DIPLOMARBEIT

Byzantine Agreement Under the Perception-Based Fault Model

carried out at the Department of Automation
of the Vienna University of Technology

under guidance of

ao.Univ. Prof. Dr. Ulrich Schmid

by

Martin Biely

Matr.Nr. 9526482

Anton Schurzgasse 13
3400 Klosterneuburg

1st February 2002

Prediction 64

The next 100 years will be a search for better perception instead of better vision.

— Scott Adams, The Dilbert Future.

Abstract

The Byzantine Agreement problem is widely known as a fundamental problem in fault-tolerant distributed computing. It was introduced by Lamport, Shostak and Pease in 1980 [43]. In 1978 [38] Gray proved that it is impossible to solve the related Coordinated Attack problem when links may be faulty. However, Schmid and Weiss showed in [61] how to avoid this result by limiting the uncertainty that faulty links can generate in each round.

Usually very simple fault models are used in the analysis of distributed algorithms, but by refining the model through classifying faults the number of necessary processors can be reduced tremendously. To account for processor and link faults in a uniform way Schmid [59] developed the *perception-based fault model*, which he and Weiss later refined for use with Byzantine agreement protocols. [61]

Along with stating the problem Lamport, Shostak and Pease also presented an algorithm for solving Consensus. Unfortunately this algorithm needs messages of exponential size. Later research has produced polynomial protocols, three of which I adapted to overcome link faults. These protocols are all presented in this thesis and have also been submitted to a renowned computer science journal.

Zusammenfassung

Das "Byzantine Agreement" Problem ist eines der fundamentalen Probleme im Gebiet der fehlertoleranten verteilten Computer-Systeme. Es wurde erstmals 1980 von Lamport, Shostak und Pease in [43] vorgestellt. Für ein verwandtes Problem zeigte Gray schon 1978 , dass es nicht deterministisch lösbar ist, wenn die Netzwerkverbindungen fehlerhaft sein können. Schmid und Weiss zeigten in [61], dass es möglich ist, dieses Ergebnis zu umgehen, indem man die Unsicherheit, die das defekte Netzwerk in einer Runde eines synchronen Algorithmus verursachen kann, beschränkt.

Üblicherweise werden bei der Analyse von verteilten Algorithmen nur sehr einfache Fehlermodelle eingesetzt. Es ist allerdings durch Klassifizierung von Fehlern möglich, einen Algorithmus genauer zu untersuchen und so die Anzahl der benötigten Knoten im System erheblich zu verringern. Solche Fehlermodelle werden als *hybrid* bezeichnet. Das von Schmid in [59] entwickelte *perception-based fault model* erlaubt Fehler von Knoten und Verbindungen in einer einheitlichen Art und Weise zu modellieren. [61]

Lamport, Shostak und Pease haben neben der Beschreibung des Problems auch gleich eine Lösung vorgestellt, allerdings benötigt diese Nachrichten von exponentieller Größe. Erst später wurden Protokolle mit polynomischem Aufwand entwickelt, von denen ich drei unter dem neuen Fehlermodell analysiert habe. Die Ergebnisse dieser Analyse werden in dieser Arbeit vorgestellt, und wurden zur Veröffentlichung in einem renommierten Fachjournal eingereicht.

Acknowledgments

I thank Ulrich Schmid for his expert guidance and his great support, and Bettina Weiss for reading and finding errors.

This research is part of and was supported by the W2F-project, which targets a wireline/wireless fieldbus based upon spread-sprectrum (CDMA) communications. W2F is supported by the Austrian START programme Y41-MAT, see `http://www.auto.tuwien.ac.at/Projects/W2F/`.

Contents

1 Introduction

The Byzantine Agreement problem is widely known as a fundamental problem in fault-tolerant distributed computing. This thesis aims at solving this problem with polynomial sized messages, even in the presence of link faults. It is organized as follows:

This section aims at giving a general background on distributed systems and the spectrum of this field covered in this thesis.

Section 2 presents the *Byzantine agreement* and *Consensus* problems. These are important problems in the field of fault-tolerant distributed systems. After a historical overview of the research on this problem (Section 2.3), I will present the formal definition of the problem in Section 2.4. Section 2.5 discusses the straightforward solution that Lamport, Shostak and Pease presented in [43].

Section 3 is devoted to different fault models. First we will survey different hybrid fault models, then fault models that include link faults will be studied. Finally, we will relate those to our perception-based fault model.

In Section 4 I present three different solutions for the *Binary* Byzantine agreement problem in the presence of link faults. A discussion of the conclusions and hints on possible future work given in Section 5 eventually concludes the thesis.

1.1 Distributed Systems

Attiya and Welch [4] define a distributed system as follows:

> "A *distributed system* is a collection of individual computing devices that can communicate with each other. This very general definition encompasses a wide range of modern day computer systems, ranging from a VLSI chip, to a tightly-coupled shared memory computer multiprocessor, to a local-area cluster of workstations, to the Internet." [4]

One aspect that is obvious is that distributed systems are omnipresent in today's world. Even the CPU of a stand-alone home-computer

	MP	SM
synch	yes	no
asynch	no	no

Figure 1. *Classification of systems in the scope of this work*

may be seen as a distributed system of computing devices, ranging from the arithmetic integer unit to the on-chip floating point or multimedia co-processors (i.e., the MMX extension and the like), which all share some resources like the memory interface or the cache.

It is easy to see that it is very difficult to argue about the whole range of this definition. Just consider communicating some value from one computing device to another. This is fairly simple in a processor or shared memory system, compared to the overhead necessary to accomplish the same in a Wide Area Network, say the Internet. Another point where the systems within this definition obviously differ is the notion of time: While the computing devices on one chip are usually driven by the same clock signal, i.e., operate in lock-step, this is obviously not true in a local area network of workstations.

Distributed systems are hence classified according to the following two properties:

- timing: asynchronous vs. synchronous
- communication: message-passing (MP) vs. shared-memory (SM)

Figure 1 shows how distributed systems may be classified and which class is in the scope of this thesis. A tightly coupled shared memory computer system would be in the top right corner of the figure. The distributed systems in the scope of this thesis are the synchronous, message-passing systems in the top left corner.

Important reasons for employing a distributed system are the possibility to share resources and the increase in speed and fault tolerance. However, there are also disadvantages arising from the inherent uncertainty, which originate from

- differing processor speeds
- varying communication delays
- failures of both links and computing devices

The synchronous distributed systems I am talking about in this thesis are based on the assumption that the first two of these can be reasonably bounded, which implies that the protocols presented only have to deal with failure of links and nodes. It is thus important to use a realistic and flexible system model.

This thesis considers a synchronous distributed system of n computing nodes connected by a point-to-point network. The topology of the network is assumed to be a fully connected graph (this assumption can be relaxed however, see Remark (SM10) on page 25). The nodes communicate by passing messages. The fault model, which is an important part of the system model, will be discussed in Section 3.3.

In the remainder of this document we will call the computing devices from the definition above *processors* for short, which is not meant to limit the meaning.

2 The Byzantine Agreement Problem

Let us assume that you have a number of processors and that all receive a value from a dedicated transmitter, e.g. an intelligent sensor. A Byzantine agreement algorithm can be used to ensure that all (non-faulty) processors obtain the same value. This greatly simplifies the design and implementation of replicated services or more generally a distributed computation.

2.1 Byzantine Generals

The Byzantine agreement problem can be explained in an informal way by using the *Byzantine Generals* metaphor [44]:

Consider a town under siege: Usually the town is surrounded by the enemy army commanded by a "Commanding General". The army is divided into battalions, headed by "Lieutenant Generals". This situation is depicted in Figure 2.

The town has a very good defence system, and thus can only be taken if sufficiently many battalions attack simultaneously. Unfortunately for the attacking army there are some dishonest generals that are more loyal to money than to their army, that is why the mayor of the town was able to bribe them, consequently they will not attack and in fact they will try to confuse the faithful generals. This kind of behavior is called *Byzantine faulty*, hence the name of the metaphor. The generals know that some of the generals may be "faulty", but they do not know which ones they are. Naturally, the Lieutenant Generals wait to get orders from the Commanding General, such as "attack" or "wait". Since even the Commanding General could be faulty the Lieutenants have to agree upon a common order of what to do.

2.2 Related Problems

A problem closely related to Byzantine agreement is called *consensus* or *interactive consistency*. With this problem there is no dedicated transmitter, but every processor has some private input value, often

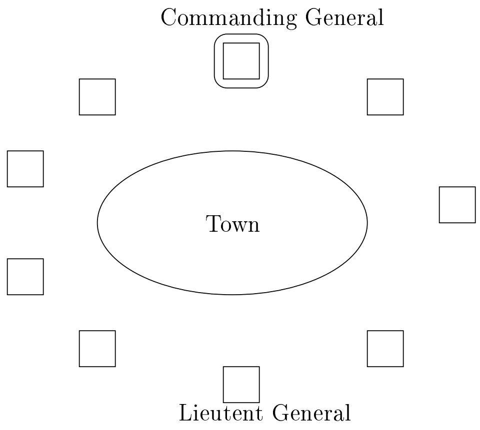

Figure 2. *A town under siege*

referred to as *initial value*, or *initial preference* and the goal of any consensus protocol is for all processors to agree on one value $v \in \mathcal{V}$. Consensus is a symmetric problem, whereas Byzantine agreement is asymmetric.

Note that any solution to one of the problems can be used to solve the other: To solve Byzantine agreement let the transmitter send its message and then run consensus using the received message as initial value. On the other hand consensus can be solved by letting each processor play the role of the transmitter in one instance of a Byzantine agreement protocol and then using one voting-function to determine the final value.

If only two values are allowed, i.e., $\mathcal{V} = \{0, 1\}$, the problem is called *binary agreement*.

Lamport [42] defined a variant called *weak Byzantine agreement*, which will be shortly discussed in Section 2.4. Another weaker form of agreement, which requires that there are at most k different values in the set of results of the processors, is called "k-set agreement", and was first introduced by Chaudhuri [20].

Another related problem is *approximate agreement*. In this problem the processors do not start with discrete input values, but with real ones. Instead of having to agree exactly, as in the "ordinary" Byzantine agreement problem, the requirement is relaxed such that the results have to be within a certain range ϵ. Much research on this subject has been conducted, e.g. in the clock synchronization context, [47, 27, 30, 31, 3].

Another agreement problem with slightly different semantics is the *distributed commit* problem, also known as *coordinated attack* problem which is a key problem in *distributed databases*. Informally speaking the problem is whether to *commit* or *abort* a transaction, the difficulty lies in the fact that if one processor starts with *abort* as its preference, then *abort* is the only legal outcome. Also agreement has to be reached by all processors, not just by non-faulty ones.[1] A discussion of this problem can be found in any decent book on databases and database theory such as [18].

2.3 Literature Overview

The Byzantine agreement (BA) problem was first posed by Lamport, Shostak and Pease in 1980 [43]. In the same paper the authors also presented a protocol that solves this problem in $f+1$ rounds provided $n > 3f$, where n is the number of processors and f an upper bound on the number of Byzantine faulty processors in any execution of the protocol. They also proved that no solution for $n \leq 3f$ exists, while Fischer and Lynch later showed that $f+1$ rounds are necessary in the worst case run of any BA protocol [32]. The protocol of [43] has one disadvantage, however: It requires the processors to send messages of exponential size. The design of more efficient protocols has been the subject of much subsequent research.

The first polynomial communication protocol for Byzantine agreement was provided by Dolev and Strong [29] and subsequently improved by Dolev, Fischer, Fowler, Lynch and Strong [25] to yield a round complexity of $2f+3$. At the same time Toueg, Perry and Srikanth provided an identical alternative [70]. Coan [21] presented a family of Byzantine agreement protocols for $n > 4f$ that – for every d – halt in $f + (f/d)$ rounds and require messages of size $\mathcal{O}(n^d)$. However, these

[1] It is thus only an interesting problem for crash faults, since arbitrary faulty processors cannot be reasonably restricted in such a way.

Protocol	Year	n	rounds	comm.	comp
LSP [43]	80	$3f+1$	$f+1$	$\exp(n)$	$\exp(n)$
DFFLS [25], TPS [70]	82	$3f+1$	$2f+c$	poly(n)	poly(n)
C [21]	85	$4f+1$	$f+\frac{f}{d}$	$\mathcal{O}(n^d)$	$\exp(n)$
DRS [28],BD [9],C [22]	86	$\Omega(f^2)$	$f+1$	poly(n)	poly(n)
BDDS [10, 8]	87	$3f+1$	$f+\frac{f}{d}$	$\mathcal{O}(n^d)$	$\mathcal{O}(n^d)$
MW [50]	88	$6f+1$	$f+1$	poly(n)	poly(n)
BGP [16]	89	$3f+1$	$f+\frac{t}{d}$	$\mathcal{O}(c^d)$	$\mathcal{O}(c^d)$
BG [15]	89	$4f+1$	$f+1$	poly(n)	poly(n)
CW [23]	90	$\Omega(f\log f)$	$f+1$	poly(n)	poly(n)
BGP [14]	91	$(3+\epsilon)f$	$f+1$	poly$(n)\mathcal{O}(2^{\frac{1}{\epsilon}})$	poly$(n)\mathcal{O}(2^{\frac{1}{\epsilon}})$
GM [34, 35]	93	$3f+1$	$f+1$	poly(n)	poly(n)

Table 1. *History of Byzantine agreement [34]*

protocols require exponential local computation. In 1986 Dolev, Reischuk and Strong presented $(f+1)$-round polynomial time BA protocols for $n=\Omega(f^2)$ [28]. At the cost of requiring $\Omega(f\log f)$ processors, Coan and Welch developed a polynomial protocol that uses 1-bit messages and asymptotically optimal total bit-transfer [23].

The best-known solution is based on the following series of papers: In 1988 Moses and Waarts [50] presented the first polynomial $(f+1)$-round protocol with linear resilience; It required $n>8f$ and was later improved to $n>6f$. Subsequent research by Berman and Garay [15] and Garay and Moses [34] has produced $f+1$ round Byzantine agreement algorithms in polynomial time, i.e., polynomial communication and local computation; the last one also achieves the $n=3f+1$ lower bound on the number of processors. Unfortunately, these algorithms are complicated.

Table 1 summarizes these results.

Dolev [24] considered non-fully connected networks and proved the lower bound $conn(G)>2f$, where G is the network graph and $conn(G)$ denotes the node-connectivity, for the minimal connectivity necessary to solve the consensus problem (cf. Remark (SM10) on page 25).

2.4 Formal Definition

In this section we provide a more formal definition of the Byzantine agreement and consensus problems. Since we are only considering deterministic algorithms for synchronous systems, where every non-faulty processor computes its decision value within a fixed number of rounds, there is an implicit *termination* property in both definitions. Any algorithm that solves consensus or Byzantine agreement has to comply to (C1) and (C2), or (B1) and (B2) respectively.

Consensus assumes, as already mentioned above, that every processor p is provided with some *initial value* $x_p \in \mathcal{V}$. A consensus algorithm will compute a *decision value* v_q at every processor q that satisfies the following properties:

(C1) (*Agreement*): If processors p and q are both non-faulty, then both compute the same value $v_p = v_q$.

(C2) (*Validity*): If all non-faulty processors start with the same initial value $\forall p : x_p = v$, then every non-faulty processor q computes $v_q = v$.

Processors that execute the particular Algorithm faithfully, but fail to communicate their values are called obedient, see Definition 1 (page 21) for details. Therefore it is often possible to show the properties above for obedient faulty processors as well. These variants are called *Uniform Agreement* and *Uniform Validity* respectively, because they treat obedient and non-faulty (which are of couse obedient) in a uniform way.

Byzantine agreement is different from consensus in the sense that there is a dedicated transmitter p_t and agreement is to be reached on the value it sends. Byzantine agreement is an asymmetric problem in the sense that one processor has a different role, whereas Consensus is symmetric since all processors are equal. Thus the Agreement and Validity properties have a slightly different definition:

(B1) (*Agreement*): If two processors p and q, one of which may be the transmitter, are both non-faulty, then both deliver the same $v_p = v_q$.

(B2) (*Validity*): If processor p is non-faulty, the value v_p delivered by p is

- v, if the transmitter is non-faulty,
- E, if the transmitter is manifest faulty,
- v or E, if the transmitter is omission faulty,
- the value actually sent, if the transmitter is symmetric faulty,
- any value including E, if the transmitter is arbitrary faulty.

The Validity property contains a forward reference to our fault model, which will be introduced in Section 3.3. The original property only considers a non-faulty and an arbitrary faulty (i.e., Byzantine faulty) transmitter. By applying the properties to all obedient processors – instead of only non-faulty ones – we can look at the uniform variants of the properties.

The following related problems are provided for reference only, and are not in the scope of this paper.

In [42] Lamport defined a variant of consensus called *Weak Agreement*. It differs from consensus only in the definition of *Validity*:

(W1) same as (C1).

(W2) (*Validity*): If there are no faulty processors and all processors start with the same initial value $\forall p : x_p = v$, then any processor q computes $v_q = v$.

On the other hand one can also tighten the *Validity* property, and require the decision value to be one benign processor's input. This problem is called called *strong agreement*.

Another related problem that is interesting to mention in this context is *k-set agreement*. Its properties require only that there are no more than k different results. The problem can be formalized with these conditions:

(S1) (*Agreement*): There is a subset $\mathcal{W}$ of $\mathcal{V}$, $|\mathcal{W}| = k$, such that all decision values are in $\mathcal{W}$: $\forall p : v_p \in \mathcal{W}$.

(S2) (*Validity*): Any decision value of any process is the initial value of some process.

2.5 Exponential Solution

The solution presented in this section is essentially the same protocol as presented by Pease, Shostak and Lamport [43], but is formulated in a slightly different way. It is widely known as *EIG*-tree protocol and goes back to Bar-Noy, Dolev, Dwork and Strong [10, 8]. It is also the basis of the optimal protocol from [34]. EIG stands for *Exponential Information Gathering*.

The basic data structure used by EIG algorithms is a labelled tree, whose paths from the root to the leaves represent the propagation of the information. It has $f + 2$ levels, the node at level 0, the root, is labelled with λ. Each path is unique and consists of distinct processors' labels, so each node at level $k, 0 \leq k \leq f$ has exactly $n - k$ children. We use $tree_i(x)$ to denote the value that is stored in processor p_i's tree in node x.

Figure 3 shows part of some processor p_i's EIG-tree. The nodes on the first level (beneath the root λ) are created in round 1 and hold the values received from the processor denoted by their label. In the second round the nodes in the second level are added. Each node $p_j p_k$ now holds the value p_k claims to have received from p_j in the first round. Assuming p_2 is a non-faulty processor, $tree_i(p_1 p_2)$ has the value that p_1 sent to p_2 in the first round.

Informally, the algorithm proceeds as follows: In round r each processor sends the values of all nodes at level $r - 1$ in his tree to all other processors.[2] Then the processor receives the values from the other processors and adds them to his tree. After $f + 1$ rounds the processor decides on the value that is delivered by applying the resolve function

[2]The processor's initial value is saved in $tree_i(\lambda)$.

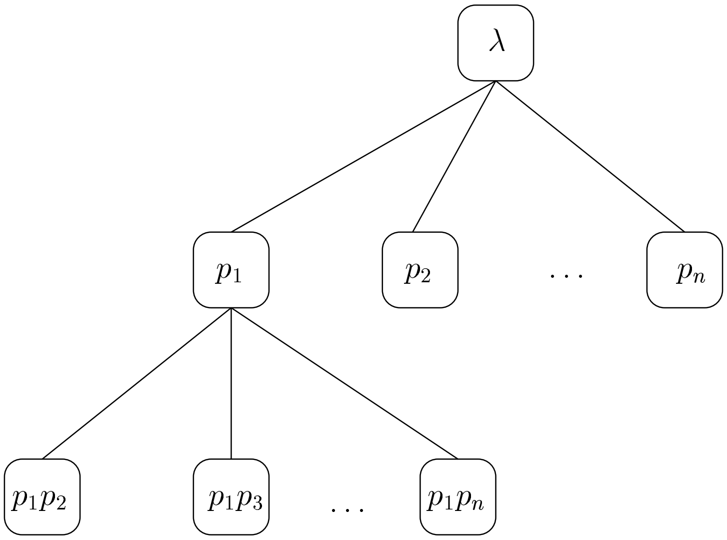

Figure 3. *Part of p_i's EIG-tree*

```
for k:=1 to f + 1 do begin
        send(level(k) of tree) to all processors
        foreach (σ,v) received from p_j do begin
                /* p_j has tree_j(σ) = v */
                tree_i(σp_j) := v
        done
done
decide(resolve_i(λ))
```

Figure 4. *The Exponential Information Gathering protocol*

to the root λ of its tree. The resolve function for processor p_i is defined as:

$$\mathbf{resolve_i}(\sigma) = \begin{cases} tree_i(\sigma), & \text{if } \mathbf{leaf}?(\sigma); \\ majority(resolve_i(\sigma q)), & \text{if majority exists;} \\ 0, & \text{otherwise.} \end{cases}$$

A more formal description of the algorithm is given in Figure 4.

Let me now sketch the proof of correctness for the EIG-protocol as presented by Lynch [46]. The first two Lemmas 1 and 2 are pretty obvious. The first one states that non-faulty processors send the same values to all receivers, and the second that a node x corresponding to a correct processor will $\mathbf{resolve_i}()$ to the value of $tree_i(x)$.

Lemma 1 (6.15 from [46]) *After $f+1$ rounds of the EIG protocol, the following holds. If i, j and k are all non-faulty processes, with $i \neq j$, then $tree_i(x) = tree_j(x)$ for every label x ending in k.* □

Lemma 2 (6.16 from [46]) *After $f+1$ rounds of the EIG protocol the following holds. Suppose that x is a label ending with the index of a non-faulty processor. Then there is a value $v \in \mathcal{V}$ such that $tree_i(x) = resolve_i(x) = v$ for all non-faulty processors i.*

Proof: The proof works by reverse induction on the depth of the node x, i.e., induction from the leaves up to the nodes at level 1. Due to the definition of the $\mathbf{resolve_i}()$ function, the lemma is obvious for the leaves. For the nodes above the leaves, the lemma follows from the fact that there is a majority of non-faulty children among the $n-|x| \geq n-f > 2f$ children of node x, and that the lemma holds for the children due to the inductive hypothesis. □

This suffices to show the *Validity* property, as defined in Section 2.4. Note that we only consider arbitrary faulty processors here.

Lemma 3 (6.17 from [46]) *If all non-faulty processors begin with the same initial value $v \in \mathcal{V}$, then v is the only possible decision value for a non-faulty processor.*

Proof: Since all non-faulty processors p_i start with the same value, say v, Lemma 1 implies that the nodes corresponding to them will have value v, i.e., $tree_j(p_i) = v$, at any non-faulty processor p_j. From this it follows by Lemma 2 that these nodes will also $\mathbf{resolve}()$ to v. Since all non-faulty processors are a majority we know that $\mathbf{resolve}(\lambda) = v$. □

A subset C of the nodes in a tree is a *path covering* if every path from root to leaves contains at least one node in C. The subset C is a *cut* if every path from root to leaves contains exactly one node in C.

A tree node x is said to be *common* in an execution α provided that after $f+1$ rounds of the EIG protocol, all non-faulty processors i have the same $\mathbf{resolve_i}(x)$. A *set of nodes* is *common* if all nodes in the set are common.

Lemma 4 (6.18 from [46]) *After $f+1$ rounds of any execution α of the EIG protocol, there exists a path covering that is common in α.*

Proof: Let C be a set of nodes of the form xi, where i is non-faulty. As observed in Lemma 1, all nodes in C are common. To see why C is a path covering, consider any path from λ to a leaf. It contains exactly $f+1$ non-root nodes, and each of the $f+1$ nodes is unique due to the construction of the tree. Since there are at most f faulty processors, there is some node on the path whose label ends in a non-faulty processor index. This node must be in C. □

Lemma 5 (6.19 from [46]) *After $f+1$ rounds of the EIG protocol, the following holds: Let x be any node label in the EIG tree. If there is a common path covering of the subtree rooted at x, then x is common.*

Proof: By induction on tree labels, working from the leaves up.

Basis: Suppose that x is a leaf. Then the only path covering of x's subtree consists of the single node x itself, which is common. So x is common, as needed.

Inductive Step: Suppose that $|x| = r$, $0 \leq r \leq f$. Suppose that there is a common path covering C of x's subtree. If x itself is in C, then x is common as asserted, so suppose $x \notin C$.

Consider *any* child xl of x. Since $x \notin C$, C induces a common path covering for the subtree rooted at xl. So by inductive hypothesis xl is common. Since xl was chosen to be an arbitrary child of x, all the children of x are common. Then the definition of **resolve**() implies that x is common. □

Lemma 6 (6.20 from [46]) *After $f+1$ rounds of the EIG protocol the root, λ, is common.*

Proof: Due to Lemma 4 there exists a common path covering in the subtree rooted at λ. Lemma 5 implies that this results in λ being common. □

Theorem 1 (6.21) *The EIG protocol solves the consensus problem for n processors with f failures, if $n > 3f$.*

Proof: Validity follows from Lemma 3 and Agreement from 6 and the fact that all non-faulty processors decide on **resolve**(λ) which is common. □

Unfortunately, there is a discouraging general impossibility result for deterministic consensus in presence of link faults, which goes back to Gray's 1978 paper [38] on atomic commitment in distributed databases:

Theorem 2 (Gray's Impossibility [46, Thm. 5.1]) *There is no deterministic algorithm that solves the coordinated attack problem in a synchronous two-processor system with lossy links.*

For that reason this area of research was more or less neglected for twenty years. Consensus in the presence of link faults was only treated with *randomized* algorithms. These suffer from the non-zero probability of non-termination. Lynch [46] gives an overview on this approach.

As Schmid, Weiss and Rushby [61, 62] show, this impossibility result can be circumvented by moderately restricting the inconsistency that link faults may cause system-wide, such that they may only affect a minority (f_ℓ^s) of processors. In the Gray's Theorem 2, there is no point in considering what happens in this case since there is no non-empty minority of processors for $n = 2$ at all.

3 Modelling Faults

One key aspect in the analysis of distributed systems and distributed algorithms is the *fault model* that the analysis is based on. In the traditional fault model at most f of the n processors may be faulty. Traditionally, only two kinds of faults were considered:

A processor experiencing a *crash fault* simply stopped to work, i.e., it does not send any more messages. Before it crashes it behaved correctly, but it may crash within a broadcast, which results in only some processors receiving a message from the crashed processor.

In contrast a *Byzantine* or *arbitrary* faulty processor can change state arbitrarily and send any messages. Arbitrary faulty processors can also send different messages to different processors, and can even cooperate to confuse non-faulty processors.

In Section 3.1 I will summarize the motivation for employing hybrid fault models, and a give a historical overview on the evolution of these models. Section 3.3 will present the *perception-based* fault model. A number of other approaches to deterministically handling link faults in synchronous systems will be presented in Section 3.4.

3.1 Hybrid Fault Models

In reality not all of the faulty processors will behave arbitrary faulty, but will exhibit less severe behavior. This is exploited by hybrid fault models, which benefit from the fact that less severe faults can be handled with fewer processors than more severe ones. For example, it will turn out that masking f symmetric faults requires only $n \geq 2f + 1$ processors, whereas $n \geq 3f + 1$ is needed if all faults are asymmetric (arbitrary) ones. Since a large number of asymmetric faults is quite unlikely in practice, this effectively leads to a smaller n for tolerating a given number of faults. This, in turn, has a positive effect upon dependability by reducing the number n of components that could be faulty, cf. [54]. System designers will hence appreciate our very detailed hybrid fault model (Section 3.3) for getting the maximum fault-tolerance out

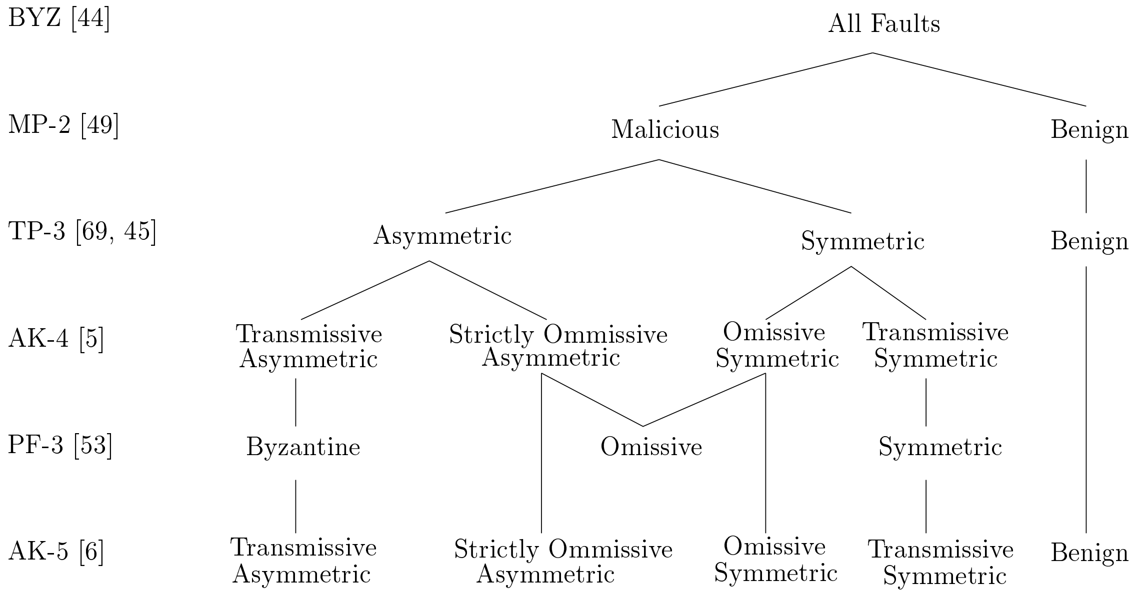

Figure 5. *Evolution of Hybrid Fault Models [6]*

of a given —and usually quite small— n. Of course, analysis is not simplified by using a hybrid fault model.

Figure 5 shows the evolution of hybrid fault models. Meyer and Pradhan were one of the first to introduce a hybrid fault model [49]. They considered two failure modes: *Benign* faults[3] which are evident to all other processors, and *malicious* faults, comprising all other faults.

In 1988 Thambidurai and Park [69] split the group of malicious faults into *symmetric* and *asymmetric* faults. Symmetric faults are perceived identically at all processors, whereas asymmetric (or Byzantine) faulty processors may send different messages to each processor. In the same paper they present an algorithm, called Z, which was thought to solve the consensus problem, until Lincoln and Rushby [45] showed that there are cases in which this is not true.

These or similar models have been successfully applied to a number of other problems in the domain of distributed algorithms [40, 56, 72, 41, 58, 60].

[3]This kind of fault was called *manifest* faulty in later work, e.g. [69, 45, 61], which is also the term used in the rest of this paper.

Dropping the benign failure mode, Plunkett and Fekete [53] introduced a system model with three failure modes. As illustrated in Figure 5 (as PF-3) their model differentiates between asymmetric, omissive and symmetric failures. An omissive faulty processor is one that does not send messages it would be required to send by the protocol. Note that this includes crash faults, which are normally in the class of benign faults. Earlier Azadmanesh and Kieckhofer [5] presented a similar model (AK-4), which is the same except that there are two different omissive failure modes, for symmetric and asymmetric omissions. Later Azadmanesh and Kieckhofer [6] added a class of benign faulty processors. These three models were primarily used to solve Approximate agreement.

3.2 Link Faults

Process fault models usually rest upon the total number of processor faults in the entire system. Given the steadily increasing dominance of communication over computation in modern distributed systems, it becomes increasingly difficult to apply fault models that capture only processor faults.

Due to the high reliability of modern processors, communication-related faults like receiver overruns (run out of buffers), unrecognized packets (synchronization errors), and CRC errors (data reception problems) in high-speed wireline and, in particular, all sorts of wireless networks are increasingly dominating processor faults. As with processor faults there are various ways to deal with of dealing with link faults in one's fault model.

We can differentiate between link faults that cause a link to lose some (or all) messages and link faults that might even change messages. We will call such links *omission* or *value* faulty, respectively. Note that most changes to messages can, of course, be detected by using checksums on the messages. In some applications, however, the probability of messages that are changed in a way that is not detectable may not be neglected.

Another way to differentiate between link faults is to categorize them as either *persistent* or *transient*. Transient faults are the predominant kind, since the problems that may cause link faults (already mentioned above) are usually transient: Receiver overruns, loss of syncronization and CRC errors are usually transient, i.e., they affect only a few messages. Less frequent problems, such as a cut network cable or broken network interfaces are reasons for persistent errors. Although these problems will most likely be repaired at some point, they are persistant in respect to an execution of an agreement algorithm. Note that networks with link faults, which are persistent throughout the execution of the algorithm, are identical to networks whose underlying network graph is not fully connected (cf. Remark (SM10) on page 25).

Likewise link faults can also be modelled in a *static* or *dynamic* way. In a static model the same links are considered faulty throughout an execution. This, however, may not be a practical approach if different links may suffer faults in different rounds, which is usually the case for transient link faults. To resolve this problem one can view link faults in a dynamic way and restrict the number of faulty links per round (instead of per execution). The perception-based fault model (presented in Section 3.3) goes one step further. It models link faults on a per round per receiver/sender basis.

In asynchronous systems link faults are relatively easy to handle if the links satisfy the "fair loss" property: If sending an infinite number of messages over a possibly faulty link results in an infinite number of messages to be received, a perfect (but slow) link can be simulated by suitable retransmission schemes (see, e.g. [12, 2, 73, 39]). This technique cannot be used in synchronous systems without increasing the duration of a round, according to the number of message losses that should be tolerated. This would result in impractically long executions.

3.3 The Perception-Based Fault Model

As the comparison with other models in Section 3.4 will show, link faults were mostly considered as static, i.e., the links which were allowed to be faulty are fixed throughout the whole execution of a given algorithm. However, given the fact that (1) the execution of algorithms, especially those with exponential runtime, can take considerable amounts of time and (2) most link faults are transient this approach requires a much higher number of processors to overcome link faults than necessary.

This shortcoming is resolved by the *perception-based* fault model of [59], which considers link faults as per-round, i.e., in a dynamic way. Thus only the number of transient and permanent link faults that can be perceived by *one processor* in *one round* have to be taken into account. The model also discriminates between different kinds of faults, for both processor and link faults, which allows even better adjustment to one's specific needs, and thus lowers the number of processors even further.

Apart from the general difficulty of modelling link faults, there is also a specific problem related to consensus in presence of link faults: As already mentioned in the well-known impossibility result going back to Gray [38], there is no deterministic algorithm that can solve consensus in presence of lossy links, cf. [46]. As shown in [61, 62] this result can be circumvented if the power of link faults is (moderately) restricted. Two different views, resulting in two constraints, on link faults are required for this purpose:

Broadcast link faults: We restrict the number f_ℓ^s of receivers that could obtain a faulty message in the broadcast of a single sender, see Figure 6.

Receive link faults: We restrict the number f_ℓ^r of senders that could appear faulty to any single receiver, see Figure 7.

Schmid [59] shows that processor and link faults are easily accommodated in a *perception-based fault model*, where the usual global, i.e., system-wide, number of faults is replaced by the number of faults that are observable in the processors' local "perceptions" of the system. Later [61, 62] this model was generalized from the single-round clock

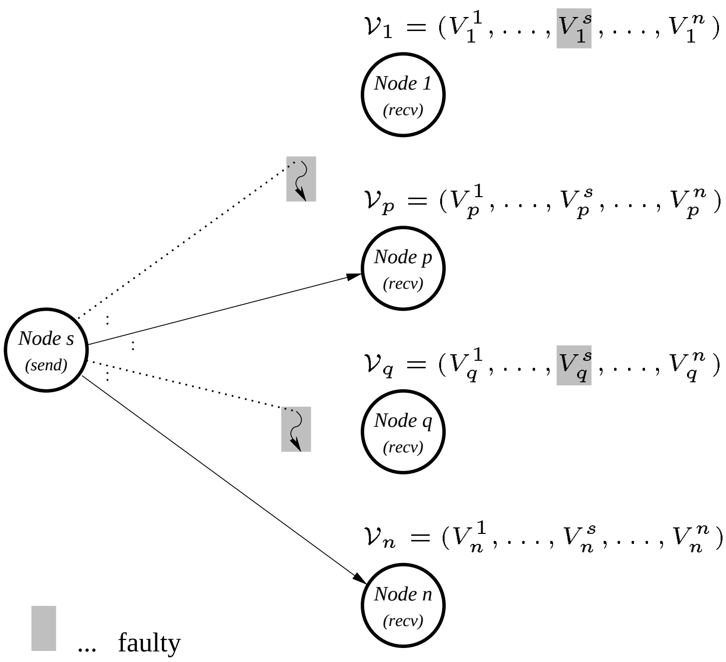

Figure 6. *Example of a broadcast fault that affects the messages of two recipients.*

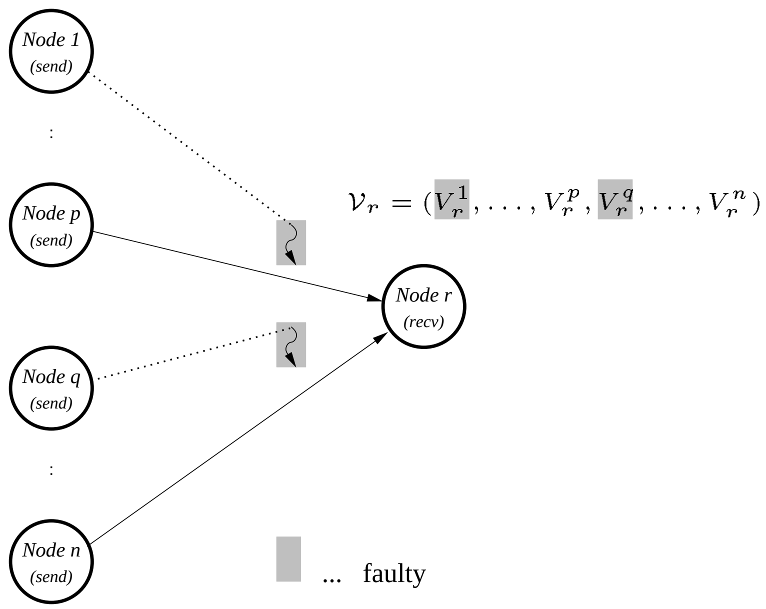

Figure 7. *Example of a receive fault that involves the messages from two senders.*

synchronization framework to general round-based synchronous algorithms.

In order to cleanly specify the semantics of such restricted faults, we will need the notion of *obedient* processors: An obedient processor is a live processor that faithfully executes the particular algorithm. It must hence get its inputs and perform its computations like a non-faulty processor, but it may fail in specific ways to communicate its value to the outside world. We will subsequently use this term instead of non-faulty whenever a processor acts as a receiver.

Definition 1 (System Model) *We consider a distributed system of n processors interconnected by a fully connected point-to-point network, which has the following properties:*

(P1) In any execution, there may be at most f_a, f_s, f_o, and f_c arbitrary, symmetric, omission, and manifest faulty processors. Those fault modes are defined via the set rvals(V_p, p) of admissible values de-

livered by obedient receivers when processor p attempts to send some value V_p in a single broadcast:

- *A* manifest *faulty processor p fails to send a message to any receiver or sends a value that all receivers q can detect as obviously bad. They all deliver the value $V_q^p = \emptyset$ in this case, i.e., rvals$(V_p, p) = \{\emptyset\}$.*
- *An* omission *faulty processor p may fail to send the correct value V^p to some of its receivers q_i, which deliver $V_{q_i}^p = \emptyset$ instead of V^p in this case. Hence, rvals$(V_p, p) = \{V_p, \emptyset\}$.*
- *A* symmetric *faulty processor p sends the same wrong (but not usually detectably bad) value X^p to every receiver q. They all deliver $V_q^p = X^p$—the value* 'actually sent'*—in this case, such that rvals$(V_p, p) = \{X^p\}$.*
- *An* arbitrary (asymmetric) *faulty processor may inconsistently send any value to any receiver, so rvals(V_p, p) is the set of all possible values, including $\emptyset$.*

A processor that is either manifest and omission faulty is called benign *faulty and is assumed to be obedient (if not crashed).*

(A1^s) If a single [faulty or non-faulty] processor p broadcasts (= successively sends) a value V^p to some set of obedient receiver processors $\mathcal{R}$, at most f_ℓ^s of the delivered values $V_{q_i}^p$ may differ from the admissible receive values in rvals(V^p, p). Let $f_\ell^{s,a} \leq f_\ell^s$ be the maximum number of non-omissive, i.e., non-empty and hence value faulty, $V_{q_i}^p$ among those.

(A1^r) If all processors $p_i \in \mathcal{S}$ of a set of [faulty or non-faulty] processors send a message containing V^{p_i} to some obedient receiver processor q, at most f_ℓ^r of the delivered values $V_q^{p_i}$ may differ from the admissible receive values in rvals(V^{p_i}, p_i). Let $f_\ell^{r,a} \leq f_\ell^r$ be the maximum number of non-omissive, i.e., non-empty and hence value faulty, $V_q^{p_i}$ among those.

(A2) The receiver of a message knows who sent it.

(A3) The absence of a message from sender p can be detected at any receiver q, which leads to $V_q^p = \emptyset$ for some distinguished value $\emptyset$.

Several essential remarks can be made about a system conforming to this system model.

(SM1) Assumption (A3) ultimately implies a synchronous system, where all non-faulty processors operate in lockstep rounds. Any processor's round consists of some local computation based upon the messages received in the previous round, the broadcast of the resulting messages to all processors (including itself) in the system ($A1^s$) and the reception of those messages ($A1^r$).

(SM2) Our manifest faults differ from the systemwide detectably ones of [54] and the "benign faults" of [6, 71] by also including symmetric omissions (produced by clean crashes). Since a receiver does not know whether a message is missing due to a symmetric omission or a more severe type of fault, they are usually more difficult to handle than pure manifest faults. This is not true for the Byzantine agreement algorithms analyzed in this paper, however, so we can safely take over the extended definition from [45] here.

(SM3) Our fault model does not contain a direct equivalent for standard *crash faults*, where a processor can die once and forever even during its broadcast. Crash faults are in fact weaker than our omission faults (=*send/receive omissions* [51]), but more severe than our manifest faults, although link faults allow even some inconsistency to occur. Nevertheless, both manifest and omission faults are more severe than crashes in that faulty processors may resume correct operation in any later round. This behavior is not equivalent to the crash-recovery model of [2], however, since our processors may not lose state, but must continuously follow the algorithm, see Remark (SM4) below.

(SM4) Benign faulty processors are allowed to deliver either the correct value or else $\emptyset$ to their receivers, which implies that they must know the correct value at least internally. Although we need not care how this is actually accomplished, it is nevertheless true that the only way to ensure this in practice is to assume that benign faulty processors are obedient (unless they have crashed).

(SM5) Faulty processors must not change their fault mode, i.e., must be counted in f_a, f_s, f_o or f_c according to their most severe behaviour. A processor that behaves symmetric faulty in one round and omission or manifest faulty in another should be considered arbitrary faulty.

(SM6) A sender processor that suffers from link faults according to ($A1^s$) is said to commit a *broadcast fault*, recall Figure 6, whereas a receiver processor that experiences link faults according to ($A1^r$) is said to commit a *receive fault*, recall Figure 7. Each processor's receive resp. broadcast fault has its own "budget" f_ℓ^r resp. f_ℓ^s of individual link faults, which are independent of processor faults, and the particular links actually hit are usually different for any two message broadcasts resp. receptions.

(SM7) We assume that links consist of a pair of unidirectional channels that can be hit by faults independently. This is also true for the hypothesized link from a processor to itself, although this one will not usually suffer from a link fault if the processor is non-faulty.

(SM8) The model parameters f_ℓ^r and f_ℓ^s are not independent of each other: If a message from processor p to q is hit by a fault in p's message broadcast, it contributes a fault in processor q's message reception as well. In fact, ($A1^s$) and ($A1^r$) can only be guaranteed unconditionally if

$$f_\ell^s \le f_\ell^r \quad \text{and} \quad f_\ell^{s,a} \le f_\ell^{r,a}, \tag{1}$$

see [61, 62] for details. Note that this implies $f_\ell^r = 0 \Rightarrow f_\ell^s = 0$.

(SM9) The system model of Definition 1 considers processor and link faults independently. Therefore, even a manifest or omission faulty processor's broadcast could generate erroneous values at $f_\ell^{s,a}$ receivers, for example. By contrast, the original model in [61, Def. 1] assumed that link faults hit only messages from non-faulty senders to non-faulty receivers. The new model is more natural and has a better coverage in real systems, but requires a slightly more complicated analysis.

(SM10) Since the consequences of an incomplete communication graph can be viewed as link omission faults, any analysis under our fault model also provides results that are valid for partially connected networks.

The following lower bound for the number of processors needed to solve the Byzantine agreement problem was developed in [62].

Theorem 3 (Lower Bound [62]) *Any deterministic algorithm that solves consensus under the system model of Definition 1 with* $f_\ell^r \geq f_\ell^s$ *and* $f_\ell^{r,a} \geq f_\ell^{s,a}$ *needs* $n > f_\ell^r + f_\ell^{r,a} + f_\ell^s + f_\ell^{s,a}$.

3.4 Related Work on Link Faults

The perception-based fault model is not the only way to circumvent the impossibility result of Gray [38]. A number of approaches to account for link faults or faults of the receiver's network interface in processor-centric models have been proposed. These are, however, insufficient as the following discussion will show.

If, for instance, link faults are simply mapped to sender process faults, as in [37], we can find many fault patterns for which all $f = n$ processors must be considered faulty. Figure 8 shows an example for $n = 4$ and $f_\ell = 1$, with f_ℓ denoting the number of incoming links that may be faulty per round for each processor.

Similar arguments can be found for the more detailed send/receive-omission fault model of [51], where receive omissions are mapped to a fault of the receiving processor. Although it has been observed that only the number of processors that commit a send omission (but not the number of processors committing a receive omission) needs to be counted in f, agreement is only shown to hold for a processor that did

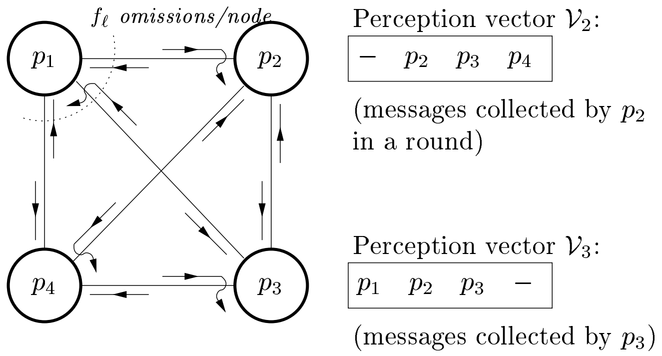

Figure 8. *Example of a 4-processor system with $f_\ell = 1$ receive link faults per processor in each round, where all processors must be considered faulty in existing fault models.*

not commit either type of fault, which implies that in the example of Figure 8, no processor would remain that could reach agreement.

In both models, the situation gets worse due to the fact that, over time, any receiver's f_ℓ receive omissions could hit different links. Since processor faults are usually considered persistent, this will result in the "exhaustion" of the number of faults f that are tolerated by the system.

Siu, Chin and Yang [65] have used a hybrid fault model, which accounts for dormant[4] and arbitrary faults of both links and processors. They presented a protocol for Byzantine agreement that works for $n > 3P_a + P_d$ and $c > 2P_a + P_d + 2(L_a + L_d)$, where n is the number of processors as usual, c denotes the connectivity of the underlying network graph and P_a, P_d, resp. L_a, L_d denote the upper bounds on the number of arbitrary and dormant processor resp. link faults. What seems favourable about this model is that the number of processors is independent of the number of faulty links. However, this is not entirely true, as the following discussion shows:

Considering the underlying graph of a network: Let κ_n denote the node connectivity and κ_e the edge-connectivity, which are defined as the number of nodes or edges that may be removed without making

[4]This class corresponds to what we call omission faults.

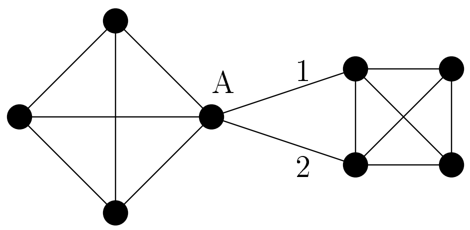

Figure 9. *A Graph with node-connectivity $\kappa_n = 1$, egde-connectivity $\kappa_e = 2$, and a minimal degree of $\delta = 3$.*

the graph unconnected,[5] and let δ denote the minimal degree[6] in the graph. The graph in Figure 9 has $\kappa_n = 1$, because removing the node labelled A will split the graph into two; removing the edges labelled 1 and 2, will also produce two sub-graphs, hence $\kappa_e = 2$.

Due to the well-known relation $\kappa_n \leq \kappa_e \leq \delta$, a network graph with connectivity c implies that $\delta \geq c$, which in turn implies that there must be $n > c + 1$ processors in the system, since a degree δ is only possible if there are δ other nodes in the graph.[7]

Thus the algorithm of [65] really needs $n > \max(3P_a + P_d, 2P_a + P_d + 2(L_a + L_d))$, which is a lot more than necessary in our systems, since Siu, Ching and Yang only consider static link faults, i.e., the total number of links that may be hit in an execution of the algorithm must be known in advance. They also claim that their algorithm can reach agreement in $t = \lfloor (n-1)/3 \rfloor$ rounds, which must be wrong, since it violates the lower bound for the required number of rounds (correctly $P_a + P_d$): if $n = 3P_a + P_d + 1$, they claim that $t = P_a + \lfloor (P_d)/3 \rfloor$ rounds suffice.

Sayeed, M. Abu-Amara and H. Abu-Amara considered an asynchronous system with arbitrary link faults only [57]. This work is not comparable to ours, since it deals with link faults only and cannot tol-

[5] or reducing it to a simple single node graph.
[6] The number of links a node, i.e. the corresponding processor, has.
[7] Assuming at most one edge between any two nodes.

erate a single processor fault.[8] Their algorithm can, however, tolerate up to $t = \frac{n-2}{2}$ Byzantine faulty channels, where t is the total number of link faults throughout an execution. Considering only Byzantine link faults in the perception-based fault model,[9] it is possible to tolerate nf_ℓ such faults in each round, which is equal to $\mathcal{O}(n^2)$.

The only other work that considers link faults in a dynamic way is [52]. Pinter and Shinahr assume that omissions may occur on a maximum of l different links system-wide per round. Their algorithm is identical to the "* algorithm", which was developed by Dolev, Fischer, Fowler, Lynch and Strong [25], except that it needs $a+l+2$ additional rounds, where a is the number of arbitrary processor faults the algorithm can tolerate. To achieve this the algorithm requires $n > 3(a+l)$ processors.

The main difference to our perception-based fault model (Section 3.3) is that Pinter and Shinahr bounded the total number of faulty links per round with $l \leq \frac{n}{3}$, whereas the total number of omission faulty links tolerated by algorithms that work under the perception-based fault model is $nf_\ell \gg n$. Note that the Phase King Protocol[10] (Section 4.2) allows for f_ℓ link faults (per round per processor) and f_a arbitrary faults, if $n > 3f_a + 3f_\ell$. Since the number of total link faults in a round is nf_ℓ in the perception-based fault model, but only l in the model of Pinter and Shinahr, it is obvious that this resilience against link faults is much better, with approximately the same message complexity of roughly $3(a+l)$ and $3(f_a+f_\ell)$ polynomial sized messages for the modified "* algorithm" and the Phase King protocol respectively.

As already mentioned, a perfect link can be simulated with a suitable retransmission scheme. As an example I will summarize the achievements of Guerraoui, Oliveira and Schiper [39], who provide a method for solving consensus in an asynchronous system with *basic unreliable*

[8] Which is, as Fischer, Lynch and Peterson [33] showed, not possible in asynchronous systems with a deterministic algorithm and without failure detectors.

[9] I.e., $f_a = f_o = f_s = f_c = 0$, and $f_\ell^r = f_\ell^{r,a} = f_\ell^s = f_\ell^{s,a}$

[10] This discussion only considers arbitrary faults. So consider $f_s = f_o = f_c = 0$ and only omission faulty links, so $f_\ell^{r,a} = f_\ell^{s,a} = 0$. Also assume $f_\ell^r = f_\ell^s$ for now (cf. Remark (SM8) above).

channels. Basic unreliable channels have the following three properties:

Fair Loss If a processor p sends an infinite number of messages to q, and q does not crash, then q receives an infinite subset of these messages.

No Creation If a processor q receives message m, then some processor p has sent m to q.

No Duplication Every message m that is sent by any process p is received at most once.

Based on these properties Guerraoui et al. implement what they call *stubborn channels*, i.e, channels that stubbornly retransmit their messages, which allow using the Chandra and Toueg algorithm [19] using *eventually strong* failure detectors. A direct comparison with our results is not feasible, since this work is based upon an asynchronous system model.

Dolev et al. [26] have presented a clock synchronization algorithm that can handle any number of process and link faults as long as the correct processors remain connected, using authentication. Schmid [58] investigated clock synchronization under the perception-based fault model.

Leader Election in asynchronous systems has been investigated for different kinds of link faults: Sayeed, Abu-Amara and Abu-Amara allowed Byzantine faulty links [57], while Abu-Amara and Lokre [1], and also Singh [64], only considered transient omission faults.

Initial faults, which denote links that are initially dead and remain dead throughout the execution[11] were covered by Bar-Yehuda et al. in [11].

Goldreich and Sneh [36] deal with the general case of a distributed global computation in an asynchronous system with unidirectional links,

[11] This kind of faults is equal to a non-existent link, thus this path of research is really about not fully connected networks.

which may suffer undetectable faults. They prove that the lower bound for the message complexity in such a system is $\frac{nm}{\mathcal{O}(\log^k(n)).}$, where n is the number of processors as usual and m the number of unidirectional links.[12]

In Wide Area Networks or even LANs routing is a fundamental service, whose reliability may not depend on the reliability of the components involved. Thus fault tolerance is a basic precondition for any reasonable routing scheme. Faulty or congested links are more frequent than failures in one of the routing components. Routing is a field of its own and can therefore not be covered here. For papers on routing theory and link faults see, e.g. [7, 48]. An introduction from the practical point of view can be found in [68], current solutions for the Internet world are found in the appropriate RFCs [55]. Most interesting might be OSPF (Open Shortest Path First), which is based on Dijkstra's shortest path algorithm.

A considerable amount of research has also been conducted on the problems link faults can cause in hypercube architectures, which is a massively parallel computer architecture.

[12] Bidirectional links can be modelled by two unidirectional links in this system.

4 Polynomial Byzantine Agreement

In Section 2.5 I presented a solution for Byzantine agreement (cf. Section 2.4 and [43, 44]). However, that protocol needs exponentially sized messages and thus exponential time to execute.[13] In this section I will present protocols that also solve the BA-problem while requiring only polynomial time (and space). The price one has to pay for increasing the message efficiency is that resilience is decreased.

Before presenting the protocols I will show two properties that will turn out to be useful in proving the correctness of the protocols. The first lemma points out that the difference in the number of messages with a particular value received at two different processors during a universal exchange can be bounded. A universal exchange is a message pattern where all (obedient) processors send a message to all other processors.

Lemma 7 (Difference in Perceptions) *Let $C_r[v]$ and $C_q[v]$ be the number of messages containing v received at two obedient processors r and q in a full message exchange of a system complying to Definition 1. Then,*

$$|C_q[v] - C_r[v]| \leq f_a + f_o + f_\ell^r + f_\ell^{r,a}.$$

Proof: Assuming w.l.o.g. $C_q[v] > C_r[v]$, at most f_a arbitrary faulty senders could have sent v to processor q and $x \neq v$ to processor r, and at most f_o processors could have sent v to q, but no message to r. Processor faults hence contribute at most $f_a + f_o$ to $|C_q[v] - C_r[v]|$. The remaining terms originate from link faults: Processor q could have received at most $f_\ell^{r,a}$ messages containing v from processors that actually sent $x \neq v$, and at most f_ℓ^r messages containing v could have been lost in transit to processor r. □

When using a majority test after a full message exchange as in Line 8 of Figure 10, inconsistent reception could produce a majority for the

[13] Exponential time always follows from exponential data, since it takes exponential time to generate or as in this case to send that data.

value v at some processor q, but a majority for $x \neq v$ at some processor $r \neq q$. The question arises how big the lead of v must be at processor q to be able to guarantee that r will compute the same majority. This is answered by the following Lemma 8.

Lemma 8 (Deviation of Differences) *Let $C_r[v]$ and $C_q[v]$ denote the number of messages containing $v \in \{0,1\}$ received at two obedient processors r and q in a binary full message exchange of a system complying to Definition 1. For $\Delta_q[v] = C_q[v] - C_q[1-v]$ and $\Delta_r[v] = C_r[v] - C_r[1-v]$, we obtain*

$$|\Delta_q[v] - \Delta_r[v]| \leq 2f_a + f_o + f_\ell^r + f_\ell^{r,a}.$$

Proof: Abbreviating $\Delta = |\Delta_q[v] - \Delta_r[v]|$, each asymmetric faulty processor could change Δ by at most 2, by sending v to q but $1-v$ to r, while each omission faulty processor could change Δ by at most 1, by failing to send a message to either q or r; the remaining symmetric and manifest faults cannot change Δ. Processor faults could hence grow/shrink Δ by at most $2f_a + f_o$. The worst case deviation due to link faults occurs when both q and r experience $f_\ell^{r,a}$ arbitrary ones (which flip the value sent) and $f_\ell^r - f_\ell^{r,a}$ omissive ones from different sender processors; the appropriate contributions sum up to $2f_\ell^{r,a} + (f_\ell^r - f_\ell^{r,a}) = f_\ell^r + f_\ell^{r,a}$. □

The Phase Queen and King Protocols, which are presented in the next two subsections (4.1 and 4.2) were developed by Berman, Garay and Perry [17].

4.1 Phase Queen

The *Phase Queen Protocol* introduced in [17] is a simple algorithm that solves consensus for binary values in systems with $n > 4f$ processors, at most f of which may be arbitrary faulty, and no link faults. It assumes unique processor identifiers $\in \{1, \ldots, n\}$, uses constant size (1-bit) messages and takes $f+1$ rounds with 2 phases each. Our contribution is

```
for k:=1 to f_a + f_s + f_o + f_c + 2 do begin

        /* Phase 1: full message exchange */
        send(v) to all processors
        receive(v_q) from all processors
        C[0] := |{v_q : v_q = 0}|
        C[1] := |{v_q : v_q = 1}|
        if C[1] > C[0] then v = 1 else v = 0 fi

        /* Phase 2: queen's broadcast */
        if k = p then
                send(v) to all processors
        fi
        receive(v_queen)
        if v_queen = ∅ then v_queen := 0 fi
        if C[v] ≤ C[1 − v] + 2f_a + f_o + f_ℓ^r + f_ℓ^{r,a} then
                v := v_queen
        fi
done
```

Figure 10. *Hybrid Phase Queen algorithm, code for processor p*

a modified version of the original algorithm that can cope with hybrid processor and link faults according to the system model of Definition 1.

Figure 10 shows the pseudo code of our hybrid algorithm, which works as follows: In the first phase of each round, every processor p broadcasts its current preference value v_p to all processors in the system (including itself) and collects the appropriate messages from its peers; we will call this a *full message exchange*. Processor p then counts how many processors have sent preference for 0 and 1 respectively and updates its v_p accordingly. In the second phase of each round, only one specific processor (the *Phase Queen*, whose identifier is equal to the current round number) broadcasts its new preference. This value is used by all processors in the system to break ties.

Note that only two non-trivial changes were made to the original algorithm: First, the bound in the decision when to use the queen's broadcast (line 16 in Figure 10) had to be adapted to our hybrid fault model. Second, in order to improve the resilience with respect to non-arbitrary faults, the original condition $C[1] > n/2$ in the update of the preference value (line 8) had to be replaced by the simple majority test $C[1] > C[0]$.

In order to show that the above algorithm satisfies the agreement (C1) and validity property (C2) of consensus, the analysis of the Phase Queen algorithm from [4] will be adopted. The first major lemma states that once sufficiently many processors have the same preference value, it will not change anymore. Lemma 9 will implies the validity property (C2).

Let d^k denote the number of actually dead processors among the manifest or omission faulty ones by the beginning of round k; obviously $f_c + f_o \geq d^i \geq d^j$ must hold[14] for all $i \geq j$. We will need this value to discriminate between obedient processors and benign faulty processors that are dead.

Lemma 9 (Persistence of Agreement) *If at least $n - f_a - f_s - d^k - f_\ell^s$ obedient processors prefer $v \in \{0, 1\}$ at the beginning of round $k \in \{1, \ldots, f_a + f_s + f_c + 2\}$ in a system with $n > 4f_a + 2f_s + 2f_o + f_c + 2f_\ell^s + 2f_\ell^r + 2f_\ell^{r,a}$, then all obedient processors prefer v at the end of round k.*

Proof: Since $n - f_a - f_s - d^k - f_\ell^s$ obedient processors prefer v at the beginning of phase k, every obedient processor q receives $C_q[v] \geq n - f_a - f_s - f_o - f_c - f_\ell^s - f_\ell^r$ preferences for v (including its own) in the first phase of round k. Moreover, since the preference of f_ℓ^s obedient processors was left unspecified by Lemma 9 and at most $f_a + f_s$ processors and $f_\ell^{r,a}$ links may cause erroneous values, processor q could also get at most $C_q[1 - v] \leq f_a + f_s + f_\ell^s + f_\ell^{r,a}$ preferences for $1 - v$. This implies

$$C_q[v] \quad \geq \quad n - f_a - f_s - f_o - f_c - f_\ell^s - f_\ell^r$$

[14] This allows clean as well as unclean crash faults. Whereas a clean crash can be accounted for in f_c, an unclean one is an asymmetric omission in one round followed by an symmetric omission in every later round, and must thus be accounted for in f_o.

$$\begin{aligned} &> 3f_a + f_s + f_o + f_\ell^s + f_\ell^r + 2f_\ell^{r,a} \\ &\geq C_q[1-v] + 2f_a + f_o + f_\ell^r + f_\ell^{r,a}. \end{aligned}$$

So every obedient processor q updates its preference v_q to v in line 8 in Figure 10 and ignores the queen's broadcast in line 16. □

The second major lemma is a prerequisite for showing that agreement can be reached by means of a non-faulty queen breaking ties.

Lemma 10 (Agreement) *Let g be a round whose queen is non-faulty. Then at least $n - f_a - f_s - d^{g+1} - f_\ell^s$ obedient processors finish round g with the same preference.*

Proof: Since the queen g is non-faulty, it sends the same v_g to all receivers. Assume first that all obedient processors use the value v_{queen} received from g as their new preference in line 17 of Figure 10. According to (A1^s), at most f_ℓ^s of those could have received a queen's preference $v_{queen} \neq v_g$,[15] such that at least $n - f_a - f_s - d^{g+1} - f_\ell^s$ obedient processors have set their preference to v_g by the end of round g.

If, on the other hand, some obedient processor p ignores the queen's broadcast and uses its own majority value v as its new preference, $C_p[v] - C_p[1-v] > 2f_a + f_o + f_\ell^r + f_\ell^{r,a}$ according to line 16 in Figure 10. Hence, for every other obedient processor q (including the non-faulty queen g), $C_q[v] - C_q[1-v] \geq C_p[v] - C_p[1-v] - 2f_a - f_o - f_\ell^r - f_\ell^{r,a} > 0$ by Lemma 8, so g must have set its preference to v in line 8 as well. Processor p can therefore safely use its majority value instead of the queen's in this case, since they are the same. □

By means of Lemma 9 and 10, it is not difficult to prove the following major theorem.

Theorem 4 (Phase Queen) *Under the system model of Definition 1 with $f_a, f_s, f_o, f_c, f_\ell^r, f_\ell^{r,a}, f_\ell^s \geq 0$ and $n > 4f_a + 2f_s + 2f_o + f_c + 2f_\ell^s + 2f_\ell^r +$*

[15] At most $f_\ell^{s,a}$ processors could receive the opposite value $1 - v_g$, and the remaining $f_\ell^s - f_\ell^{s,a}$ ones could suffer from omissions, that is, deliver $\emptyset$.

$2f_\ell^{r,a}$, *the Phase Queen algorithm of Figure 10 solves binary consensus in* $2(f_a+f_s+f_o+f_c+2)$ *phases with a total of* $(f_a+f_s+f_o+f_c+2)(n+1)$ *1-bit message broadcasts from obedient processors.*

Proof: Lemma 9 already implies the validity property (C2): If all obedient processors start out with the same value v, they will continue to prefer v until the algorithm terminates after $f_a + f_s + f_o + f_c + 2$ rounds, since there are $n - f_a - f_s - d^k \geq n - f_a - f_s - f_o - f_c - f_\ell^s$ non-faulty processors in any round k.

To show agreement (C1), we note that all – but at most f_ℓ^s – obedient processors will have the same preference at the end of a non-faulty queen's round g by Lemma 10. Definition 1 ensures that at least one round $g \in \{1, \dots, f_a+f_s+f_o+f_c+1\}$ has a non-faulty queen. Since the algorithm takes $f_a+f_s+f_o+f_c+2$ rounds, Lemma 9 eventually assures that all obedient processors will have the same (persistent) preference by the end of round $g+1$,[16] no matter how many rounds with faulty queens follow.

To justify the claimed time and communication complexity of our algorithm, we note that there is one full message exchange consisting of n broadcasts in phase 1 and one additional queen's broadcast in phase 2 of every round. Hence, at most $(f_a + f_s + f_o + f_c + 2)(n + 1)$ 1-bit message broadcasts are performed during the whole execution by processors that faithfully[17] follow the algorithm in Figure 10. □

As a final remark, we note that the number of message broadcasts of the Phase Queen algorithm could be reduced by one by omitting the queen's broadcast in the last round: According to the proof of Theorem 4, no non-faulty processor uses v_{queen} in the last round at all.

4.2 Phase King

The *Phase King Protocol* of [17] improves the Phase Queen algorithm by adding one phase in every round, in which the processors' prefer-

[16]Remember that $d^{k+1} \leq d^k$.

[17]Clearly, there is no way to restrict the number of message broadcasts initiated by an arbitrary faulty processor.

ences from the first phase are exchanged system-wide. This eventually reduces the sub-optimal fault-tolerance degree $n > 4f$ of the Phase Queen algorithm to $n > 3f$. We provide a hybrid variant of this algorithm in Figure 11, which can cope with processor and link faults according to Definition 1.

The proof for the Phase King Protocol is similar to the one for the Phase Queen Protocol: The first Lemma 11 will show that once sufficiently many processors have the same preference value, it will not change anymore.

Lemma 11 (Persistence of Agreement) *If at least $n-f_a-f_s-d^k-f_\ell^s$ obedient processors prefer v at the beginning of round $k \in \{1,\dots,f_a+f_s+f_o+f_c+2\}$ in a system with $n > 3f_a+2f_s+2f_o+f_c+2f_\ell^s+2f_\ell^r+2f_\ell^{r,a}$ processors, then all obedient processors prefer v at the end of round k.*

Proof: Since at least $n - f_a - f_s - f_o - f_c - f_\ell^s$ non-faulty processors broadcast v in the first full message exchange, every obedient processor q obtains $C_q[v] \geq n - f_a - f_s - f_o - f_c - f_\ell^s - f_\ell^r$ and $C_q[1-v] \leq f_a+f_s+f_\ell^s+f_\ell^{r,a}$ according to our fault model in Definition 1. Similar to the proof of Lemma 9, this leads to

$$C_q[v] > C_q[1-v] + f_a + f_o + f_\ell^r + f_\ell^{r,a},$$

which implies that every non-faulty processor among the obedient ones will send the same $M[j]$ in the second phase. Therefore, every obedient processor r obtains $D_r[v] \geq n - f_a - f_s - f_o - f_c - f_\ell^r$ and $D_r[1-v] \leq f_a + f_s + f_\ell^{r,a}$ and thus sets its local preference to the same value v in line 18 in Figure 11; the king's value v_{king} is ignored in line 26 since $D_r[v] > 2f_a + f_s + f_o + f_\ell^r + 2f_\ell^{r,a}$. □

The second major lemma is again a prerequisite for showing that agreement can be reached by means of a non-faulty king breaking ties.

Lemma 12 (Agreement) *Let g be a round whose king g is non-faulty. If $n > 3f_a+2f_s+2f_o+f_c+2f_\ell^s+2f_\ell^r+2f_\ell^{r,a}$, then at least $n-f_a-f_s-d^{g+1}-f_\ell^s$ obedient processors start round $g+1$ with the same preference.*

```
for k:=1 to f_a + f_s + f_o + f_c + 2 do begin

        /* Phase 1: initial full message exchange */
        send(v) to all processors
        receive(v_q) from all processors
        C[0] := |{v_q : v_q = 0}|
        C[1] := |{v_q : v_q = 1}|

        /* Phase 2: C[j] full message exchange */
        for j:=0 to 1 do begin
                if C[j] > C[1 - j] + f_a + f_o + f_ℓ^r + f_ℓ^{r,a} then
                        M[j]:=1 else M[j]:=0
                fi
                send(M[j]) to all processors
                receive(M_q[j]) from all processors
                D[j] := |{M_q[j] : M_q[j] = 1}|
        done
        if D[1] > f_a + f_s + f_ℓ^{r,a} then v:=1 else v:=0 fi

        /* Phase 3: king's broadcast */
        if k = p then
                send(v) to all processors
        fi
        receive(v_king)
        if v_king = ∅ then v_king := v fi
        if D[v] ≤ 2f_a + f_s + f_o + f_ℓ^r + 2f_ℓ^{r,a} then
                v := v_king
        fi
done
```

Figure 11. *Hybrid Phase King algorithm, code for processor p*

Proof: At the end of round g one of the following cases applies in line 26 of Figure 11:

(1) $D_p[v_p] \le 2f_a + f_s + f_o + f_\ell^r + 2f_\ell^{r,a}$ for every obedient processor p, which thus assigns the value v_{king} to v_p in line 27. Since at most f_ℓ^s processors could have received a value v_{king} different from the value v_g actually sent, at least $n - f_a - f_s - d^{g+1} - f_\ell^s$ obedient processors end up with the same preference v_g as asserted.

(2) $D_p[v_p] > 2f_a + f_s + f_o + f_\ell^r + 2f_\ell^{r,a}$ for some obedient processor p. We have to show that p's preference at the end of the round is equal to the king's value v_g in this case, which will again imply that all obedient processors except at most f_ℓ^s will end up with a preference for the king's value v_g — either by receiving it, or by being convinced of it in the first place. We distinguish the following two possible cases here:

If $v_p = 1$, it must be that $D_g[1] > f_a + f_s + f_\ell^{r,a}$ and hence $v_g = 1$, since Lemma 7 implies $D_g[1] \ge D_p[1] - f_a - f_o - f_\ell^r - f_\ell^{r,a} > f_a + f_s + f_\ell^{r,a}$. If, on the other hand, $v_p = 0$, we can use the following argument to show that v_g cannot be 1:

We first prove that if an obedient processor q sends $M_q[b] = 1$ for some $b \in \{0,1\}$ in the second phase, then no obedient processor r can send $M_r[1-b] = 1$. Assuming the contrary, line 11 in Figure 11 would require both

$$\begin{aligned} \Delta_q &= C_q[b] - C_q[1-b] > f_a + f_o + f_\ell^r + f_\ell^{r,a} \\ -\Delta_r &= C_r[1-b] - C_r[b] > f_a + f_o + f_\ell^r + f_\ell^{r,a} \end{aligned}$$

and hence $\Delta_q - \Delta_r > 2f_a + 2f_o + 2f_\ell^r + 2f_\ell^{r,a}$, which would contradict Lemma 8.

Now $D_p[0] > 2f_a + f_s + f_o + f_\ell^r + 2f_\ell^{r,a}$ implies that at least one non-faulty processor sent $M[0] = 1$ in the second phase. Consequently, as we have just shown, no obedient processor can have sent $M[1] = 1$. Since this implies $D_q[1] \le f_a + f_s + f_\ell^{r,a}$ for any obedient processor q (including the king g), v_g is set to 0 in line 18, just before the king's broadcast. This eventually completes the proof of Lemma 12. □

Based on the two Lemmas 11 and 12, it is not hard to show the following theorem, which summarizes the properties of the *Phase King* protocol.

Theorem 5 (Phase King) *Under the system model of Definition 1 with* $f_a, f_s, f_o, f_c, f_\ell^r, f_\ell^{r,a}, f_\ell^s, f_\ell^{s,a} \geq 0$ *and* $n > 3f_a + 2f_s + 2f_o + f_c + 2f_\ell^s + 2f_\ell^r + 2f_\ell^{r,a}$*, the Phase King algorithm of Figure 11 solves binary consensus in* $3(f_a + f_s + f_o + f_c + 2)$ *phases with a total of* $(f_a + f_s + f_o + f_c + 2)(3n + 1)$ *1-bit message broadcasts from obedient processors.*

Proof: Since Lemma 11 and 12 are the same as Lemma 9 and 10, respectively, the proof of the agreement (C1) and validity property (C2) is literally the same as in Theorem 4. To justify the claimed time and message complexity, Figure 11 reveals that every processor broadcasts three 1-bit messages per round in Phases 1 and 2. Note that the broadcasts of $M[0]$ and $M[1]$ occur simultaneously in the same phase. Adding the single message broadcast by every round's king, $(f_a + f_s + f_o + f_c + 2)(3n + 1)$ messages are broadcast during the whole execution by not arbitrary faulty processors. □

As in the case of the Phase Queen algorithm, it would again be possible to omit the king's broadcast in the last round in this algorithm as well. Moreover, 2-bit messages could be used in the second phase, which would reduce the number of message broadcasts to $(2n+1)(f_a + f_s + f_o + f_c + 2) - 1$.

4.3 Simulating Authentication

This section features the hybrid version of the simple binary *Byzantine agreement* algorithm of [67] under the system model of Definition 1. The Byzantine agreement problem was already introduced and defined formally in Section 2.4.

The algorithm of [67] is built upon two communication primitives: **broadcast**(p, m, k) is used to broadcast messages and **accept**(p, m, k) is called whenever a message should be received. These primitives are used to exchange messages (p, m, k) consisting of the identifier of the sending processor p, some value $m \in \{0, 1\}$ and the round number k. The semantics of the broadcast primitive are fully captured by the following three properties:

(C) *Correctness*:
If a non-faulty processor p executes **broadcast**(p, m, k) in round k, then every obedient processor **accepts**(p, m, k) in the same round.

(U) *Uniform Unforgeability*:
If obedient processor p never executes **broadcast**(p, m, k), then no obedient processor ever **accepts**(p, m, k).

(R) *Uniform Relay*:
If an obedient processor **accepts**(p, m, k) in round $r \geq k$, then every obedient processor also **accepts**(p, m, k) in round $r + 1$ or earlier.

The usual way to implement these primitives is by means of signed messages in conjunction with relaying, i.e., forwarding accepted messages to all other processors in the system. An alternative solution that circumvents cryptography will be described later in Subsection 4.3.2. By plugging this hybrid simulated broadcast primitive into the "generic" hybrid algorithm introduced in Subsection 4.3.1, a non-authenticated algorithm for Byzantine agreement will be obtained that tolerates hybrid processor and link faults. Note that cryptographic implementations of the broadcast primitive cannot usually tolerate link faults, since f_ℓ^s messages could get lost in every broadcast, thereby violating correctness (C).

4.3.1 Srikanth & Toueg Authenticated Byzantine Agreement

Figure 12 shows our hybrid version of the algorithm for binary Byzantine agreement of [67]. It differs from the original algorithm only in that

```
/* Transmitter t */
if p = t then v := m else v := 0 fi
/* Receiver p */
for r:=1 to f_a + f_s + f_o + f_c + 1 do
    /* Round r */
    if v = 1 and not yet broadcast(p,_,_) then
        broadcast(p, 1, r)
    fi
    if accept(p_k, 1, r_k) from r-th distinct p_k, including t, then
        v := 1
    fi
done
```

Figure 12. *The hybrid binary Byzantine agreement algorithm of Srikanth & Toueg, code for processor p*

the number of faulty processors t has been replaced by $f_a + f_s + f_o + f_c$ and in that relaying, which was done explicitly in [67, Fig. 1], has been hidden within **broadcast**( ) for simplicity.

The algorithm proceeds in $f_a + f_s + f_o + f_c + 1$ rounds as defined by the broadcast primitive, where the only value broadcast by obedient processors is 1; the value 0 is decided upon by default. An obedient transmitter broadcasts 1 in round 1, if agreement is to be reached on the value $m = 1$. Any obedient receiver p sets $v := 1$ at the end of round r and hence delivers 1 if it has accepted r messages broadcast by different processors $P_r = \{p_1, \ldots, p_r\}$, one of which is the transmitter. Note that $p \notin P_r$ since p still has $v = 0$ during round r. The correctness and relay properties of the broadcast primitive ensure that all other correct processors accept all messages $\in P_r$ plus the one broadcast by p by round $r + 1$, hence they will also deliver 1 by then.

If $m = 0$ is the value to be agreed upon, an obedient transmitter never broadcasts any message. By the unforgeability property no obedient receiver ever accepts any message originating from the transmitter. Consequently, no obedient processor sets $v := 1$ and all must hence deliver 0 by default. Note that a manifest faulty transmitter is indistinguishable from a correct transmitter that tries to communicate $m = 0$, which explains why we assumed $\emptyset := 0$ for the validity property (B2).

By formalizing the above line of reasoning the following Theorem 6 shows that our hybrid algorithm achieves agreement (B1) and validity (B2), provided that the broadcast primitive employed in the algorithm can cope with the processor and link faults of Definition 1.

Theorem 6 (Hybrid Authenticated Algorithm) *Given an implementation of the broadcast primitive that guarantees (C), (U) and (R) under the system model of Definition 1, the hybrid algorithm of Figure 12 achieves the Agreement (B1) and Validity (B2) properties of Byzantine agreement within $f_a + f_s + f_o + f_c + 1$ rounds, where every obedient processor calls* **broadcast**( ) *at most once during the whole execution. Either, at least one non-faulty processor calls* **accept**( ) *by round $f_a + f_s + f_o + f_c$ or no non-faulty processor ever calls* **accept**( ).

Proof: We first show that validity (B2) is achieved. Since (B2) is void in case of (a) an arbitrary and (b) an omission faulty transmitter with $m = 1$ (recall that $\emptyset = 0$ here), we only have to deal with the following cases:

(1) If the transmitter t is non-faulty and agreement is to be reached on 0, then t does not call **broadcast**( ). Hence, no obedient processor will ever **accept**$(t, 1, \_)$ due to (U) and all obedient receivers will decide upon the default value $v = 0$. If agreement is to be reached on 1, a non-faulty transmitter calls **broadcast**$(t, 1, 1)$. By (C) all obedient processors will accept this message in round 1 and set $v := 1$ as required.

(2) If the transmitter is manifest faulty or omission faulty with $m = 0$, it either does not call **broadcast**( ), or if it does, all resulting messages are discarded. Hence, those two cases are indistinguishable for the receivers and no non-faulty processor will ever **accept**$(t, 1, \_)$ due to (U). Consequently, all non-faulty receivers will decide upon the default value $v = 0 = \emptyset$ as required.

(3) If the transmitter is symmetric faulty, it can only[18] appear like a non-faulty transmitter that broadcasts the opposite value. Hence, case (1) above applies.

In order to show the agreement property (B1), it suffices to show that if one non-faulty processor sets $v := 1$, then all other obedient processors will do so as well; agreement on 0 occurs per default. Let q be the first non-faulty processor that executes $v := 1$, and let l be the round in which this happens. Then, q must have accepted at least l different messages containing a value of 1 in round l. We distinguish two cases: If $l < f_a + f_s + f_o + f_c + 1$, then any obedient processor p will accept these messages due to (R) at most one round later as well. Since p will also accept q's broadcast of $(q, 1, l + 1)$ in round $l + 1$ due to (C), it will eventually end up with $v := 1$ as required.

If, on the other hand, $l = f_a + f_s + f_o + f_c + 1$, then the first non-faulty processor q that executed $v := 1$ does so in the last round $f_a+f_s+f_o+f_c+1$, where it accepted at least $f_a+f_s+f_o+f_c+1$ messages with a value of 1. However, at least one of these messages originates from a non-faulty processor, which must have executed $v := 1$ in some round $l' < l$. This contradicts our assumption of q being the first one, so at least one non-faulty processor must call **accept**( ) strictly before the last round. □

We finally note that it would be sufficient to use only $f_a+f_s+f_o+1$ instead of $f_a+f_s+f_o+f_c+1$ rounds in the algorithm of Figure 12 when the broadcast primitive of Section 4.3.2 is used: The above proof of the agreement property (B1) considers only messages that a non-faulty processor q accepted. However, the algorithm of Figure 13 ensures that no obedient receiver ever accepts a message from a manifest faulty broadcaster.

[18] Recall from Definition 1 that a symmetric faulty processor consistently sends information that is not detectably bad. A wrong transmitter identifier p or round number r in a message (p, m, r) would be detected at any receiver.

4.3.2 Hybrid Simulated Broadcast Primitive

As already mentioned, algorithms based on authenticated messages, like the one of Figure 12, are usually simple, easy to understand and have superior fault-tolerance properties as well. However, cryptographic methods (see [63] for an overview) typically used for authentication are expensive operations, that cannot always be afforded in power/bandwidth-limited systems. Moreover, since there is always a non-zero probability of guessing or forging a processor's signature, no present cryptographic authentication technique is unconditionally secure. Our algorithm's fault-tolerance properties, however, depend critically upon an uncompromised cryptosystem. As mentioned in [62], processors with broken signatures can — in the worst case — act as maliciously as arbitrary faulty processors, which are very costly to tolerate.

Those deficiencies make the non-authenticated implementation of the broadcast primitive developed by Srikanth & Toueg in [67] attractive. Instead of using cryptography, their *simulated broadcast primitive* relies upon the idea of letting all processors witness a processor's message broadcast. Using this approach, the algorithm of [67, Fig. 2] unconditionally guarantees the correctness (C), unforgeability (U) and relay (R) properties, provided that $n > 3f$ and at most f processors are arbitrary faulty.

Figure 13 shows our hybrid variant of this simulated broadcast primitive, which works under the fault model of Definition 1. It proceeds in consecutive rounds consisting of two phases each, where two kinds of messages are exchanged: When calling **broadcast**(p, m, k) in round k, processor p enters the code in phase $2k$. It first sends a message $(init, p, m, k)$ to all processors in the system, which is answered by every witness with a message $(echo, p, m, k)$ in phase $2k + 1$. A processor calls **accept**(p, m, k) in some round $r \geq k$, when it has received $n - f_a - f_s - f_o - f_c - f_\ell^s - f_\ell^r$ messages containing $(echo, p, m, k)$ from different processors. Note that the rounds of simulated authenticated broadcasting run in lockstep with the rounds of the application, i.e., the Byzantine agreement algorithm of Figure 12.

Multiple rounds can occur if the initial broadcaster p is faulty. The additional rounds consist of two identical phases each, where a processor that has not yet send $(echo, p, m, k)$ could get sufficient evidence that it should do so, which could in turn convince some processors to accept in the following phase.

In the original algorithm [67, Fig. 2] any processor q can terminate its instance of **broadcast**(p, m, k) after having called **accept**(p, m, k), since any other still active processor in the system must have seen q's echo message by then. However, in order to deal with link faults, we had to change the original algorithm in a subtle, but important way: Link faults can produce $f_\ell^{r,a}$ erroneous $(echo, p, m, k)$ messages per round at any obedient processor, which would accumulate over multiple rounds. In Figure 13, we hence retransmit $(echo, p, m, k)$ in every additional phase, instead of sending it only once and remembering receptions from previous rounds. Since other processors may accept one round later than q according to the relay property (R), however, any processor must retransmit $(echo, p, m, k)$ up to and including the phase following acceptance.

By adopting the analysis of [67] with guidance from [59], where the closely related asynchronous consistent broadcasting primitive of [66] was analyzed, we can prove the following Theorem 7.

Theorem 7 (Simulated Broadcast) *Under the system model of Definition 1 with $f_a, f_s, f_o, f_c, f_\ell^r, f_\ell^{r,a}, f_\ell^s, f_\ell^{s,a} \geq 0$ and $n > 3f_a + 2f_s + 2f_o + f_c + f_\ell^s + f_\ell^{s,a} + 2f_\ell^r + 2f_\ell^{r,a}$, the simulated broadcast primitive of Figure 13 guarantees correctness (C), uniform unforgeability (U), and uniform relay (R). During $0 < r \leq R$ rounds, where R is an a priori upper bound upon the maximum number of rounds the algorithm is allowed to run, at most $(2r-1)n + 1$ broadcasts of $(\log_2 n + \log_2 R + 2)$-bit messages are performed by obedient processors.*

Proof: We show each of the three properties separately:

Correctness: Since p is non-faulty, $n - f_a - f_s - f_o - f_c - f_\ell^s$ non-faulty processors receive the message $(init, p, m, k)$ in phase $2k$ and

/ Round k */*
    / Phase $2k$: Processor p only (entered by* **broadcast**( )*) */*
    send$(init, p, m, k)$ **to all processors**;
    / Phase $2k + 1$: All processors */*
    if received$(init, p, m, k)$ from p in phase $2k$ **then**
        send$(echo, p, m, k)$ **to all processors**
    fi
    if received$(echo, p, m, k)$ in phase $2k + 1$
        $\geq n - f_a - f_s - f_o - f_c - f_\ell^s - f_\ell^r$ **then**
        accept(p, m, k)
    fi

/ Round $r \geq k + 1$ */*
    / Phase $2r$, $2r + 1$ (same code): All processors */*
    if received$(echo, p, m, k)$ in previous phase
        $\geq n - 2f_a - f_s - 2f_o - f_c - f_\ell^s - 2f_\ell^r - f_\ell^{r,a}$
    or sent$(echo, p, m, k)$ in previous phase **then**
        send$(echo, p, m, k)$ **to all processors**
    fi
    if accepted(p, m, k) in previous phase **then**
        terminate
    fi
    if received$(echo, p, m, k)$ in this phase
        $\geq n - f_a - f_s - f_o - f_c - f_\ell^r$ **then**
        accept(p, m, k)
    fi

Figure 13. *The hybrid version of the simulated broadcast primitive of Srikanth & Toueg*

send$(echo, p, m, k)$ to all processors in phase $2k + 1$. Hence, every obedient processor receives $(echo, p, m, k)$ from at least $n - f_a - f_s - f_o - f_c - f_\ell^s - f_\ell^r$ distinct processors in phase $2k+1$ and thus **accepts**(p, m, k) in phase $2k + 1$, i.e., in round k.

Uniform Unforgeability: The proof is by contradiction: Assume that some obedient processor q **accepts**(p, m, k) in some round $r \geq k$, although the obedient processor p did not execute **broadcast**(p, m, k) and hence did not send any $(init, p, m, k)$ message in phase $2k$. Nevertheless, $f_\ell^{s,a}$ obedient processors might have received $(init, p, m, k)$ as the result of a "foreign" broadcast link fault at p, such that at most $f_a + f_s + f_\ell^{r,a} + f_\ell^{s,a}$ $(echo, p, m, k)$ messages might arrive at any obedient processor. Since $f_a + f_s + f_\ell^{r,a} + f_\ell^{s,a} < n - 2f_a - f_s - 2f_o - f_c - f_\ell^s - 2f_\ell^r - f_\ell^{r,a}$, no obedient processor, except the at most $f_\ell^{s,a}$ ones that retransmit *echo* (cf. line 17), will ever execute **send**$(echo, p, m, k)$ in line 18 of Figure 13. Since q executed **accept**(p, m, k), however, it must have received $(echo, p, m, k)$ from at least $n - f_a - f_s - f_o - f_c - f_\ell^s - f_\ell^r > f_a + f_s + f_\ell^{r,a} + f_\ell^{s,a}$ processors, which provides the required contradiction.

Uniform Relay: Let l be the phase in which the first non-faulty processor q accepts (p, m, k) in round r. Processor q must have received at least $n - f_a - f_s - f_o - f_c - f_\ell^s - f_\ell^r$ *echo*-messages from different processors in phase l, which implies that every other non-faulty processor must have received at least $n - 2f_a - f_s - 2f_o - f_c - f_\ell^s - 2f_\ell^r - f_\ell^{r,a}$ *echo*-messages as well by Lemma 7. According to line 16 of Figure 13, every non-faulty processor thus emits an *echo*-message in phase $l + 1$. Therefore, by phase $l + 1$ in round $r' \leq r + 1$, every obedient processor will receive $(echo, p, m, k)$ at least $n - f_a - f_s - f_o - f_c - f_\ell^r$ times and will hence **accept**(p, m, k).

As far as the claimed message complexity of **broadcast**( ) is concerned, it is of course again impossible to bound the number of message broadcasts by (arbitrary) faulty processors. Every processor that faithfully executes the algorithm of Figure 13, however, executes one broadcast of $(echo, p, m, k)$ (Lines 6 and 18) in each phase following the initial one, where only processor p broadcasts $(init, p, m, k)$ (Line 3). Hence, during $0 < r \leq R$ rounds, at most $1 + n + 2(r-1)n = (2r-1)n + 1$ broadcasts of $(1 + \log_2 n + 1 + \log_2 R)$-bit messages are performed. This eventually completes the proof of Theorem 7. $\square$

Note that the bound R upon the number of rounds required by Theorem 7 must be enforced externally, as in the Byzantine agreement algorithm of Figure 12. After all, the collusion of a faulty broadcaster with faulty witnesses could let the number of required rounds grow without bounds, as already noted in [67]. Still, this could only happen if no non-faulty processor has accepted yet: By the relay property, all non-faulty processors must accept by the end of the phase following acceptance of the first non-faulty processor. Since any non-faulty processor terminates the algorithm of Figure 13 in the phase after acceptance, all non-faulty processors must have terminated by the end of the second phase following acceptance of the first non-faulty processor, i.e., at most one round later.

With those preparations, it is not difficult to prove our major Theorem 8.

Theorem 8 (Hybrid Srikanth & Toueg) *Under the system model of Definition 1 with $f_a, f_s, f_o, f_c, f_\ell^r, f_\ell^{r,a}, f_\ell^s, f_\ell^{s,a} \geq 0$ and $n > 3f_a + 2f_s + 2f_o + f_c + f_\ell^s + f_\ell^{s,a} + 2f_\ell^r + 2f_\ell^{r,a}$, the algorithm of Figure 12 in conjunction with the simulated broadcast primitive of Figure 13 achieves binary Byzantine agreement in at most $2(f_a + f_s + f_o + f_c + 1)$ phases, with a total of at most $(2(f_a + f_s + f_o + f_c + 1) - 1)n^2 + n$ broadcasts of $(2\log_2 n + 2)$-bit messages from obedient processors.*

Proof: The major part of our theorem follows immediately by combining Theorem 6 with Theorem 7.

As far as the claimed time and message complexity is concerned, we know from Theorem 6 that the first accept of a non-faulty processor happens in round $f_a + f_s + f_o + f_c$ or earlier. Hence, any instance of the simulated broadcast primitive at an obedient processor must terminate by round $R = f_a + f_s + f_o + f_c + 1 \leq n$ and could generate at most $(2(f_a + f_s + f_o + f_c + 1) - 1)n + 1$ broadcasts of $(2\log_2 n + 2)$-bit messages by Theorem 7, since each of the at most n obedient processors calls **broadcast**( ) at most once by Theorem 6, the proof of Theorem 8 is completed. □

Theorem 6 in conjunction with the tight lower bound $n > f_\ell^r + f_\ell^{r,a} + f_\ell^s + f_\ell^{s,a}$ for consensus developed in [61] finally allows us to derive a—not necessarily tight—lower bound for the number of processors required for establishing *Correctness*, *Unforgeability* and *Relay*, cp. [67]: If there was a broadcast primitive that needs only $f_\ell^r + f_\ell^{r,a} + f_\ell^s + f_\ell^{s,a}$ processors, it would be possible to solve Byzantine agreement by means of the algorithm of Figure 12, thereby violating the lower bound for consensus. We thus have the following Theorem 9, which indicates that the hybrid simulated broadcast primitive of Figure 13 may not be optimal.

Theorem 9 (Lower Bound) *Achievement of Correctness (C), Unforgeability (U), and Relay (R) under the system model of Definition 1 requires* $n > f_\ell^r + f_\ell^{r,a} + f_\ell^s + f_\ell^{s,a}$ *processors.* □

5 Conclusion

This paper makes two contributions to the state of the art: (1) It shows that message-efficient deterministic consensus in synchronous systems is possible even in the presence of link faults, and (2) improves and adapts three existing algorithms by analyzing them for hybrid faults.

These adaptions were made using the novel preception-based fault model of [61], which grants every processor at most f_ℓ^r independent receive link faults and f_ℓ^s broadcast link faults in each round, in addition to at most f_a, f_s, f_o, f_c arbitrary, symmetric, omission, and manifest processor faults. The required number of processors for the Phase King and Queen Algorithms [17] and the non-authenticated algorithm of Srikanth & Toueg [67] are shown to satisfy

$$\begin{aligned} n &> 2f_\ell^s + 2f_\ell^r + 2f_\ell^{r,a} + 4f_a + 2f_s + 2f_o + f_c, \\ n &> 2f_\ell^s + 2f_\ell^r + 2f_\ell^{r,a} + 3f_a + 2f_s + 2f_o + f_c, \\ n &> f_\ell^s + f_\ell^{s,a} + 2f_\ell^r + 2f_\ell^{r,a} + 3f_a + 2f_s + 2f_o + f_c. \end{aligned}$$

Comparison with the (optimal) non-authenticated algorithm OMH of [61] shows that those algorithms are less resilient to arbitrary link faults. The resulting fault-tolerance degree $n > 2f_\ell^s + 2f_\ell^r$ does not match the lower bound $n > f_\ell^s + f_\ell^r$.

The question whether an optimal message-efficient algorithm exists remains open and is definitely one direction of future work. Another direction for further study may be the analysis of solutions for multi-valued agreement and for asynchronous systems.

List of Figures

List of Tables

References

[1] Hosame Abu-Amara and Jahnavi Lokre. Election in asynchronous complete networks with intermittent link failures. *IEEE Transactions on Computers*, 43(7):778–788, July 1994.

[2] Marcos Kawazoe Aguilera, Wei Chen, and Sam Toueg. Failure detection and consensus in the crash-recovery model. *Distributed Computing*, 13(2):99–125, April 2000.

[3] Hagit Attiya, Nancy Lynch, and Nir Shavit. Are wait-free algorithms fast? *Journal of the ACM*, 41(4):725–763, July 1994.

[4] Hagit Attiya and Jennifer Welch. *Distributed Computing*. McGraw-Hill, 1998.

[5] M. H. Azadmanesh and Roger M. Kieckhafer. New hybrid fault models for asynchronous approximate agreement. *IEEE Transactions on Computers*, 45(4):439–449, 1996.

[6] M.H. Azadmanesh and Roger M. Kieckhafer. Exploiting omissive faults in synchronous approximate agreement. *IEEE Transactions on Computers*, 49(10):1031–1042, October 2000.

[7] Anindo Banerjea. Fault recovery for guaranteed performance communications connections. *IEEE/ACM Transactions on Networking*, 7(5):653–668, 1999.

[8] A. Bar-Noy, D. Dolev, C. Dwork, and H.R. Strong. Shifting gears: changing algorithms on the fly to expedite Byzantine agreement. *Information and Computation*, 97(2):205–233, April 1992.

[9] Amotz Bar-Noy and Danny Dolev. Consensus algorithms with one-bit messages. *Distributed Computing*, 4:105–110, 1991.

[10] Amotz Bar-Noy, Danny Dolev, Cynthia Dwork, and H. Raymond Strong. Shifting gears: Changing algorithms on the fly to expedite byzantine agreement. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, pages 42–51, August 1987.

[11] R. Bar-Yehuda, S. Kutten, Y. Wolfstahl, and S. Zaks. Making distributed spanning tree algorithms fault-resilient. In Franz-Josef Brandenburg, Guy Vidal-Naquet, and Martin Wirsing, editors,

Lecture Notes in Computer Science, volume 247, pages 432–444. Springer, February 1987. Proceedings to STACS'97: 4th annual Symposium on Theoretical Aspects of Computer Science.

[12] Anindya Basu, Bernadette Charron-Bost, and Sam Toueg. Simulating reliable links with unreliable links in the presence of process crashes. In *Proceedings of the 10th International Workshop on Distributed Algorithms (WDAG'96)*, volume 1151 of *LNCS*, pages 105–122. Springer, October 1996.

[13] Piotr Berman and Juan A. Garay. Asymptotically optimal consensus. In *Proc. 16th International Colloquium on Automata, Languages and Programming*, pages 80–94, July 1989.

[14] Piotr Berman and Juan A. Garay. Efficient distributed consensus with $n = (3 + \epsilon)t$ processors. In *Proceedings of the 5th International Workshop on Distributed Algorithms*, pages 129–142, October 1991.

[15] Piotr Berman and Juan A. Garay. Cloture votes: $n/4$-resilient distributed consensus in $t + 1$ rounds. *Mathematical Systems Theory*, 26:3–19, July 1993.

[16] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Towards optimal distributed consenus. In *Proceedings of the 30th IEEE Symposium on the Foundations of Computer Science*, pages 410–415. IEEE Computer Society Press, 1989.

[17] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Asymptotically optimal distributed consensus. http://www.bell-labs.com/user/garay/#distributed-pub, 1992. (A combination of results from ICALP'89[13], FOCS'89[16], and WDAG'91[14]).

[18] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading, Mass., 1987.

[19] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(4), March 1996.

[20] Soma Chaudhuri. More *choices* allow more *faults*: Set consensus problems in totally asynchronous systems. *Information and Control*, 105(1):132–158, July 1993.

[21] Brian Coan. A communication-efficient canonical form for fault-tolerant distributed algorithms. In *Proccedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing*, pages 63–72, 1986.

[22] Brian Coan. Efficient agreement using fault diagnosis. *Distributed Computing*, 7, 1993. also in Proc 26th Allerton Conf. on Communication, Control and Computing, 1988.

[23] Brian Coan and Jennifer Welch. Modular construction of an efficient 1-bit byzantine agreement protocol. *Mathematical Systems Theory*, 26:131–154, July 1993.

[24] Danny Dolev. The Byzantine generals strike again. *Journal of Algorithms*, 3(1):14–30, 1982.

[25] Danny Dolev, Michael J. Fischer, R. Fowler, N. Lynch, and H. Raymond Strong. An efficient algorithm for Byzantine agreement without authentication. *Information and Control*, 52:257–274, 1982.

[26] Danny Dolev, Joseph Y. Halpern, Barbara Simons, and Ray Strong. Dynamic fault-tolerant clock synchronization. *Journal of the Association for Computing Machinery*, 42(1):143–185, January 1995.

[27] Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and H. Raymond Strong. Reaching approximate agreement in the presence of faults. *Journal of the ACM*, 33(3):499–516, July 1986.

[28] Danny Dolev, Rudiger Reischuk, and H. Raymond Strong. Early stopping in byzantine agreement. *Journal of the ACM*, 37(4):720–741, October 1990.

[29] Danny Dolev and H. Raymond Strong. Polynomial algorithms for multiple processor agreement. In *Proceedings 14th Annual ACM Symposium on Theory of Computing (STOC'82)*, pages 401–407, San Francisco, May 1982.

[30] A. D. Fekete. Asymtotically optimal algorithms for approximate agreement. *Distributed Computing*, 4(1):9–29, March 1990.

[31] A. D. Fekete. Asynchronous approximate agreement. *Information and Computation*, 115(1):95–124, November 15 1994.

[32] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistancy. *Information Processing Letters*, 14(4):198–202, May 1982.

[33] Michael J. Fischer, Nancy A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty processor. *Journal of the ACM*, 32(2):374–382, April 1985.

[34] Juan A. Garay and Yoram Moses. Fully polynomial byzantine agreement in t+1 rounds. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 31–41, San Diego, May 1993. Extended abstract of [35].

[35] Juan A. Garay and Yoram Moses. Fully polynomial byzantine agreement for $n > 3t$ processors in $t + 1$ rounds. *SIAM Journal of Computing*, 27(1):247–290, February 1998.

[36] Oded Goldreich and Dror Sneh. On the complexity of global computation in the presence of link failures: The case of uni-directional faults. In *Symposium on Principles of Distributed Computing*, pages 103–111, 1992.

[37] Li Gong, Patrick Lincoln, and John Rushby. Byzantine agreement with authentication: Observations and applications in tolerating hybrid and link faults. In *Proceedings Dependable Computing for Critical Applications-5*, pages 139–157, Champaign, IL, September 1995.

[38] J.N. Gray. Notes on data base operating systems. In G. Seegmüller R. Bayer, R.M. Graham, editor, *Operating Systems: An Advanced Course*, volume 60 of *Lecture Notes in Computer Science*, chapter 3.F, page 465. Springer, New York, 1978.

[39] R. Guerraoui, R. Oliveira, and A. Schiper. Stubborn communication channels. Technical report, D'eoartment d'Informatique, Ecole Polytechnique F'ed'erale de Lausanne, Switzerland, 96.

[40] R.M. Kieckhafer and M.H. Azadmanesh. Reaching approximate agreement with mixed failure modes. *IEEE Transactions on Parallel and Distributed Systems*, 5(1):53–63, January 1994.

[41] R.M. Kieckhafer and M.H. Azadmanesh. Unified approach to synchronous and asynchronous approximate agreement in the presence

of hybrid faults. *IEEE Transactions on Reliability*, 44(4):622–631, December 1995.

[42] Leslie Lamport. The weak byzantine generals problem. *Journal of the ACM*, 30:668–676, July 1983.

[43] Leslie Lamport, Robert Shostak, and Marshall Pease. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[44] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[45] Patrick Lincoln and John Rushby. A formally verified algorithm for interactive consistency under a hybrid fault model. In *Proceedings Fault Tolerant Computing Symposium 23*, pages 402–411, Toulouse, France, June 1993.

[46] Nancy Lynch. *Distributed Algorithms.* Morgan Kaufman, 1996.

[47] Stephen R. Mahaney and Fred B. Schneider. Inexact agreement: Accuracy, precision, and graceful degradation. In *Proceedings 4th ACM Symposium on Principles of Distributed Computing*, pages 237–249, Minaki, Canada, August 1985.

[48] J. McAlpin and J. Liu. An adaptive routing algorithm for a network with distributed control. In *Proceedings of the ISMM International Conference. Parallel and Distributed Computing, and Systems*, page 409 pp. Acta Press, Anaheim, CA, USA, 1990.

[49] Fred J. Meyer and Dhiraj K. Pradhan. Consensus with dual failure modes. In *In Digest of Papers of the 17th International Symposium on Fault-Tolerant Computing*, pages 48–54, Pittsburgh, July 1987.

[50] Yoram Moses and Orli Waarts. Coordinated traversal: (t+1)-round byzantine agreement in polynomial time. *Journal of the ACM*, 17(1):110–156, July 1994.

[51] Kenneth J. Perry and Sam Toueg. Distributed agreement in the presence of processor and communication faults. *IEEE Transactions on Software Engineering*, SE-12(3):477–482, March 1986.

[52] S. S. Pinter and I. Shinahr. Distributed agreement in the presence of communication and process failures. In *Proceedings of the 14th IEEE Convention of Electrical & Electronics Engineers in Israel.* IEEE, March 1985.

[53] R. Plunkett and A. Fekete. Approximate agreement with mixed mode faults. In *Proccedings of the 12th International Symposium on Distributed Computing*, pages 333–346. Springer Lecture Notes in Computer Science 1499, September 1998.

[54] David Powell. Failure mode assumptions and assumption coverage. In *Proc. 22nd IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-22)*, pages 386–395, Boston, MA, USA, 1992. (Revised version available as LAAS-CNRS Research Report 91462, 1995).

[55] Rfc index. Technical report, Internet Engineering Task Force, `http://www.ietf.org/iesg/1rfc_index.txt`, 1969.

[56] John Rushby. A formally verified algorithm for clock sychronization under a hybrid fault model. In *Proceedings ACM Principles of Distributed Computing (PODC'94)*, pages 304–313, Los Angeles, CA, August 1994.

[57] Hasan M. Sayeed, Marwan Abu-Amara, and Hosame Abu-Amara. Optimal asynchronous agreement and leader election algorithm for complete networks with Byzantine faulty links. *Distributed Computing*, 9(3):147–156, 1995.

[58] Ulrich Schmid. Orthogonal accuracy clock synchronization. *Chicago Journal of Theoretical Computer Science*, 2000(3):3–77, 2000.

[59] Ulrich Schmid. How to model link failures: A perception-based fault model. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'01)*, pages 57–66, Göteborg, Sweden, July 1–4, 2001.

[60] Ulrich Schmid and Klaus Schossmaier. How to reconcile fault-tolerant interval intersection with the Lipschitz condition. *Distributed Computing*, 14(2):101 – 111, May 2001.

[61] Ulrich Schmid and Bettina Weiss. Consensus with oral/written messages: Link faults revisited. Technical Report 183/1-110, Department of Automation, TU Vienna, February 2001.

[62] Ulrich Schmid, Bettina Weiss, and John Rushby. Formally verified byzantine agreement in presence of link faults. 2001. (submitted).

[63] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, second edition, 1996.

[64] Gurdip Singh. Leader election in the presence of link failures. *IEEE Transactions on Parallel and Distributed Systems*, 7(3):231–236, March 1996.

[65] Hin-Sing Siu, Yeh-Hao Chin, and Wei-Pang Yang. Byzantine agreement in the presence of mixed faults on processors and links. *IEEE Transactions on Parallel and Distributed Systems*, 9(4):335–345, 1998.

[66] T. K. Srikanth and Sam Toueg. Optimal clock synchronization. *Journal of the ACM*, 34(3):626–645, July 1987.

[67] T.K. Srikanth and Sam Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2:80–94, 1987.

[68] W. Richard Stevens. *TCP/IP Illustrated*, volume I. Addison-Wesley, 1994.

[69] P. M. Thambidurai and Y. K. Park. Interactive consistency with multiple failure modes. In *Proceedings 7th Reliable Distributed Systems Symposium*, October 1988.

[70] Sam Toueg, Kenneth J. Perry, and T. K. Srikanth. Fast distributed agreement. *SIAM Journal on Computing*, 16(3):445–457, 1987.

[71] Chris J. Walter and Neeraj Suri. The customizable fault/error model for dependable distributed systems. *Theoretical Computer Science*, 2000. (Special issue on Dependable Computing, to appear).

[72] Chris J. Walter, Neeraj Suri, and M. M. Hugue. Continual on-line diagnosis of hybrid faults. In *Proceedings DCCA-4*, January 1994.

[73] Lidong Zhou, Fred B. Schneider, and Robbert van Renesse. COCA: A secure distributed on-line certification authority. Technical Report TR2000-1828, Computer Science Department, Cornell University, December 2000.