

Fig. 2. Complete graph K_5 with chordal sense of direction.

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 a 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 $k+1$ rounds. The idea is that during the i-th round, 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 round will span over d steps. In a j-th step, 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.*

Proof: By induction on k. The statement holds for phase 0, as the initiator sends $n-1$ messages and at least one of them will be delivered. Consider now the situation after phase $k-1$ with exactly k informed vertices distinct from the initiator. We show that at least one vertex will be informed during phase k. Divide the interval $[1, n-1]$ into $k+2$ parts in such a way that each of the first $k+1$ parts is of size $d = \left\lfloor \frac{n-1}{k+1} \right\rfloor$. During step j of the round i of

Algorithm 2 Complete graphs with chordal sense of direction - Part I

```
1: the initiator sends message to all other vertices    // phase 0
2: for 1 ≤ k ≤ ⌈n/2⌉ − 2 do    // phase k
3:     d = ⌊(n−1)/(k+1)⌋
4:     for 0 ≤ i ≤ k do    // round i
5:         for 1 ≤ j ≤ d do    // step j
6:             all informed vertices send messages to vertices
7:             {di + 1, di + 2, ..., di + d, d((i + 1) mod (k + 1)) + j}
8:         end for
9:     end for
10: end for
```

the phase k all informed vertices (including the initiator) send messages to all vertices in part i and to the j-th vertex of part $(i+1) \mod (k+1)$. As $(k+1)(d+1) \geq n-1$ messages are sent, at least one of them is delivered. Hence it is sufficient to prove the following claim:

Claim 1 *There exists $0 \leq i \leq k$ such that there is no informed vertex in the i-th part and there is at least one non-informed vertex in the $(i \oplus 1)$-st part, where $\oplus$ is addition modulo $k+1$.*

Let there be exactly e parts $I_{z_1}, I_{z_2}, \ldots, I_{z_e}$ of size d containing no informed vertices. Assume that the claim does not hold, hence there are at least e disjoint parts $I_{z_1 \oplus 1}, I_{z_2 \oplus 1}, \ldots, I_{z_e \oplus 1}$ of size d containing only informed vertices. Since each of the remaining $k+1-2e$ parts contains at least one informed vertex, there are at least $ed+k+1-2e$ informed vertices. So it must hold $ed+k+1-2e \leq k$ which is equivalent to $e(d-2)+1 \leq 0$. As $e \geq 0$, this yields $d = \left\lfloor \frac{n-1}{k+1} \right\rfloor < 2$, and hence $\frac{n-1}{k+1} < 2$. However, this contradicts to $k \leq \lceil \frac{n}{2} \rceil - 2$. □

Each phase k consists of $k+1$ rounds with d steps 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 t, all informed vertices consider the t-th pair $[i,j]$ 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 uninformed 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 an $O(n^2)$ 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 $O(n)$ time 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. These properties are fullfilled by the definition of a weak sense of direction from [12]. Without diving into the details we mention that a local edge-labelling is a weak sense of direction, if it admits so called coding function: a function that, given a sequence of edge labels, computes a local name in such a way, that for any vertex v, and any two paths starting from v, these two paths lead to the same vertex exactly if the coding function returns the same local name. It is not difficult to see that any weak sense of direction enables the algorithm to compute appropriate vertex identifiers. Therefore, we get:

Corollary 1 *There is a $O(n^2)$ time broadcasting algorithm for complete graphs with weak 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).

The main idea of the algorithm is that of exploring an area. Informally, an area is a set of arcs that are known to have already delivered the message. Indeed, once the area is large enough, every vertex must have received the message. More formally, the knowledge of an area is described by a set of triples of the form (a, p, b), meaning that the port p of vertex a delivered a message to vertex b. This knowledge is stored in the vertices of the communication graph; the knowledge of an informed vertex u (i.e. the set of corresponding triples) is denoted as $K(u)$. Throughout the algorithm, the knowledge of the sending

vertex v will always be piggybacked on every message sent by v. This means that if u and v are vertices such that at step t a message passed from u to v via the port p of u, and $K(u)$ and $K(v)$ is their knowledge at time t, then the knowledge of v at time $t+1$ is a superset of $K(u) \cup K(v) \cup \{(u, p, v)\}$.

An informal view of the algorithm is as follows (see Fig. 3): once a message arrives to a vertex v, v tries to expand the knowledge of the area by sending the message along "exploring" arcs (i.e. arcs that are not in the known area). Although the knowledge of v does not change in case of successful delivery, the knowledge of the destination vertex will contain the full knowledge of v plus the used exploring arc. Hence, the goal of the algorithm is to ensure delivery of the message along the exploring arcs.

To do so, v must make sure that at least $n-1$ messages are sent during each step. Hence, v might be forced to send also some messages along arcs that are in the known area. These messages are "helper" messages: upon the receipt of a helper message, a vertex v' sends messages along its exploring arcs, and may send additional helper messages. The algorithm is constructed in such a way that after the helper messages have been sent, the vertex continues to send the exploring messages for a certain number of steps. The timig is chosen in such a way that, finally, only exploring messages are being sent which ensures progress.

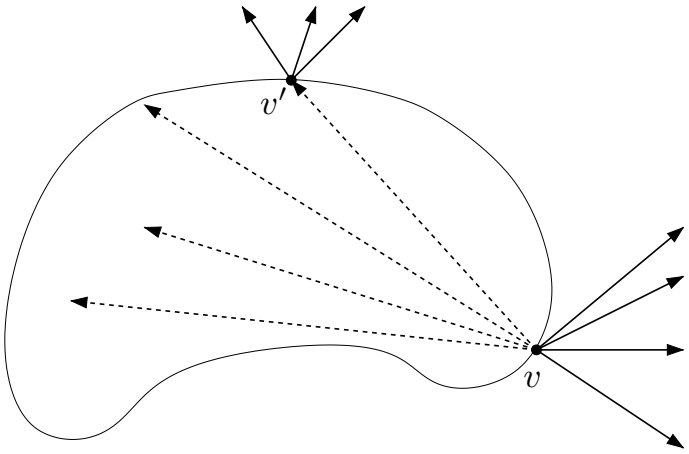

Fig. 3. Main idea of the algorithm. Vertex v tries to expand the current area by sending exploring messages. However, in order to ensure that enough messages are sent, it sends also helper messages along arcs already included in the current area. If such helper message arrives to a vertex v', it starts sending exploring messages to help v reach the required number of messages. If the number is still not sufficient, v' may decide to send other helper messages.

This simple approach is complicated by the fact that in a given step there may be many independent initiators having (possibly intersecting) areas. Hence, care must be given to ensure that the computations of these multiple initiators do not collide. To remedy this situation, we introduce the "family tree" of

messages. In our algorithm, a message is sent only as a response to another (exploring or helper) message (with the obvious exception of the first step). Hence, there is a natural notion of an ancestor of a message. An example of such message-tree is given on Figure 4.

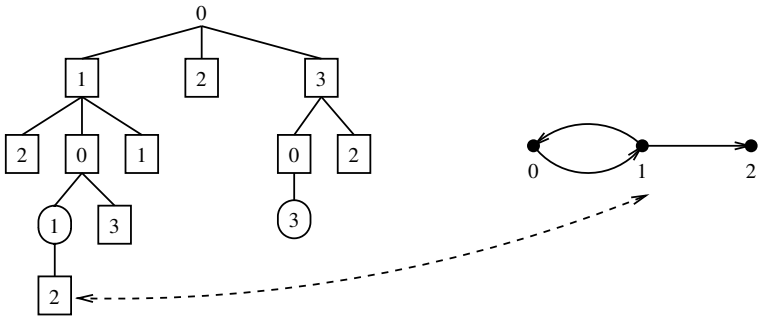

Fig. 4. *Left part:* An example of a message-tree. Vertices are messages, and edges correspond to the ancestry relation. Messages can be either exploring (squares) or helper (circles). Numbers inside the vertices denote the nodes processing corresponding messages. *Right part:* The knowledge carried by the marked message.

Since a message always carries all available area knowledge, it is clear that when a particular message is sent, all arcs traversed by some ancestor of the message in the message-tree are in the known area.

Let T be any message. The *exploring ancestor* of a message T is the closest ancestor of T which is an exploring message.[3] The *helper tail* of message T is the path (i.e. sequence of messages) between the exploring ancestor of T and T itself. Note that all messages on the helper tail are helper messages except the first one.

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

I1 Let T be a message that is sent over an arc $\langle a, b\rangle$. If T is exploring, then it holds that no ancestor of T has been sent over $\langle a, b\rangle$. Conversely, if T is helper, there exists some ancestor of T that has been sent over $\langle a, b\rangle$.

I2 Let T be a helper message. Then the helper tail of T contains at most n vertices.

I3 Let T be a helper message. Then T is sent exactly one step later than the parent of T.

I4 Let T be an exploring message. Then T is sent at most $n + 1$ steps later than the exploring ancestor of T.

I5 Let T be an exploring message delivered in a step t. If the broadcast is not finished yet, at least one exploring message is delivered in some of the steps

[3] We assume that T is not an ancestor of itself, hence T and the exploring ancestor of T are always distinct.

$t+1, \ldots, t+n$.

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 message-tree. Invariant *I1* ensures that the leaf is an exploring message and that there are at most $O(n^2)$ exploring messages on this path. Invariant *I4* implies that there are at most $n+1$ consecutive helper messages on the path. Hence the overall time of the broadcasting algorithm is $O(n^3)$.

The algorithm works as follows. In the first step of the algorithm, the initiator sends exploring messages through all its outgoing arcs. All these messages are children of some virtual root message in the message-tree. In each subsequent step t, each vertex gathers all received messages in this step and processes them in parallel using the procedure PROCESS described in Algorithm 3.

Algorithm 3 Complete graphs without sense of direction

```
 1: procedure PROCESS(T)    // process message T
 2:     Let P be the set of all arcs outgoing out of the vertex processing T
 3:     Let A ⊆ P be the set of arcs 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 helper message in
        the helper tail of T.
 6:     Let B ⊆ P − A be the set of arcs that lead to a vertex in S.
 7:     Let C = P − (A ∪ B)
    // Note that since only arcs already traversed by (an ancestor of) T
    // are considered, the vertex processing T can indeed compute B and C.

 8:     for the first step of processing T do
 9:         Send new helper messages with parent T to all arcs in C
10:         Send new exploring messages with parent T to all arcs in A
11:     end for

12:     Let l be the length of the helper tail of T.
13:     for subsequent n − l steps of processing T do
14:         Send new exploring messages with parent T to all arcs in A
15:     end for
16: end procedure
```

If some processor should send more than one message through an arc in one step, it (arbitrarily) chooses one of them to send and discards the remaining ones.

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

Proof:

I1 Since exploring (helper) messages are sent only at lines 10 and 14 (at line 9), the definition of A at line 3 (C at line 7) ensures the first (second) statement of this invariant, respectively.

I2 Helper message T can not be sent to a vertex that sent some message from the helper tail of T. This is ensured by the definition of C at line 7. (Note that this does not hold for exploring message T.) Hence, the helper tail of any message can contain each vertex at most once.

I3 The helper message can be sent only at line 9, i.e. immediately after receiving its parent message.

I4 Let T be an exploring message, message U be the parent of T, l be the length of the helper tail of U and V be the first message on the helper tail of U. It is necessary to prove that T is sent at most $n+1$ steps later that V.

The invariant *I2* implies that $l \leq n$. The invariant *I3* ensures that U is sent exactly l steps later than V. Hence, if T is sent at line 10, it is sent $l+1$ steps later than V. Otherwise T is sent at line 14 at most $l+1+n-l$ steps later than V.

I5 Assume the contrary, i.e. some exploring message is delivered in step t, only helper messages are delivered in subsequent n steps and the broadcast is not finished. Invariants *I2* and *I3* ensure that no helper messages can be delivered in the step $t+n$ or later. Therefore, the last message is delivered in the step u, for some $u < t+n$. Since exactly $n-1$ messages are sent in the first step of processing message T (see lines 9 and 10), it holds that $t < u$.

Let T be any message delivered in the step u to vertex p. Let $k < n$ be the number of messages in the helper tail of T. Let $S \neq T$ be a message in the helper tail of T, delivered to a vertex q. From construction it follows that in step u the message S is still processed by q at line 14. Because the broadcast is not finished yet, q sends at least one message in step u; combining for all such vertices in the helper tail of T yields $k-1$ messages sent in the step u. Processor p sends $|P| - (|A| + |B|) \geq (n-1) - (|A| + k - 1)$ messages at line 9 and A messages at line 10 in the step u. Summing up, there are at least $n-1$ messages sent in step u, therefore there is at least one message delivered in step $u+1$, a contradiction.

□

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.*

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 steps 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 $\frac{(n-1)(n-2)}{2} = \Omega(n^2)$ time steps. 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

$1 \ldots n-1$, 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[4] $\mathcal{P} = \{P_1 \ldots P_k\}$ from itself to v and then pushes the information through all the paths simultaneously. Let $P_i = (u_0 = u, u_1, \ldots u_{l_i} = v)$; consider an arc $e = \langle u_j, u_{j+1} \rangle$. This arc can be either *sleeping*, *active* or *passive*:

(1) The arc e is *passive* if and only if a message has been received over both e and the arc opposite to e, i.e. $\langle u_{j+1}, u_j \rangle$.
(2) The arc e is *active* if and only if it is not passive and a message has been received over the arc $\langle u_{j-1}, u_j \rangle$. In case $j = 0$ the arc e is active whenever it is not passive.

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

(3) The arc 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 arc has been made: at least one active arc becomes passive, at least one sleeping arc becomes active or the vertex v becomes informed.

The procedure ROUND() defined in Algorithm 4 is the core of the algorithm; it is performed in each round by every vertex $w \in \mathcal{P}$.

Algorithm 4 k-connected graphs

```
 1: procedure ROUND(vertex w)
 2:     Let A be the set of active arcs outgoing out of 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
 5:                 // one iteration per time step
 6:             Let C be the set of arcs outgoing out of w via which an activating
 7:             message has been received in the current round (not necessarily
 8:             in the current time step)
 9:             for e such that opposite arc of e is in C do
10:                 send deactivating message through e
11:                     // all in one time step
12:             end for
13:             for e ∈ A such that e ∈ P_z ∧ z ∉ B do
14:                 send activating message through e
15:                     // all in the same time step as in 10
16:             end for
17:         end for
18:     end for
19: end procedure
```

It is easy to see that the uninformed vertices never send any messages and that at any time each vertex can determine all active arcs 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 arc becomes passive, or a sleeping arc 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 arc 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 arc 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 i paths $\mathcal{P}'$ with an arc 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 arc and from construction it follows that unless the vertex v is informed, there is at least one active arc 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. $B = \{j \mid P_j \in \mathcal{P}'\}$). 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 arc $e \in \mathcal{P}'$ and no deactivating message is delivered (by assumption that no active arc 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[5] $2m$, as all paths in $\mathcal{P}$ together cannot contain more than all m arcs and each arc 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 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

[5] some topology-specific optimization is possible here

(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 $O(d)$ arcs 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 for arbitrary graphs with 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. Now we present this algorithm in more detail.

As in the case of complete graphs, we use the notion of area knowledge; the knowledge of an informed vertex u is denoted as $K(u)$. It is clear that if there is at least one vertex u such that $|K(u)| = 2m$ then all vertices are informed.

The broadcasting algorithm runs in $2m$ phases. At the beginning of phase i there is at least one vertex (phase initiator) with a knowledge of at least i triples, the goal of phase i is to ensure this invariant holds for phase $i + 1$.

Let a be an initiator [6] of a phase and $K(a)$ be its knowledge at the beginning of the phase. Then each arc $\langle b, c\rangle$ can be classified as *helper* or *exploring* (with respect to a, in the current phase), depending on whether the corresponding triple (b, p, c) is contained in $K(a)$ or not. The goal of a is to propagate its knowledge through some of its exploring arcs. Any vertex u that receives information through such arc becomes an initiator in the next phase. As u's knowledge at that moment is strictly larger than what $K(a)$ was at the beginning of the phase, the phase invariant holds.

The main idea of the algorithm for one phase is similar to the approach used in the case of full topology knowledge: The initiator chooses k edge-disjoint

[6] a phase might have several initiators

paths such that the last arc of each path is exploring and all inside arcs are helper arcs, and then pushes the information along these paths until at least one path is fully traversed. The fact that the last arc of a path is exploring ensures progress; the fact that inside arcs are helper arcs ensures that the participating vertices know where the paths continue (in fact, the initiator cannot construct paths with inside exploring arcs as it does not know what their endpoints are).

If there is a single initiator in the phase, this approach would work nicely with no changes to procedure Round(). However, each phase can have several initiators and their computations can disrupt each other. The algorithm for one phase thus consists of $n-1$ subphases. The goal of a subphase is to ensure that either some knowledge is propagated (i.e. one path is fully traversed), or at least one initiator (but not all of them) is eliminated. Hence, after $n-1$ subphases some knowledge is propagated.

The algorithm for one subphase is as follows. Each initiator v chooses k edge disjoint directed paths starting at v. We call these paths *threads* and denote them $T(v,j)$ for $j = 1, 2 \ldots, k$. Let $T(v,j) = (v = u_0, u_1, \ldots, u_l)$. The arcs (u_i, u_{i+1}) are called *forward arcs* of $T(v,j)$; analogously, arcs (u_i, u_{i-1}) are called *backward arcs*. The threads are chosen in such a way that all forward arcs of $T(v,j)$ are helper with respect to v, except the last arc (u_{l-1}, u_l), which is exploring. Note that v can always choose k such threads (from the k-connectivity of the graph; a thread might consist of a single exploring arc). An example of such situation is depicted at Figure 5.

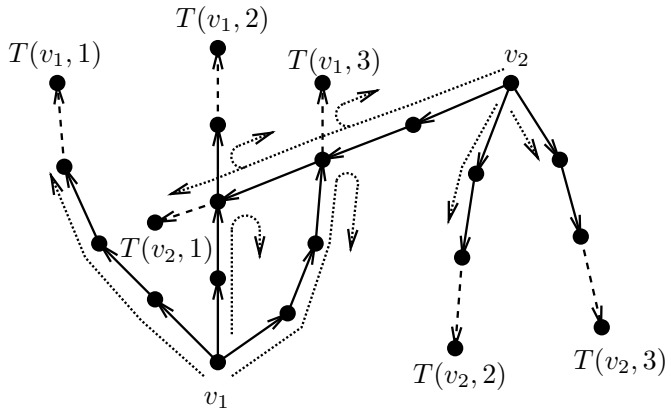

Fig. 5. An example of a subphase. Both v_1 and v_2 constructed 3 threads. Helper arcs are marked as solid lines, exploring arcs are marked as dashed lines. Dotted lines show one possible propagation of messages.

In a subphase, the messages are propagated in the same way as in a phase of the Algorithm described in the section 4.1, until the threads of different initiators collide (that is, a vertex receives messages from different initiators). In such case, the threads "bounce" back to their initiators. The algorithm guarantees that if no knowledge has been propagated then each subphase

initiator receives at least one bounced thread. Each bounced thread carries the ID of the initiator of the colliding thread, therefore each initiator knows the ID of at least one other initiator. Only the initiators which have not seen an ID higher then their own proceed to become initiators of the next subphase. Obviously, the initiator with the biggest ID survives and initiator with the smallest ID does not. This means that the number of initiators decreases, but there is always at least one initiator left.

Each subphase consists of $4m$ rounds. Let $p = \langle u_1, u_2 \rangle$ be an arc outgoing out of vertex u_1. At the beginning of each round the state of p with respect to a thread $T(v, j)$ is determined as follows (only messages received in the current subphase are considered):

(1) The arc p is *passive* if and only if u_1 has received message of type *passive* via the opposite arc of p (i.e. from u_2).
(2) The arc p is *forward-active* if and only if all of the following conditions hold:
   (a) p is not passive.
   (b) p is a forward arc in $T(v, j)$ and u_1 has received some message of the thread $T(v, j)$
   (c) u_1 has not received any message of a thread $T(v', j')$ such that $v' \neq v$.
(3) The arc p is *backward-active* if and only if all of the following conditions hold:
   (a) p is not passive.
   (b) p is a backward arc of $T(v, j)$ and u_1 has received some message of the thread $T(v, j)$
   (c) u_1 has received a message from a thread $T(v', j')$ such that $v' \neq v$ **or** the vertex u_1 has received a message of the thread $T(v, j)$ of type *backward-active*.
(4) Otherwise the arc p is *sleeping*.

An arc is caled *active* if it is either forward-active or backward-active.

In each round each vertex tries to push *forward-active* messages through all its outgoing forward-active arcs and *backward-active* messages through all its outgoing backward-active arcs. An initiator receives a bounced thread if it receives a *backward active* message of that thread.

Each round ensures that if there is at least one initiator that has not received a bounced thread nor propagated its knowledge, then at least one active arc becomes passive or at least one new active message is delivered, i.e. some sleeping arc becomes active in the next round. Hence after applying at most $4m$ rounds the subphase is complete.

The algorithm for one round, described as Algorithm 6, is very similar to the one used before. The correctness of the algorithm relies on the following facts:

Algorithm 5 k-connected graphs without additional structural information

```
1: procedure ROUND(vertex v)
2:     Let A is the set of arcs outgoing out of v active 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
5:                 // all messages for one B are sent simultaneously
6:             for e such that an activating message has been received
7:             through e in the current round do
8:                 if the activating message belongs to T(u, z)
9:                    such that z ∈ B then
10:                    send a message of type passive through e
11:                end if
12:            end for
13:            for e ∈ A such that e ∈ T(u, z) ∧ z ∉ B do
14:                send an activating message of same type as the state of e
15:                through e
16:            end for
17:        end for
18:    end for
19: end procedure
```

(1) No arc can be forward-active and backward-active at the same time. Indeed, if some vertex v has received messages from threads of different initiators, then it cannot have outgoing forward-active arcs. Moreover, the threads of the same initiator are edge-disjoint.
(2) Each forward-active arc is forward-active on exactly one thread. (The same reasons.)
(3) Each backward-active arc is backward-active on exactly one thread. From construction: if some arc outgoing out of the vertex v was backward-active on two threads $T_1 = T(v_1, j_1)$, $T_2 = T(v_2, j_2)$, then T_1 and T_2 share the predecessor u of v and the arc $\langle u, v\rangle$ would had been forward-active on both threads.

This implies that at most one active message is sent per arc per time step. If both an active and a passive message should be sent through one arc in the same time step, the passive message takes priority.

Lemma 5 *During one round either an initiator has propagated its knowledge, or all initiators have received a bounced thread, or an active arc becomes passive or a sleeping arc becomes active.*

Proof: By contradiction, analogous to the proof of Lemma 4. Assuming the contrary, we show that at the beginning of the i-th subround there are at least i threads $\mathcal{T} = T(v_1, j_1), \ldots, T(v_i, j_i)$ such that no two $j_1, \ldots j_i$ are equal and

some active message belonging to any of these threads has been delivered in the current round. This means that in the k-th subround at least k passive messages are being sent and at least one of them is delivered, contradiction.

The proof is by induction on i, with the case $i = 0$ being trivial. Assume that at the beginning of the i-th subround there are exactly i threads satisfying above-mentioned conditions (if there are more than i such threads, the invariant obviously holds for the subround $i + 1$). Let v be any initiator that has not received a bounced thread. It is easy to see that there is at least one active arc (either forward-active or backward-active) on each thread of v such that no passive message is to be sent on this arc. Indeed, let T be any such thread. If it has not collided with any other thread yet, then the last informed vertex of T has an outgoing forward-active arc (and a passive message is never sent through a forward-active arc). Otherwise at least one vertex of T has an outgoing backward-active arc such that no passive message is to be sent on it (the one that is closest to the thread initiator).

Now focus on the time step in the subround when B contains exactly the numbers of the threads from $\mathcal{T}$ (i.e. $B = \{j \mid T(u, j) \in \mathcal{T}\}$). In this time step at least $|B|$ passive and at least $k - |B|$ active messages are sent, therefore at least one of them must be delivered. Assuming no passive message is delivered yields that an active message on a new, $i + 1$-th, thread is delivered. Hence the induction hypothesis is ensured for $i + 1$ as well. □

Putting the pieces together:

(1) *The whole algorithm:* Consists of $2m$ phases.
(2) *One phase:* The goal is to increase the maximal knowledge in the system by at least one arc. Consists of $n - 1$ subphases.
(3) *One subphase:* The goal is to eliminate at least one phase initiator, unless the maximal knowledge is increased. Consists of $4m$ rounds.
(4) *One round:* The goal is to make some sleeping arc active or some active arc passive, unless the subphase makes progress. Consists of $k+1$ subrounds.
(5) *One subround:* The goal is to deliver a new active message, thus increase the number of passive messages that are to be sent. The i-th subround of the round consists of $\binom{k}{i}$ time steps.

Combining together results in $2m \times (n - 1) \times 4m \times 2^k = O(2^k m^2 n)$ overall time complexity.

Theorem 8 *There is a fault-tolerant broadcasting algorithm on k-connected graphs without sense of direction that uses $O(2^k m^2 n)$ time, where m is the number of edges and n is the number of the vertices of the graph.*

As the time complexity is only exponential in the connectivity of the graph,

we get:

Corollary 2 *There is a polynomial-time broadcasting algorithm on any graph without sense of direction with edge connectivity* $O(\log n)$.

Directly applying Theorem 8 yields:

Corollary 3 *There is a broadcasting algorithm on d-dimensional hypercube without sense of direction 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] R. Ahlswede, L. Gargano, H. S. Haroutunian and L. H. Khachatrian, "Fault-Tolerant Minimum Broadcast Networks". *Networks* 27, 1996.

[2] P. Berman, K. Diks, and A. Pelc, "Reliable broadcasting in logarithmic time with Byzantine link failures". *Journal of Algorithms*, 22 (2), 199–211, 1997.

[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. Flocchini, B. Mans, N. Santoro, "Sense of Direction: Definitions, Properties and Classes", *Networks*, 32(3), p. 165-180, 1998.

[13] P. Fraigniaud and C. Peyrat, "Broadcasting in a hypercube when some calls fail", *Information Processing Letters* 39, 115–119, 1991.

[14] L. Gargano, A. A. Rescigno and U. Vaccaro, "Minimum time broadcast in faulty star networks". *Discrete Applied Mathematics* 83, 97–119, 1998.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.