Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 384 (2007) 232–249
Theoretical Computer Science
www.elsevier.com/locate/tcs

Agreement in synchronous networks with ubiquitous faults

Nicola Santoro[a,*], Peter Widmayer[b]

[a] *School of Computer Science, Carleton University, Ottawa, Canada*
[b] *Institute for Theoretical Informatics, ETH Zentrum, Zurich, Switzerland*

Abstract

In this paper we are interested in synchronous distributed systems subject to transient and ubiquitous failures. This includes systems where failures will occur on any communication link, systems where every processor will experience at one time or another send or receive failure, etc., and, following a failure, normal functioning resuming after a finite time. Notice that these cases cannot be handled by the traditional *component failure* models.

The model we use is the *communication failure* model, also called the *transmission failure* or *dynamic faults* or *mobile faults* model. Using this model, we study the fundamental problem of *agreement* in synchronous networks of arbitrary topology with ubiquitous faults.

We establish bounds on the number of dynamic faults that make any non-trivial form of agreement (even strong majority) impossible; in turn, these bounds express connectivity requirements that must be met to achieve any meaningful form of agreement. We also provide, constructively, bounds on the number of dynamic faults in spite of which any non-trivial form of agreement (even unanimity) is possible. These bounds are shown to be tight for a large class of networks, which includes hypercubes, toruses, rings, and complete graphs; incidentally, we close the existing gap between possibility and impossibility of non-trivial agreement in complete graphs in the presence of dynamic Byzantine faults.

None of these results is derivable in the component failure models; in particular, all our *possibility* results hold in situations for which those models indicate *impossibility*.

Keywords: Theory of distributed computation; Synchronous systems; Dynamic faults; Mobile faults; Transmission failure model; Arbitrary network topology; Agreement; Majority; Bivalency argument; Limits to computability

1. Introduction

1.1. The framework

In this paper we are concerned with the fundamental problem of reaching *agreement* in the presence of faults. We are interested in synchronous distributed systems subject to (possibly transient) *ubiquitous* failures; that is, faults occur anywhere in the system and, following a failure, normal functioning can resume after a finite (although unpredictable)

* Corresponding author. Tel.: +1 613 520 4333; fax: +1 613 520 4334.
E-mail addresses: santoro@scs.carleton.ca (N. Santoro), widmayer@inf.ethz.ch (P. Widmayer).

0304-3975/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2007.04.036

time. This includes systems where failures will occur on any communication link, systems where every processor will experience at one time or another send or receive failure, etc.

The reality of these systems is not fully captured by the *component failure* models proposed in the literature.

Consider, for instance, the *processor failure* model (e.g., see [15,17,19,21,28]). In this model, only processors can be faulty; any other type of failure is inscribed to the faulty behaviour of some of the involved processors: if a message is lost, either the sending or the receiving processor will be declared faulty; once this happens, that processor will be forever considered faulty. This leads to undesirable conclusions: in the case of ubiquitous failures where any processor may occasionally lose messages (a situation that clearly occurs in real systems), the entire system will be declared unusable for any computation.

Analogous undesirable situations occur in the *link failure* model and in the *hybrid failure* models that consider both links and processors (e.g. [8,33,36]). Some attempts have been made to remedy this situation by modifying the model, allowing some failed processors and links to recover (e.g. [1,25]), but they are just partial and limited to eventually-synchronous systems.

The model we use is the *communication failure* model, known also as the *transmission failure* or *dynamic faults* or *mobile faults* model, introduced in [34] and investigated e.g. in [6,7,9,10,12,11,13,14,22,24,27,29,35].

In this model, a *communication* is a pair (α, β) of messages $\alpha, \beta \in M \cup \{\Omega\}$ for a pair (i, j) of neighbouring processors called *source* and *destination*, where M is a fixed and possibly infinite message universe and Ω is the null message: α is the message sent by the source and β is the message received by the destination; by convention $\alpha = \Omega$ denotes that no message is sent, and $\beta = \Omega$ denotes that no message is received. A communication (α, β) is *faulty* if $\alpha \neq \beta$, non-faulty otherwise. In this model, the only failures occurring in the system are communication faults, and failures are fully *dynamic*: the set of pairs (*source*, *destination*) whose communications are faulty may change at every clock cycle. An instance of this model is the *single mobile failure* model described in [30]: at each time instant, the communications of at most one processor are faulty.

Notice that localized and permanent failures can be easily modeled by communication faults; for instance, omission (i.e. $\alpha \neq \Omega = \beta$) of all messages sent by and to a processor can be used to describe the crash failure of that processor. Analogously, with enough dynamic communication faults of the appropriate type, it is easy to describe faults such as send and receive failures, Byzantine link failures, etc. In fact, most processor and link failure models can be seen as a special *localized* case of the communication failure model, where all the faults are restricted to the communications involving a fixed (though, a-priori unknown) set of processors or of links.

Dynamic communication faults are clearly more difficult to handle than those that occur always in the same places. In the latter case, for example, once a fault is detected on a link, we know that we cannot trust that link; with dynamic faults, detection will not help us with future events. The natural questions and open problems are about the nature of this difficulty.

In this paper, we address those questions and problems, focusing on the agreement problem in synchronous systems of arbitrary topology subject to dynamic communication faults.

1.2. Main contributions

In this paper we present several results, extending the existing knowledge on agreement to ubiquitous and transient failures in arbitrary topologies.

Impossibility

First of all, we prove a general theorem characterizing classes of faulty communications for which any non-trivial agreement is *impossible*. As a corollary, we prove several impossibility results, some of them quite unexpected, and not inferable from the existing results of the other models.

Consider, e.g., a d-dimensional *hypercube* and let us consider the occurrence of just d communication faults per clock cycle. With d *omissions* per clock cycle, we can simulate the crash failure of a single processor; hence, *unanimity* clearly cannot be expected. What about other levels of agreement?

Something can be learned from the processor failure model. For the hypercube, Dolev's results in the processor failure model [15] state that unanimity among the non-faulty processors is possible if the number of the faulty processors is less than d. If the d omissions per clock cycle are localized to a single processor, there is only one "faulty" processor; hence, by Dolev's result, an agreement among $n - 1$ processors is possible.

What happens if those d omissions per clock cycle are *dynamic* (i.e. not localized to a single processor)? Since with at most d omissions per clock cycle a single processor can be isolated from the rest, one might still reasonably expect that an agreement among $n-1$ processors can be reached even if the faults are dynamic. Not only is this expectation false, but we prove that any form of non-trivial agreement can not be reached under those conditions. In fact, even a strong majority (i.e. an agreement among $\lceil n/2 \rceil + 1$ processors) is impossible.

If the communication faults are arbitrary (the *Byzantine* case), the gap between the static and dynamic cases is even stronger. By Dolev's result [15], it follows again that in the hypercube an agreement among $n-1$ processors is possible in spite of d Byzantine communication faults if they are statically restricted to the messages sent by a single processor. From our results with omissions, we already know that if the faults are dynamic then strong majority is impossible; if the faults are Byzantine, we show that this is so even if the number of faults is just $\lceil d/2 \rceil$.

These results for the hypercube are instances of the more general results obtained here. We consider *arbitrary networks* and establish bounds on the number of communication faults that make any non-trivial form of agreement impossible; in turn, these bounds express topological requirements that must be met to achieve any meaningful form of agreement. Let $G = (V, E)$ be the network topology, and let $d(G)$ be its degree. We prove that:

(a) with $d(G)$ omissions per clock cycle, strong majority cannot be reached;
(b) if the failures are any mixture of corruptions and additions, the same bound $d(G)$ holds for the impossibility of a strong majority;
(c) in the case of arbitrary faults (omissions, additions, and corruptions: the Byzantine case), a strong majority cannot be reached if only $\lceil d(G)/2 \rceil$ communications may be faulty.

A summary is shown in Fig. 2.

These results are established using the proof structure for dynamic faults introduced in [34]. Although based on the bivalency argument of Fischer et al. [21], the framework differs significantly from those for asynchronous systems since we are dealing with a *fully synchronous* system where time is a direct computational element, with all its consequences; e.g., non-delivery of an expected message is detectable, unlike asynchronous systems where a "slow" message is indistinguishable from an omission; etc. This framework has been first defined and used in [34]; more recently, similar frameworks have been used also in [2,3,30].

Possibility

We then turn to the possibility of agreement in spite of dynamic faults. We examine all the combinations of different types of fault, and for each we establish bounds for achieving *unanimity* among the processors. The results we obtain vary with the types of fault and are sometimes counterintuitive.

Consider, e.g., the case when the faults are just *omissions*. It is known that, in the d-dimensional *hypercube*, if the faults are at most $d-1$ omissions per clock cycle, then broadcast (and thus unanimity) is possible (e.g. [10,13]).

These results are actually just instances of the more general results established here. In fact, we prove that in any network G, if the faults are *omissions*, then *unanimity* can be reached if the number of faults per clock cycle is at most $c(G)-1$, where $c(G)$ is the edge-connectivity of G.

Interestingly, if the faults are *corruptions*, we show that unanimity can always be achieved regardless of the number of faults. This is true also in systems where the faults are only *additions*. On the other hand, the combination of *additions and corruptions* creates a $c(G)-1$ threshold for unanimity.

In the more complex *Byzantine* case of *omissions, additions and corruptions*, we prove that unanimity is still possible if at most $\lceil c(G)/2 \rceil - 1$ transmissions per clock cycle may be faulty. A summary of all the possibility results is shown in Fig. 4.

Let us stress that, regardless of any analogy with bounds established in the component failure models, none of these results is implied or derivable from those models. On the contrary, these *possibility* results are obtained with a number and type of fault for which all the component failure models indicate *impossibility*.

Tightness

For all systems, except those where faults are just corruptions or additions (and in which unanimity is possible regardless of faults), the bounds we have established are similar except that the possibility ones are expressed in terms of the *connectivity* $c(G)$ of the graph, while the impossibility ones are in terms of the *degree* $d(G)$ of the graph.

This means that in the case of $d(G)$-*edge-connected graphs*, the impossibility bounds are indeed tight:

1. With the number of faults (or more) specified by the impossibility bound, even strong majority is impossible.
2. With one less fault than specified by the impossibility bound, even unanimity can be reached.
3. Any agreement among less than a strong majority of the processors can be reached without any communication.

In other words, in these systems, agreement is either trivial, complete or impossible.

This large class of networks includes hypercubes, toruses, rings, complete graphs, etc. In these networks, the obtained results draw a precise "impossibility map" for the agreement problem in the presence of dynamic communication faults, thus clarifying the difference between the dynamic and the static cases. In the case of complete graphs, our results close the existing gap in [34,35] between possibility and impossibility with dynamic Byzantine faults.

For those graphs where $c(G) < d(G)$ there is a gap between possibility and impossibility. Closing this gap is clearly a goal of future research.

1.3. Related work

Synchronous agreement in the *component failure* models is perhaps the most intensively and extensively studied problem in distributed computing. We will just note that most of the work has focused on the *complete graph* (e.g. [5,16,17,19,20,23,26,28,32,36]); fewer studies have focused on other classes of graph (e.g. [4,18,20,28]) or on arbitrary networks [15].

In the *communication faults model*, the studies on agreement have focused on synchronous systems whose communication topology is the *complete graph*, and both possibility and impossibility results have been established [34,35]. In particular, for these systems the established bounds have been shown to be tight in the case of *omission* (i.e. $\alpha \neq \beta = \Omega$) failures, as well as in the case of any mix of *corruption* (i.e. $\Omega \neq \alpha \neq \beta \neq \Omega$) and *addition* (i.e. $\Omega = \alpha \neq \beta$) failures. In the case of *Byzantine* (i.e. arbitrary) communication failures, there was, However, a gap between the possibility and impossibility bounds.

The link between conditions for (partial) *broadcast* and for possibility of (partial) agreement with dynamic faults was established in [34]. Most of the subsequent research on dynamic faults has focused on reliable broadcast in the case of *omission* failures; the problem has been investigated in complete graphs [11,29,35], hypercubes [10,13,22,31], tori [9,14], star graphs [10], as well as in arbitrary topologies [6].

The broadcast problem has also been studied when the upper bound on the number of dynamic communication faults per clock cycle is not fixed [7,24,27].

The more general problem of evaluation of Boolean functions in the presence of dynamic communication faults has been studied only for complete networks [12,35]; computation of some special functions has been investigated also in the case of *anonymous* networks [12].

The employment of a bivalency argument to achieve impossibility results in a *fully synchronous* system has been first done in [34]; more recently, it has been used also in [2,3,30].

2. Communication faults and agreement

Our universe is a synchronous system of $n \geq 2$ processors $p_1, \ldots, p_n$ connected through dedicated communication links. The connection structure is an arbitrary undirected, connected simple graph $G = (V, E)$. An edge joining two nodes of the graph is a bidirectional link between the two processors. We will denote by d_i the degree of a node p_i in the graph, and by $d(G)$ the maximum node degree. We will also denote by $c(G)$ the edge-connectivity of G.

Processors communicate by sending messages to adjacent processors. The message sent by a processor need not be the same for all destination processors, i.e. separate links allow for different messages to different destinations at the same time.

The failure model we use is the *communication failure* (also known as *transmission failure* or *dynamic faults*) model, introduced by us in [34], and described below.

A *communication* is a pair (α, β) of messages $\alpha, \beta \in M \cup \{\Omega\}$ for a pair (i, j) of neighbouring processors called *source* and *destination*, where M is a fixed and possibly infinite message universe and Ω is the null message: α is the message sent by the source and β is the message received by the destination; by convention $\alpha = \Omega$ denotes that no message is sent, and $\beta = \Omega$ denotes that no message is received. Let Φ denote the set of all possible communications.

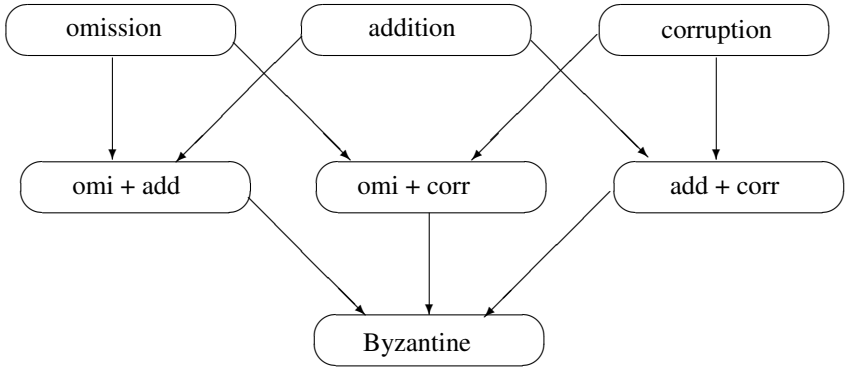

Fig. 1. Hierarchy of combinations of fault types in the communication faults model.

A communication (α, β) is *faulty* if $\alpha \neq \beta$, *non-faulty* otherwise. Faulty communications can be partitioned into three sets, corresponding to the three types of fault considered in this paper:

- *omissions*: $\mathcal{O} = \{(\alpha, \beta) \in \Phi : \alpha \neq \Omega = \beta\}$
  (a sent message is not delivered to its destination processor);
- *additions*: $\mathcal{A} = \{(\alpha, \beta) \in \Phi : \alpha = \Omega \neq \beta\}$
  (a message is delivered to a processor, although no message was sent);
- *corruptions*: $\mathcal{C} = \{(\alpha, \beta) \in \Phi : \Omega \neq \alpha \neq \beta \neq \Omega\}$
  (a sent message is delivered with different content to its destination processor).

While the nature of omissions and corruptions is quite obvious, that of additions may require some explanation. Indeed, an addition describes a variety of situations. The most obvious one is when sudden noise in the transmission channel is mistaken for a message. The more important occurrence of additions in systems is rather subtle: when we say that, in an addition, the received message "was not sent", this could mean that it "was not sent by any authorized user". Indeed, additions can be seen as messages surreptitiously inserted in the system by some outside, and possibly malicious, entity. Spam being sent from an unintended and unsuspecting site clearly fits the description of an addition. Summarizing, additions do occur and can be very dangerous.

These three types of fault are quite incomparable with each other in terms of danger. The hierarchy of faults comes into place when two or all of these basic fault types can occur in the system (see Fig. 1). The presence of all three types of fault creates what will be called a Byzantine faulty behaviour.

Notice that localized and permanent failures can be easily modeled by communication faults; for instance, omission of all messages sent by and to a processor can be used to describe the crash failure of that processor. Analogously, with enough dynamic communication faults of the appropriate type, it is easy to describe faults such as send and receive failures, Byzantine link failures, etc. In fact, most processor and link failure models can be seen as a special *localized* case of the communication failure model, where all the faults are restricted to the communications involving a fixed (though, a priori unknown) set of processors or of links.

Each processor p_i has an input register with an initial value x_i, and an output register for which it must choose a value v_i as the result of its computation. For simplicity, we limit ourselves to Boolean inputs, i.e. $x_i \in \{0, 1\}$ for all i; all the results hold also for non-binary values. In the *k-agreement problem* (**Agree(k)**), at least k processors must choose the same value v within a finite amount of time, subject to the *validity* constraint that, if all values x_i were the same, then v must be that value.

Depending on the value of parameter k, we have different types of agreement problem. Of particular interest are

- $k = \lceil \frac{n}{2} \rceil + 1$ called *strong majority*;
- $k = n$ called *unanimity*, in which all processors must decide on the same value.

Note that any agreement requiring *less* than a strong majority (i.e. $k \leq \lceil n/2 \rceil$) can be trivially reached without any communication; e.g. each p_i chooses x_i for v. In this paper, we are interested only in *non-trivial* agreements (i.e. $k > \lceil n/2 \rceil$).

For both impossibility and possibility results, we will consider the case when the system operates in complete *synchrony*: each processor has direct read-only access to a global clock, and every message sent at time t is received (if no error occurs) and processed at its destination at time $t + 1$.

3. Impossibility of strong majority

3.1. Terminology and definitions

Most of the terminology is taken from [34]; some of it was in turn a modification and adaptation for the synchronous case of that of [21].

A k-agreement protocol P, $k > \lceil n/2 \rceil$, is a synchronous system where each processor p_i, in addition to its one-bit input register with initial value $x_i \in \{0, 1\}$ and to its output register, has an unbounded amount of local storage. In particular, it has a message register holding a message vector $m_i \in (M \cup \{\Omega\})^{d_i}$, where M is a fixed and possibly infinite message universe, and Ω is the null element indicating absence of communication. The component m_{ij} of m_i is the message to be sent from p_i to its neighbour p_j. The values of the registers and of the global clock, together with the program counters and the internal storage, comprise the *internal state* of each processor.

For each processor, the *initial state* prescribes fixed starting values for all but the input register; in particular, the output register starts with null value $b \notin \{0, 1\}$ and the clock starts with value 0. Each processor acts deterministically; it can never change the input register nor the clock, and can change the value of its output register only once, from b to $v \in \{0, 1\}$. The states in which the output register has value $v \in \{0, 1\}$ are distinguished as being *v-decision states*.

A *configuration* of the system consists of the internal state of all processors at a given time. An *initial* configuration is one in which all processors are in an initial state at time $t = 0$. A configuration C has *decision value* v if at least k processors are in a v-decision state, $v \in \{0, 1\}$; note that since $k > \lceil n/2 \rceil$, a configuration can have at most one decision value.

At any time t, the system is in some configuration C, and every processor may send a message to any of its neighbours. The nature of the messages is defined by the contents of the message registers in C. The set of all messages sent in a configuration C is represented by means of a *message array* $\Lambda(C)$ composed of n^2 entries defined as follows: if p_i and p_j are neighbours, then the entry $\Lambda(C)[i, j]$ contains the (possibly empty) message sent by p_i to p_j; if p_i and p_j are *not* neighbours, then we denote this fact by $\Lambda(C)[i, j] = *$, where $* \notin M$ is a distinguished symbol; by definition, $(p_i, p_i) \notin E$.

During the actual communication, some of these messages will not be delivered, or their content will be corrupted, or a message arrives when none was sent. We will describe what happens by means of another $n \times n$ array called *transmission matrix* τ for $\Lambda(C)$ and defined as follows: if p_i and p_j are neighbours, then the entry $\tau[i, j]$ of the matrix contains the communication pair (α, β), where $\alpha = \Lambda(C)[i, j]$ is what p_i sent and β is what p_j actually receives; if p_i and p_j are *not* neighbours, then we denote this fact by $\tau[i, j] = (*, *)$. Where no ambiguity arises, we will omit the indication C from $\Lambda(C)$. Due to the different number and types of fault and the different way the faults can occur, many transmission matrices are possible for the same Λ. We will denote by $\mathcal{T}(\Lambda)$ the set of all possible transmission matrices τ for Λ.

Once the transmission specified by τ has occurred, the clock is incremented by one unit to $t + 1$; depending on its internal state, on the current clock value and on the received messages, each processor p_i prepares a new message for each neighbour p_j, and enters a new internal state. The entire system enters a new configuration. Since the processors act deterministically, the new configuration is completely determined by the transmission matrix τ; we will denote by $\tau(C)$. We will call τ an *event* and the passage from one configuration to the next a *step*.

We will limit ourselves to sets of events that contain (at least) an event for each possible message array and at most f faulty communications. A set S of events is *f-admissible*, $0 \leq f \leq 2|E|$, if

1. For each message array Λ, there is an event $\tau \in S$ for Λ;
2. No event in S contains more than f faulty communications;
3. There is an event in S that contains exactly f faulty communications.

Let $R^1(C) = R(C) = \{\tau(C) : \tau \in \mathcal{T}(\Lambda(C))\}$ be the set of all possible configurations resulting from C in one step, sometimes called *succeeding* configurations of C. Generalizing, let $R^t(C)$ be the set of all possible configurations resulting from C in $t > 0$ steps, and $R^+(C) = \{C' : \exists t > 0,\ C' \in R^t(C)\}$ be the set of configurations reachable from C. A configuration that is reachable from some initial configuration is said to be *accessible*.

Let $v \in \{0, 1\}$. A configuration C is *v-valent* if there exists a $t > 0$ such that all $C' \in R^t(C)$ have decision value v; that is, a v-valent configuration will always result in at least k processors deciding on v. A configuration C is *bivalent* if there exist in $R^+(C)$ both a 0-valent and a 1-valent configuration.

If two configurations C' and C'' differ only in the internal state of processor p_j we say that they are *j-adjacent*; and we call them *adjacent* if they are j-adjacent for some j.

We will be interested in sets of events (i.e. transmission matrices) that preserve adjacency of configurations. We call a set S of events *j-adjacency-preserving* if for any two j-adjacent configurations C' and C'' there exist in S two events τ' and τ'' for $\Lambda(C')$ and $\Lambda(C'')$, respectively, such that $\tau'(C')$ and $\tau''(C'')$ are j-adjacent. We call S *adjacency-preserving* if it is j-adjacency-preserving for *all* j.

A set S of events is *continuous* if for any configuration C and for any $\tau', \tau'' \in S$ for $\Lambda(C)$, there exists a finite sequence $\tau_0, \ldots, \tau_m$ of events in S for $\Lambda(C)$ such that $\tau_0 = \tau', \tau_m = \tau''$, and $\tau_i(C)$ and $\tau_{i+1}(C)$ are adjacent, $0 \leq i < m$.

Let S be an f-admissible set of events and let the message system return only events in S. A k-agreement protocol is *correct* in spite of f transmission faults in S if

(a)for each initial configuration C there exists a $t \geq 0$ such that every $C' \in R^t(C)$ has a decision value $v \in \{0, 1\}$;
(b)if all values x_i are the same, then v is that value.

As we will see, any set of f-admissible events that is both continuous and j-adjacency-preserving for some j will make any strong majority protocol fail.

3.2. Basic properties and main theorem

To prove our impossibility results, we are going to use two properties that follow immediately from the definitions of state and of event. Let $s_i(C)$ denote the internal state of p_i in C.

First of all, if a processor p_j is in the same state in two different configurations A and B, then it will send the same messages in both configurations. That is,

Property 3.1. *For two configurations A and B, let $\Lambda(A)$ and $\Lambda(B)$ be the corresponding message matrices. If $s_j(A) = s_j(B)$ for some processor p_j, then $\langle \Lambda(A)[j, 1], \ldots, \Lambda(A)[j, n] \rangle = \langle \Lambda(B)[j, 1], \ldots, \Lambda(B)[j, n] \rangle$.*

Next, if a processor p_j is in the same state in two different configurations A and B, and it receives the same messages in both configurations, then it will enter the same state in both resulting configurations. That is,

Property 3.2. *Let A and B be two configurations such that $s_j(A) = s_j(B)$ for some processor p_j, and let τ' and τ'' be events for $\Lambda(A)$ and $\Lambda(B)$, respectively. Let $\tau'[i, j] = (\alpha'_{i,j}, \beta'_{i,j})$ and $\tau''[i, j] = (\alpha''_{i,j}, \beta''_{i,j})$. If $\beta'_{i,j} = \beta''_{i,j}$ for all i, then $s_j(\tau'(A)) = s_j(\tau''(B))$.*

Given a set S of f-admissible events and an agreement protocol P, let $\mathcal{C}(P, S)$ denote the set of all initial and accessible configurations when executing P and the events are those in S.

Theorem 3.1. *Let S be continuous, j-adjacency-preserving, and f-admissible, $f > 0$. Let P be a $(\lceil \frac{n}{2} \rceil + 1)$-agreement protocol. If $\mathcal{C}(P, S)$ contains two accessible j-adjacent configurations, a 0-valent and a 1-valent one, then P is not correct in spite of f transmission faults in S.*

Proof. Assume on the contrary that P is a $(\lceil \frac{n}{2} \rceil + 1)$-agreement protocol that is correct in spite of f communication faults when the message system returns only events in S; and let A and B be j-adjacent accessible configurations in $\mathcal{C}(P, S)$ that are 0-valent and 1-valent, respectively.

Since S is j-adjacency-preserving, there exist in S two events, $\pi_{A,B}$ for $\Lambda(A)$ and $\rho_{A,B}$ for $\Lambda(B)$, such that the resulting configurations $\pi_{A,B}(A)$ and $\rho_{A,B}(B)$ are j-adjacent. Let $\pi^0(A) := A \quad \text{and} \quad \rho^0(B) := B$; for any $t > 0$ define $\pi^t(A) := \pi_{\pi^{t-1}(A), \rho^{t-1}(B)}(\pi^{t-1}(A))$ and $\rho^t(B) := \rho_{\pi^{t-1}(A), \rho^{t-1}(B)}(\rho^{t-1}(B))$. It is easy to verify that $\pi^t(A)$ and $\rho^t(B)$ are j-adjacent for all $t \geq 0$.

Since P is correct, there exists a $t \geq 1$ such that $\pi^t(A)$ and $\rho^t(B)$ have a decision value. Since A is 0-valent, at least $\lceil \frac{n}{2} \rceil + 1$ processors have a decision value 0 in $\pi^t(A)$; similarly, since B is 1-valent, at least $\lceil \frac{n}{2} \rceil + 1$ processors have a decision value 1 in $\rho^t(B)$. This means that there exists at least one processor p_i, $i \neq j$, that has a decision value 0 in $\pi^t(A)$ and 1 in $\rho^t(B)$; hence, $s_i(\pi^t(A)) \neq s_i(\rho^t(B))$.

However, since $\pi^t(A)$ and $\rho^t(B)$ are j-adjacent, they only differ in the state of one processor, p_j: a contradiction. As a consequence, P is not correct. ■

We can now prove the main negative result.

Theorem 3.2. *Let S be adjacency-preserving, continuous and f-admissible. Then no k-agreement protocol is correct in spite of f communication faults in S for $k > \lceil n/2 \rceil$.*

Proof. Assume P is a correct $(\lceil n/2 \rceil+1)$-agreement protocol in spite of f communication faults when the message system returns only events in S. In a typical bivalency approach, the proof involves two steps: first it is argued that there is some initial configuration in which the decision is not already predetermined; second, it is shown that it is possible forever to postpone entering a configuration with a decision value.

Lemma 3.1. *$\mathcal{C}(P, S)$ has an initial bivalent configuration.*

Proof. By contradiction, let every initial configuration be v-valent for $v \in \{0, 1\}$ and let P be correct. Since, by definition, there is at least a 0-valent initial configuration A and a 1-valent initial configuration B, then there must be a 0-valent initial configuration and a 1-valent initial configuration which are adjacent. In fact, let $A_0 = A$, and let A_h denote the configuration obtained by changing into 1 a single 0 input value of A_{h-1}, $1 \leq h \leq w'$ where w' is the number of 0 input values in A; similarly, define B_h, $0 \leq h \leq w''$, where w'' is the number of 0 input values in B. By construction, $A_{w'} = B_{w''}$. Consider the sequence $A_0, A_1, \ldots, A_{w'} = B_{w''}, \ldots, B_1, B_0$; in it, each configuration is adjacent to the following one; since it starts with a 0-valent and ends with a 1-valent configuration, it contains a 0-valent configuration adjacent to a 1-valent one. By Theorem 3.1 it follows that P is not correct. Hence $\mathcal{C}(P, S)$ must have an initial bivalent configuration. ■

Lemma 3.2. *Every bivalent configuration in $\mathcal{C}(P, S)$ has a succeeding bivalent configuration.*

Proof. Let C be a bivalent configuration in $\mathcal{C}(P, S)$. If C has no succeeding bivalent configuration, then C has at least one 0-valent and at least one 1-valent succeeding configuration, say A and B. Let $\tau', \tau'' \in S$ such that $\tau'(C) = A$ and $\tau''(C) = B$. Since S is continuous, there exists a sequence $\tau_0, \ldots, \tau_m$ of events in S for $\lambda(C)$ such that $\tau_0 = \tau'$, $\tau_m = \tau''$, and $\tau_i(C)$ and $\tau_{i+1}(C)$ are adjacent, $0 \leq i < m$. **Consider now the corresponding sequence of configurations:** $A = \tau'(C) = \tau_0(C), \tau_1(C), \tau_2(C), \ldots, \tau_m(C) = \tau''(C) = B$.

Since the sequence starts with a 0-valent and ends with a 1-valent configuration, it contains a 0-valent configuration adjacent to a 1-valent one. By Theorem 3.1, P is not correct: a contradiction. Hence, every bivalent configuration in $\mathcal{C}(P, S)$ has a succeeding bivalent configuration. ■

From Lemmas 3.1 and 3.2 it follows that there exists an infinite sequence of accessible bivalent configurations, each derivable in one step from the preceding one. This contradicts the assumption that for each initial configuration C there exists a $t \geq 0$ such that every $C' \in R^t(C)$ has a decision value; thus, P is not correct. This concludes the proof of Theorem 3.2. ■

The above theorem provides a powerful tool for proving impossibility results for non-trivial agreement: if it can be shown that a set S of events is adjacency-preserving, continuous, and f-admissible, then by Theorem 3.2 no non-trivial agreement is possible for the types and numbers of faults implied by S. Clearly, not every set S of events is adjacency-preserving.

3.3. Impossibility: Omission faults

We can now show that no *strong majority* protocol is correct in spite of $d(G)$ communication faults, even when the faults are only *omissions*.

For message matrix A = (α_{ij}), let O(A) be the set of all events τ on A defined as follows: for at most $d(G)$ pairs $(i, j) \in E$, $\tau[i, j] = (\alpha_{ij}, \Omega)$, and for all other pairs $(i, j) \in E$, $\tau[i, j] = (\alpha_{ij}, \alpha_{ij})$. Then, $\mathbf{O} := \bigcup_A$ O(A) is the set of all events containing at most $d(G)$ omission faults.

Lemma 3.3. **O** *is $d(G)$-admissible, continuous and adjacency-preserving.*

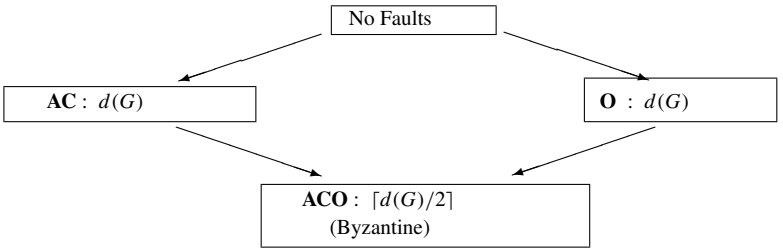

Fig. 2. Minimum number of faults per clock cycle that may render strong majority impossible.

Proof. From the definition of **O**, it follows immediately that **O** is $d(G)$-admissible.

To prove that **O** is continuous, consider a configuration C and any two events $\tau', \tau'' \in \mathbf{O}$ for $\Lambda(C)$. Let $m'_1, m'_2, \ldots, m'_{f'}$ be the f' faulty communications in τ', and let $m''_1, m''_2, \ldots, m''_{f''}$ be the f'' faulty communications in τ''. Since **O** is $d(G)$-admissible, then $f' \leq d(G)$ and $f'' \leq d(G)$. Let $\tau'_0 = \tau'$, and let τ'_h denote the event obtained by replacing the faulty communication m'_h in τ'_{h-1} with a non-faulty one (with the same message sent in both), $1 \leq h \leq f'$; Similarly define τ''_h, $0 \leq h \leq f''$. By construction, $\tau'_{f'} = \tau''_{f''}$. Consider the sequence $\tau'_0, \tau'_1, \ldots, \tau'_{f'} = \tau''_{f''}, \ldots, \tau''_1, \tau''_0$. In this sequence, each event is adjacent to the following one; furthermore, since by construction each event contains at most $d(G)$ omissions, it is in **O**. Thus, **O** is continuous.

Given a message matrix Λ, let $\sigma_{\Lambda,l}$ denote the event for Λ where all and only the messages sent by x_l are lost. Then, for each Λ and l, $\sigma_{\Lambda,l} \in \mathbf{O}$. Let configurations A and B be l-adjacent. Consider the events $\sigma_{\Lambda(A),l}$ and $\sigma_{\Lambda(B),l}$ for A and B, respectively, and the resulting configurations A' and B'. By Properties 3.1 and 3.2, it follows that A' and B' are also l-adjacent. Hence **O** is adjacency-preserving. ■

Then, by Theorem 3.2, it follows that

Theorem 3.3. *No k-agreement protocol P is correct in spite of $d(G)$ communication faults in* **O** *for $k > \lceil n/2 \rceil$.*

3.4. Impossibility: Addition and corruption faults

Here, we show that no strong majority protocol is correct in spite of $d(G)$ communication faults, when the faults are additions and corruptions.

For message matrix $\Lambda = (\alpha_{ij})$, let $\mathbf{AC}(\Lambda)$ be the set of all events τ on Λ defined as follows: for at most $d(G)$ pairs $(i, j) \in E$, $\tau[i, j] = (\alpha_{ij}, \beta_{ij}), \alpha_{ij} \neq \beta_{ij}$, where $\beta_{ij} \neq \Omega$ if $\alpha_{ij} \neq \Omega$, and for all other pairs $(i, j) \in E$, $\tau[i, j] = (\alpha_{ij}, \alpha_{ij})$.

Then $\mathbf{AC} := \bigcup_{\Lambda} \mathbf{AC}(\Lambda)$ is the set of all events containing at most $d(G)$ addition and corruption faults.

Lemma 3.4. **AC** *is $d(G)$-admissible, continuous and adjacency-preserving.*

Proof. From the definition of **AC**, it follows directly that **AC** is $d(G)$-admissible.

To prove that **AC** is continuous, consider a configuration C and any two events $\tau', \tau'' \in \mathbf{AC}$ for $\Lambda(C)$. Let $m'_1, m'_2, \ldots, m'_{f'}$ be the f' faulty communications in τ', and let $m''_1, m''_2, \ldots, m''_{f''}$ be the f'' faulty communications in τ''. Since **AC** is $d(G)$-admissible, then $f' \leq d(G)$ and $f'' \leq d(G)$. Let $\tau'_0 = \tau'$, and let τ'_h denote the event obtained by replacing the faulty communication m'_h in τ'_{h-1} with a non-faulty one (with the same message sent in both), $1 \leq h \leq f'$; Similarly define τ''_h, $0 \leq h \leq f''$. By construction, $\tau'_{f'} = \tau''_{f''}$. Consider the sequence $\tau'_0, \tau'_1, \ldots, \tau'_{f'} = \tau''_{f''}, \ldots, \tau''_1, \tau''_0$. In this sequence, each event is adjacent to the following one; furthermore, since by construction each event contains at most $d(G)$ additions and/or corruptions, it is in **AC**. Thus, **AC** is continuous.

For any two h-adjacent configurations A and B, consider the events π_h and ρ_h for $\Lambda(A) = \{\alpha_{ij}\}$ and $\Lambda(B) = \{\gamma_{ij}\}$, where for all $(x_i, x_j) \in E$

$$\pi_h[i, j] = \begin{cases} (\alpha_{ij}, \gamma_{ij}) & \text{if } i = h \text{ and } \alpha_{ij} = \Omega \\ (\alpha_{ij}, \alpha_{ij}) & \text{otherwise} \end{cases}$$

and

$$\rho_h[i,j] = \begin{cases} (\gamma_{ij}, \alpha_{ij}) & \text{if } i = h \text{ and } \alpha_{ij} \neq \Omega \\ (\gamma_{ij}, \gamma_{ij}) & \text{otherwise} \end{cases}$$

It is not difficult to verify that $\pi_h, \rho_h \in \mathbf{AC}$ and the configurations $\pi_h(C')$ and $\rho_h(C'')$ are h-adjacent. Hence **AC** is adjacency-preserving. ■

Then, by Theorem 3.2, it follows that

Theorem 3.4. *No k-agreement protocol $\mathcal{P}$ is correct in spite of $d(G)$ communication faults for* **AC** *for $k > \lceil n/2 \rceil$.*

3.5. Impossibility: Byzantine faults

We show that no strong majority protocol is correct in spite of $\lceil d(G)/2 \rceil$ arbitrary communication faults.

For a message matrix $\Lambda = (\alpha_{ij})$, let $\mathbf{ACO}(\Lambda)$ be the set of all events τ on Λ containing at most $\lceil d(G)/2 \rceil$ faulty communications. Then $\mathbf{ACO} := \bigcup_{\Lambda} \mathbf{ACO}(\Lambda)$ is the set of all events containing at most $\lceil d(G)/2 \rceil$ communication faults, where the faults may be omissions, corruptions and additions.

Lemma 3.5. **ACO** *is $\lceil d(G)/2 \rceil$-admissible, continuous and adjacency-preserving.*

Proof. From the definition of **ACO**, it follows directly that **ACO** is $\lceil d(G)/2 \rceil$-admissible.

To prove that **ACO** is continuous, consider a configuration C and any two events $\tau', \tau'' \in \mathbf{ACO}$ for $\Lambda(C)$. Let $m'_1, m'_2, \ldots, m'_{f'}$ be the f' faulty communications in τ', and let $m''_1, m''_2, \ldots, m''_{f''}$ be the f'' faulty communications in τ''. Since **ACO** is $\lceil d(G)/2 \rceil$-admissible, then $f' \leq \lceil d(G)/2 \rceil$ and $f'' \leq \lceil d(G)/2 \rceil$. Let $\tau'_0 = \tau'$, and let τ'_h denote the event obtained by replacing the faulty communication m'_h in τ'_{h-1} with a non-faulty one (with the same message sent in both), $1 \leq h \leq f'$. Similarly, define τ''_h, $0 \leq h \leq f''$. By construction, $\tau'_{f'} = \tau''_{f''}$. Consider the sequence $\tau'_0, \tau'_1, \ldots, \tau'_{f'} = \tau''_{f''}, \ldots, \tau''_1, \tau''_0$. In this sequence, each event is adjacent to the following one; furthermore, since by construction each event contains at most $\lceil d(G)/2 \rceil$ omissions, additions and/or corruptions, it is in **ACO**. Thus, **ACO** is continuous.

Given any two h-adjacent configurations A and B, consider the events π_h and ρ_h for $\Lambda(A) = \{\alpha_{ij}\}$ and $\Lambda(B) = \{\gamma_{ij}\}$, respectively, where for all $(x_i, x_j) \in E$

$$\pi_h[i,j] = \begin{cases} (\alpha_{ij}, \gamma_{ij}) & \text{if } i = h \text{ and } j \in \{j_{\lceil d(h)/2 \rceil + 1}, \ldots, j_{d(h)}\} \\ (\alpha_{ij}, \alpha_{ij}) & \text{otherwise} \end{cases}$$

and

$$\rho_h[i,j] = \begin{cases} (\gamma_{ij}, \alpha_{ij}) & \text{if } i = h \text{ and } j \in \{j_1, \ldots, j_{\lceil d(h)/2 \rceil}\} \\ (\gamma_{ij}, \gamma_{ij}) & \text{otherwise} \end{cases}$$

where $d(h)$ denotes the degree of p_h and $\{j_1, j_2, \ldots, j_{d(h)}\}$ are the indices of the neighbours of p_h. Clearly the configurations $\pi_h(A)$ and $\rho_h(B)$ are h-adjacent; furthermore, since $d(h) \leq d(G)$ and both π_h and ρ_h contain at most $\lceil d(h)/2 \rceil$ faults, then $\pi_h, \rho_h \in \mathbf{ACO}$. Hence **ACO** is adjacency-preserving. ■

Then, by Theorem 3.2, it follows that

Theorem 3.5. *No k-agreement protocol P is correct in spite of $\lceil d(G)/2 \rceil$ communication faults in* **ACO** *for $k > \lceil n/2 \rceil$.*

4. Achieving unanimity

In this section we examine the possibility of achieving *unanimity* among the processors in spite of dynamic faults. Surprisingly, this can be achieved in several cases; the exact conditions depend not only on the type and number of faults but also on the edge-connectivity $c(G)$ of G.

In each graph G there are at least $c(G)$ edge-disjoint paths between any pair of nodes. This fact has been used in the component failure model to show that, with enough redundant communications, information can be correctly

propagated in spite of faults and that the processors can reach some form of agreement (e.g. [15,18]). Our results on the possibility of unanimity in spite of a certain amount f of *dynamic* faults also exploit this fact; the value of f clearly depends on the nature of the faults.

In all cases, we will reach unanimity, in spite of f communication faults per clock cycle, by computing the *OR* of the input values and deciding on that value. This is achieved by first constructing (if not already available) a mechanism for correctly broadcasting the value of a bit within a fixed amount of time T in spite of f communication faults per clock cycle. This reliable broadcast, once constructed, is then used correctly to compute the logical *OR* of the input values: all processors with an input value 1 will reliably broadcast their value; if at least one of the input values is 1 (thus, the result of *OR* is 1), then this fact will be communicated to everybody within time T; on the contrary, if all input values are 0 (thus, the result of *OR* is 0), there will be no broadcasts and everybody will be aware of this fact within time T.

The variable T will be called *timeout*. The actual reliable broadcast mechanism will differ, depending on the nature of the faults.

4.1. Possibility: Single type faults

4.1.1. Omissions

Consider the case when the communication errors are just *omissions*. Let $f \leq c(G) - 1$. When broadcasting in this situation, it is rather easy to circumvent the loss of messages. In fact, it suffices for all processors involved, starting from the initiator of the broadcast, to send the same message to the same neighbours for several consecutive time steps. More precisely, consider the following algorithm:

Algorithm *Bcast-Omit*

1. To broadcast in G, node x sends its message at time 0 and continues transmitting it to all its neighbours in each time step until time $T(G) - 1$ (the actual value of the timeout $T(G)$ will be determined later).
2. A node y receiving the message at time $t < T(G)$ will transmit the message to all its other neighbours in each time step until time $T(G) - 1$.

It is not difficult to verify that for $T(G)$ large enough (e.g. $T(G) \geq c(G)\ (n-2)\ + 1$), protocol *Bcast-Omit* broadcast is allowed in spite of $f \leq c(G) - 1$ omissions. Let us denote by $T^*(G)$ the *minimum* timeout value ensuring that the broadcast is correctly performed in G in spite of $c(G) - 1$ omissions per clock cycle. Then, using *Bcast-Omit* to compute the *OR* we have:

Theorem 4.1. *Let the system faults be* omissions*. Unanimity can be reached in spite of $f = c(G) - 1$ faults per clock cycle in time $T = T^*(G)$ transmitting at most $2m(G)\ T^*(G)$ bits.*

Where $m(G) = |E(G)|$ denotes the number of links in G. As for estimates on the actual value of $T^*(G)$, as already mentioned, it is easy to verify that

Lemma 4.1. $T^*(G) \leq c(G)\ (n-2)\ + 1$.

This value for the timeout is rather high, and depending on the graph G can be substantially reduced. Currently, the best available bound is [6]:

Lemma 4.2. $T^*(G) = O(D(G)^{c(G)})$

where $D(G)$ is the diameter of the graph G. Which estimate is better (i.e. smaller), depends on the graph G. For example, in a *hypercube* H, $c(H) = diam(H) = \log n$; hence if we use Lemma 4.1 we have $O(n \log n)$ while with Lemma 4.2 we would have a time $O(n^{\log\log n})$.

Better bounds are known for specific networks [9,10,13,14,22,29,35]. Interestingly, in a hypercube, both estimates are far from accurate; in fact, $D(G) + 2$ clock cycles are known to suffice [22]. In other words, with only two time units more than in the fault-free case, broadcast can tolerate up to $\log n - 1$ message losses per time unit.

4.1.2. Additions

Let us consider systems where the faults are *additions*; that is, messages are received although none of them were sent by any authorized user. To deal with additions in a fully synchronous system is possible albeit expensive. Indeed, if each processor transmits to its neighbours at each clock cycle, it leaves no room for additions.

The processors can correctly compute the *OR* using a simple diffusion mechanism in which each processor transmits for the first $T(G) - 1$ time units: initially, a processor sends its value; if during this time it is aware of the existence of a 1 in the system, it will only send 1 from that moment on.

The process clearly can terminate after $T(G) = D(G)$ clock cycles. Hence

Theorem 4.2. *L Let the system faults be* additions*. Unanimity can be reached* regardless of the number of faults *in time* $T = diam(G)$ *transmitting at most* $2m(G)\ D(G)$ *bits.*

Observe that if a spanning-tree $ST(G)$ of G is available, it can be used for the entire computation. In this case, the number of bits is $2(n-1)\ diam(ST(G))$ while time is $diam(ST(G))$.

4.1.3. Corruptions

Surprisingly, if the faults are just *corruptions*, unanimity can be reached *regardless of the number of faults*.

To understand this result, first consider that, since the only faults are corruptions, there are no omissions; thus, any message transmitted will arrive, although its content may be corrupted. Furthermore, there are no additions; thus, only the messages that are transmitted by some processor will arrive. This means that if a processor starts a broadcast protocol, every node will receive a message (although not necessarily the correct one). Notice also that since there are no omissions nor additions, each processor needs to participate in the broadcast (i.e. transmit to its neighbours) only once.

We can use this fact in computing the *OR*. Each processor with an input value 1 starts a broadcast. Regardless of its content, a message will always and only communicate the existence of an initial value 1. A processor receiving a message thus knows that the correct value is 1 regardless of the content of the message, and will forward it to all is neighbours (if it has not already done so). As we already observed, since there are no omissions nor additions, each processor needs to participate in this computation (i.e. transmit to its neighbours) at most once.

If there is an initial value 1, since there are no omissions, all processors will receive a message within time $T(G) = D(G)$. If all initial values are 0, no broadcast is started and, since there are no additions, no messages are received; thus, all processors will detect this situation since they will not receive any message by time $T(G)$.

The resulting protocol yields the following:

Theorem 4.3. *Let the system faults be* corruptions*. Unanimity can be reached* regardless of the number of faults *in time* $T = D(G)$ *transmitting at most* $2\ m(G)$ *bits.*

4.2. Possibility: Composite faults

4.2.1. Omissions and corruptions

If the system suffers from *omissions and corruptions*, the situation is fortunately no worse than that of systems with only omissions.

Since there are no additions, no unintended message is generated. Indeed, in the computation of the *OR*, the only intended messages are those originated by processors with an initial value 1 and only those messages (possibly corrupted) will be transmitted along the network.

A processor receiving a message thus knows that the correct value is 1, regardless of the content of the message. If we use *Bcast-Omit*, we are guaranteed that everybody will receive a message (regardless of its content) within $T = T^*(G)$ clock cycles in spite of $c(G) - 1$ or fewer omissions, iff at least one originated (i.e. if there is at least one processor with initial value 1). Hence

Theorem 4.4. *Unanimity can be reached in spite of* $f = c(G) - 1$ *faults per clock cycle if the system faults are* omissions and corruptions*. The time to agreement is* $T = T^*(G)$ *and the number of bits is at most* $2\ m(G)\ T^*(G)$.

Observe that, although expensive, it is no more so than what we have been able to achieve with just omissions.

4.2.2. Omissions and additions

In the case of systems with *omissions and additions*, consider the following strategy.

To counter the negative effect of additions, each processor transmits to all its neighbours in every clock cycle. Initially, a processor sends its value; if at any time it is aware of the existence of a 1 in the system, it will send only 1 from that moment on. Since there are no corruptions, the content of a message can be trusted.

Clearly, with such a strategy, no additions can ever take place. Thus, the only negative effects are due to omissions; however, if $f \leq c(G) - 1$, omissions can not stop the nodes from receiving a 1 within $T = T^*(G)$ clock cycles if at least one processor has such an initial value. Hence

Theorem 4.5. *Unanimity can be reached in spite of* $f = c(G) - 1$ *faults per clock cycle if the system faults are* omissions and additions. *The time to agreement is* $T = T^*(G)$ *and the number of bits is at most* $2\, m(G)\, T^*(G)$.

4.2.3. Additions and corruptions

Consider the environment when faults can be both *additions and corruptions*. In this environment messages are not lost but none can be trusted; in fact the content could be incorrect (i.e. a corruption) or it could be a fake (i.e. an addition).

This makes the computation of *OR* quite difficult. If we only transmit when we have 1 (as we did with only *corruptions*), how can we trust that a received message was really transmitted and not caused by an addition ? If we always transmit the *OR* of what we have and receive (as we did with only *additions*), how can we trust that a received 1 was not really a 0 transformed by a corruption?

For this environment, indeed we need a more complex mechanism employing several techniques, as well as knowledge of the network G by the processors.

The first technique we use is that of *time slicing* [35]:

Technique *Time Slice*:

1. We distinguish between *even* and *odd* clock ticks; an even clock tick and its successive odd tick constitute a *communication cycle*.
2. To broadcast 0 (resp. 1), x will send a message to all its neighbours only on *even* (resp., *odd*) clock ticks.
3. When receiving a message at an *even* (resp., *odd*) clock tick, processor y will forward it only on *even* (resp., *odd*) clock ticks.

In this way, processors are going to propagate 1 only at odd ticks and 0 at even ticks.

This technique, however, does not solve the problem created by additions; in fact, the arrival of a fake message created by an addition at an odd clock tick can generate an unwanted propagation of 1 in the systems through the odd clock ticks.

To cope with the presence of additions, we use another technique based on the edge-connectivity of the network. Consider a processor x and a neighbour y. Let $SP(x, y)$ be the set of the $c(G)$ shortest disjoint paths from x to y, including the direct link (x, y); see Fig. 3. To communicate a message from x to y, we use a technique in which the message is sent by x simultaneously on all the paths in $SP(x, y)$. This technique, called *Reliable Neighbour Transmission*, is as follows:

Technique *Reliable Neighbour Transmission*

1. For each pair of adjacent processors x, y and paths $SP(x, y)$, every processor determines in which of these paths it resides.
2. To communicate a message M to neighbour y, x will send along each of the $c(G)$ paths in $SP(x, y)$ a message, containing M and the information about the path, for t consecutive communication cycles (the value of t will be discussed later).
3. A processor z on one of those paths, upon receiving in communication cycle i a message for y with the correct path information, will forward it only along that path for $t - i$ communication cycles. A message with incorrect path information will be discarded.

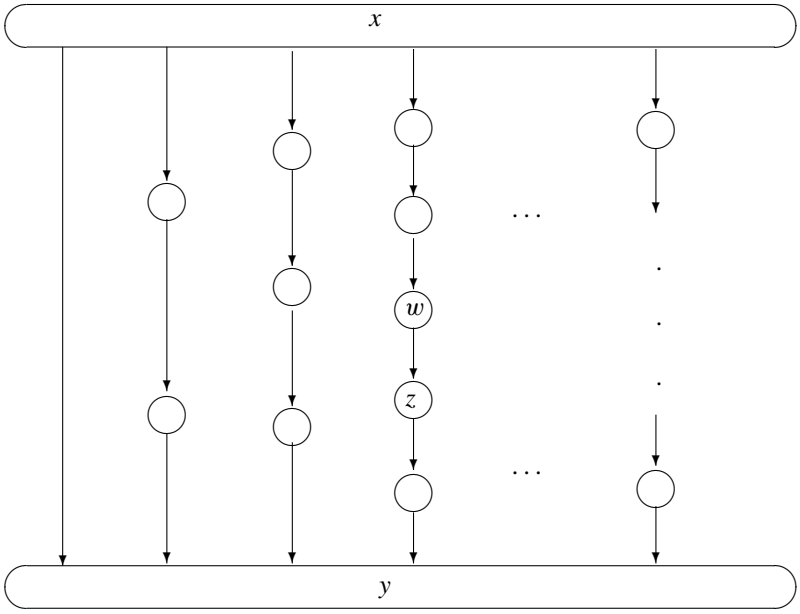

Fig. 3. The $c(G)$ edge-disjoint paths in $SP(x, y)$.

Note that incorrect path information (due to corruptions and/or additions) in a message for y received by z is *detectable* and so is incorrect timing since

- because of local orientation, z knows the neighbour w from which it receives the message;
- z can determine if w is really its predecessor in the claimed path to y;
- z knows at what time such a message should arrive if really originated by x.

Let us now combine these two techniques. To compute the *OR*, all processors broadcast their input value using the *Time Slice* technique: the broadcast of 1's will take place at odd clock ticks, that of 0's at even ones. However, every step of the broadcast, in which every involved processor sends the bit to its neighbours, is done using the *Reliable Neighbour Transmission* technique. This means that each step of the broadcast now takes t communication cycles. Let us call *OR-AddCorrupt* the resulting protocol.

Since there are no omissions, any transmitted message is possibly corrupted but it arrives; the clock cycle in which it arrives at y will indicate the correct value of the bit (even cycles for 0, odd for 1). Therefore, if x transmits a bit, y will eventually receive one and be able to decide the correct bit value. This is, however, not sufficient. We need now to choose the appropriate value of t so that y will not mistakenly interpret the arrival of bits due to additions, and will be able to decide if they were really originated by x.

The obvious property of *Reliable Neighbour Transmission* is that

Lemma 4.3. *In t communication cycles at most $f\ t$ copies of incorrect messages arrive at y.*

The other property of *Reliable Neighbour Transmission* is less obvious. Observe that, when x sends 1 to neighbour y using *Reliable Neighbour Transmission*, y will receive many copies of this "corrected" (i.e. corrected using the properties of time slicing) bit. Let $l(x, y)$ be the maximum length of the paths in $SP(x, y)$; and let $l = \max\{l(x, y) : (x, y) \in E\}$ be the largest of such lengths over all pairs of neighbours. Then

Lemma 4.4. *y will receive at least $(l - 1) + c(G)(t - (l - 1))$ corrected copies of the bit $b_{x,y}$ from x within $t > l$ communication cycles.*

Proof. Let $b_{x,y} = 1$ (respectively, $b_{x,y} = 0$). For the first $l - 1$ odd (respectively, even) clock ticks y will receive the corrected copy of $b_{x,y}$ through link (x, y). During this time, the corrected copy of $b_{x,y}$ will travel down each of

the other $c(G) - 1$ disjoint paths in $SP(x, y)$, one link forward at each odd (respectively, even) clock tick. Since the paths in $SP(x, y)$ have length at most l, from the l-th communication cycle onward, y will receive the corrected copy of $b_{x,y}$ from all the $c(G)$ disjoint paths in $SP(x, y)$ at each odd (respectively even) clock tick. Thus, after $t > l$ communication cycles, y will receive at least $l - 1 + c(G)(t - (l - 1))$ corrected copies of $b_{x,y}$. ■

Processor y can determine the original bit sent by x provided that the number of corrected copies received is greater than the number of incorrect ones.

Lemma 4.5. *After $t > (c(G) - 1)(l - 1)$ communication cycles, y can determine $b_{x,y}$.*

Proof. For the number $(l - 1) + c(G)(t - (l - 1))$ of corrected copies to be larger than the number $(c(G) - 1)t$ of incorrect ones received as a result of additions, it is enough to request $t > (c(G) - 1)(l - 1)$. By Lemmas 4.3 and 4.4, y can identify and reject incorrect messages. ■

Consider that broadcast requires $D(G)$ steps, each requiring t communication cycles, each composed of two clock ticks. Hence

Lemma 4.6. *Using Algorithm* OR-AddCorrupt*, it is possible to compute the OR of the input value in spite of $c(G) - 1$ additions and corruptions in time at most $2D(G)$ $(c(G) - 1)(l - 1)$.*

Hence, unanimity can be guaranteed if at most $c(G) - 1$ additions and corruptions occur in the system:

Theorem 4.6. *Let the system faults be* additions and corruptions*. Unanimity can be reached in spite of $f = c(G) - 1$ faults per clock cycle; the time is $T \leq 2\ D(G)\ (c(G) - 1)\ (l - 1)$ and the number of bits is at most $4m(G)(c(G) - 1)(l - 1)$ messages.*

4.2.4. Byzantine faults

In case of *Byzantine* faults, any type of fault can occur: omissions, additions and corruptions. Nevertheless, using a simpler mechanism than that for additions and corruptions we are able to achieve consensus, albeit tolerating fewer $(f = \lceil c(G)/2 \rceil - 1)$ faults per clock cycle.

To broadcast, we use precisely the technique *Reliable Neighbour Transmission* introduced to deal with additions and corruptions; we do *not*, however, use time slicing: this time, a communication cycle lasts only one clock cycle; that is, any received message is forwarded along the path immediately.

The decision process (i.e. how y, out of the possibly conflicting received messages, determines the correct content of the bit) is according to the simple rule:

Acceptance Rule

y selects as correct the bit value received most often during the t time units.

To see why the technique *Reliable Neighbour Transmission* with this *Acceptance Rule* will work, let us first pretend that no faults occur. If this is the case, then in each of the first $(l - 1)$ clock cycles, a message from x will reach y through the direct link between x and y. In each later clock cycle out of the t cycles, a message from x to y will reach y on each of the at least $c(G)$ paths. This amounts to a total of at least $(l - 1) + c(G)(t - (l - 1))$ messages arriving at y if no fault occurs.

However, as we know, there can be up to $t(\lceil c(G)/2 \rceil - 1)$ faults in these t cycles. This leaves us with a number of correct messages that is at least the difference between both quantities. If the number of correct messages is larger than the number of faulty ones, the *Acceptance Rule* will decide correctly. Therefore, we need that $(l - 1) + c(G)(t - (l - 1)) > 2t(\lceil c(G)/2 \rceil - 1)$. This is satisfied for $t > (c(G) - 1)(l - 1)$. We therefore obtain:

Lemma 4.7. *Communication to a neighbour using* Reliable Neighbour Transmission *tolerates $\lceil c(G)/2 \rceil - 1$ Byzantine communication faults per clock cycle, and uses $(c(G) - 1)(l - 1) + 1$ clock cycles.*

Consider that broadcasting requires $D(G)$ rounds of Reliable Neighbour Transmission. Hence

Theorem 4.7. *Let the system faults be* arbitrary*. Unanimity can be reached in spite of $f = \lceil c(G)/2 \rceil - 1$ faults per clock cycle; the time is at most $T \leq D(G)\ ((c(G) - 1)\ (l - 1) + 1)$.*

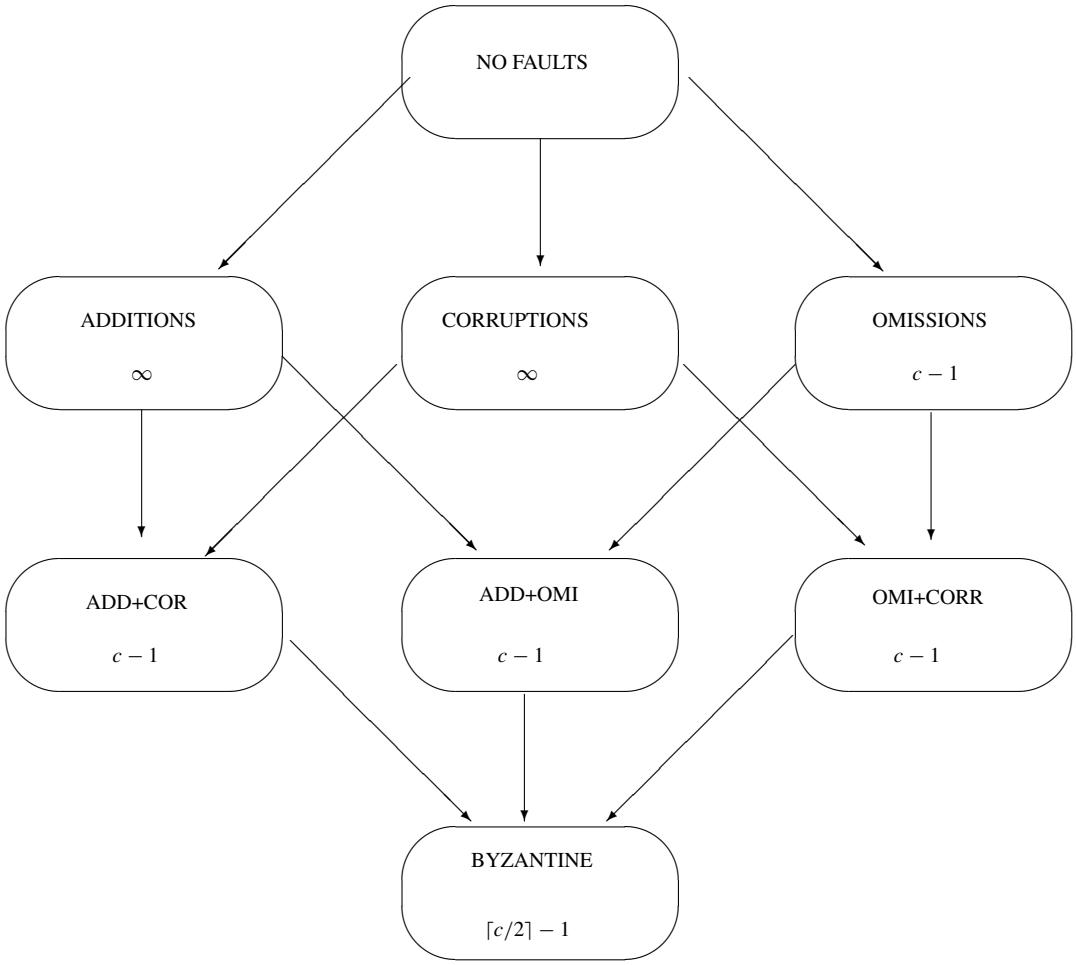

Fig. 4. Maximum number of faults per clock cycle in spite of which *unanimity* is possible.

4.3. Tightness of bounds

The results of this section together with those of Section 3 show that the established bounds for agreement protocols are indeed *tight* for those graphs G where $c(G) = d(G)$. In fact, in this case:

1. With the number of faults (or more) specified by the impossibility bound, even strong majority is impossible.
2. With one less fault than specified by the impossibility bound, even unanimity can be reached.
3. Any agreement among less than a strong majority of the processors can be reached without any communication.

This large class of networks includes hypercubes, toruses, rings, complete graphs, etc. As a consequence, we also close the existing gap in [34,35] between possibility and impossibility for non-trivial agreement with dynamic Byzantine faults in complete graphs.

5. Concluding remarks

We have employed a failure model for synchronous distributed systems, the *communication failure* model, originally introduced in [34]. This model allows ubiquitous transient failures to be represented simply and explicitly, avoiding many of the unwanted conclusions derivable by the component models; furthermore, it has allowed us to focus on the inherent complexity of computing with dynamic faults.

As already observed, localized and permanent faults can be described easily in terms of communication failures, with enough dynamic communication faults of the appropriate type. Indeed, most component failure models can be seen as a special *localized* case of the communication failure model, where all the faults are restricted to the communications involving a fixed (though, a-priori unknown) set of processors or of links. Because of this fact, traditional possibility/impossibility results in the component failure models immediately carry over, as special *localized* instances, to the communications failure model. The only proviso is that, in this model, processors are never faulty: only their actions (i.e. communication with the neighbours) possibly are.

Using the communication failure model, we have established bounds for arbitrary graphs on the impossibility of non-trivial agreement in the presence of dynamic faults.

For graphs whose connectivity is the same as the degree, we have drawn a precise map of safe and unsafe computations in the presence of dynamic faults, generalizing the existing results for complete graphs.

For those graphs where $c(G) < d(G)$, the results established here leave a gap between possibility and impossibility. Closing this gap is the goal of future investigations. Preliminary results indicate that neither parameter provides the "true bound" for arbitrary graphs: there are graphs where $d(G)$ is too large a parameter and there are networks where $c(G)$ is too low. An intriguing open question is whether there actually exists a single parameter for all graphs.

The performance (in terms of time and/or messages) of the proposed protocols for achieving unanimity was not the main concern of this paper. Designing more efficient reliable broadcast protocols would be of considerable interest, both practically and theoretically. Extensive investigations exist in the case of *omissions* (e.g., [6,9,10,13,14,22]). No results, other that those presented here, are known for other types and combinations of fault.

Acknowledgements

The authors would like to thank Friedbert Huber for pointing out Lemmas 4.3 and 4.4; John Dumovich, Andreas Hutflesz, Shay Kutten, Jan van Leeuwen, Jan Pachl, Gabriele Reich, and Shmuel Zaks for helpful discussions on some of these results over the years; and Yoram Moses and Sergio Rajsbaum for bringing reference [34] to the attention of the mainstream distributed computing community.

This work has been carried out in part while the first author has been visiting Carleton University and, non simultaneously, the second author has been visiting ETH Zentrum in Zurich; it has been supported in part by the Natural Sciences and Engineering Research Council and by the Swiss SBF for Project C05.0047 within COST 295 (DYNAMO) of the European Union.

References

[1] M.K. Aguilera, W. Chen, S. Toueg, Failure detection and consensus in the crash-recovery model, Distributed Computing 13 (2) (2000) 99–125.
[2] M.K. Aguilera, S. Toueg, A simple bivalency proof that t-resilient consensus requires $t + 1$ rounds, Information Processing Letters 71 (1999) 155–158.
[3] Z. Bar-Joseph, M. Ben-Or, A tight lower bound for randomized synchronous consensus, in: Proc. 17th ACM Symposium on Principles of Distributed Computing, PODC'98, 1998, pp. 193–199.
[4] M. Ben-Or, D. Ron, Agreement in presence of faults on networks of bounded degree, Information Processing Letters 57 (6) (1996) 329–334.
[5] P. Berman, J. Garay, Cloture votes: $n/4$ resilient distributed consensus in $t + 1$ rounds, Mathematical System Theory 26 (1) (1993) 3–19.
[6] B.S. Chlebus, K. Diks, A. Pelc, Broadcasting in synchronous networks with dynamic faults, Networks 27 (1996) 309–318.
[7] K. Ciebiera, A. Malinovski, Efficient broadcasting with linearly bounded faults', Discrete Applied Mathematics 89 (1998) 99–105.
[8] F. Cristian, H. Aghili, R. Strong, D. Dolev, Atomic broadcast: From simple message diffusion to Byzantine agreement, Information and Computation 118 (1) (1995) 158–179.
[9] G. De Marco, A. Rescigno, Tighter bounds on broadcasting in torus networks in presence of dynamic faults, Parallel Processing Letters 10 (2000) 39–49.
[10] G. De Marco, U. Vaccaro, Broadcasting in hypercubes and star graphs with dynamic faults, Information Processing Letters 66 (1998) 321–326.
[11] S. Dobrev, Communication-efficient broadcasting in complete networks with dynamic faults, in: Proc. 9th Colloquium on Structural Information and Communication complexity, SIROCCO'02, 2002, pp. 101–113.
[12] S. Dobrev, Computing input multiplicity in anonymous synchronous networks with dynamic faults, Journal of Discrete Algorithms 2 (2004) 425–438.
[13] S. Dobrev, I. Vrt'o, Optimal broadcasting in hypercubes with dynamic faults, Information Processing Letters 71 (1999) 81–85.
[14] S. Dobrev, I. Vrt'o, Optimal broadcasting in even tori with dynamic faults, Parallel Processing Letters 12 (2002) 17–22.
[15] D. Dolev, The Byzantine generals strike again, Journal of Algorithms 3 (1) (1982) 14–30.
[16] D. Dolev, M.L. Fischer, R. Fowler, N.A. Lynch, H.R. Strong, Efficient Byzantine agreement without authentication, Information and Control 52 (3) (1982) 256–274.

[17] D. Dolev, H.R. Strong, Polynomial algorithms for multiple processor agreement, in: Proc. 14th ACM Symposium on Theory of Computing, STOC 82, Washington, 1982, pp. 401–407.
[18] C. Dwork, D. Peleg, N. Pippenger, E. Upfal, Fault tolerance in networks of bounded degree, SIAM Journal on Computing 17 (5) (1988) 975–988.
[19] M.J. Fischer, N.A. Lynch, A lower bound for the time to assure interactive consistency, Information Processing Letters 14 (4) (1982) 183–186.
[20] M.J. Fischer, N.A. Lynch, M. Merritt, Easy impossibility proofs for distributed consensus problems, Distributed Computing 1 (1) (1986) 26–39.
[21] M.J. Fischer, N.A. Lynch, M.S. Paterson, Impossibility of distributed consensus with one faulty process, Journal of the ACM 32 (2) (1985).
[22] P. Fraigniaud, C. Peyrat, Broadcasting in a hypercube when some calls fail, Information Processing Letters 39 (1991) 115–119.
[23] J. Garay, Y. Moses, Fully polynomial Byzantine agreement for $n > 3t$ processors in $t + 1$ rounds, SIAM Journal on Computing 27 (1) (1998) 247–290.
[24] L. Gasieniec, A. Pelc, Broadcasting with linearly bounded faults, Discrete Applied Mathematics 83 (1998) 121–133.
[25] R. Guerraoui, R.R. Levy, Robust emulation of shared memory in a crash-recovery model, in: Proceedings of the 24th IEEE International Conference on Distributed Computing Systems, ICDCS 04, 2004, pp. 400–407.
[26] V. Hadzilacos, Connectivity requirements for Byzantine agreement under restricted types of failures, Distributed Computing 2 (1987) 95–103.
[27] R. Kralovic, R. Kralovic, P. Ruzicka, Broadcasting with many faulty links, in: Proc. 10th Colloquium on Structural Information and Communication Complexity, SIROCCO'03, 2003, pp. 211–222.
[28] L. Lamport, R. Shostak, M. Pease, The Byzantine generals problem, ACM Transactions on Programming Languages and Systems 4 (3) (1982) 382–401.
[29] Z. Liptak, A. Nickelsen, Broadcasting in complete networks with dynamic edge faults, in: Proc. 4th Int. Conf. on on Principles of Distributed Systems, OPODIS 00, Paris, 2000, pp. 123–142.
[30] Y. Moses, S. Rajsbaum, A layered analysis of consensus, SIAM Journal on Computing 31 (4) (2002) 989–1021.
[31] Tz. Ostromsky, Z. Nedev, Broadcasting a Message in a Hypercube with Possible Link Faults, in: K. Boyanov (Ed.), Parallel and Distributed Processing '91, Elsevier, 1992, pp. 231–240.
[32] M. Pease, R. Shostak, L. Lamport, Reaching agreement in presence of faults, Journal of the ACM 27 (2) (1980) 228–234.
[33] K.J. Perry, S. Toueg, Distributed agreement in the presence of processor and communication faults, IEEE Transactions on Software Engineering SE-12 (3) (1986) 477–482.
[34] N. Santoro, P. Widmayer, Time is not a healer, in: Proc. 6th Ann. Symposium on Theoretical Aspects of Computer Science, STACS 89, Paderborn, in: LNCS, vol. 349, 1989, pp. 304–313.
[35] N. Santoro, P. Widmayer, Distributed function evaluation in the presence of transmission faults, in: Proc. International Symposium on Algorithms, SIGAL 90, Tokyo, in: LNCS, vol. 450, 1990, pp. 358–367.
[36] U. Schmid, B. Weiss, Formally verified Byzantine agreement in presence of link faults, in: Proc. 22nd Int. Conf. on Distributed Computing Systems, ICDCS 02, Vienna, 2002, pp. 608–616.