Shifting Gears: Changing Algorithms on the Fly To Expedite Byzantine Agreement

Preliminary Report

Amotz Bar-Noy and Danny Dolev
Hebrew University

Cynthia Dwork and H. Raymond Strong
IBM Almaden Research Center

Section 1. Introduction

In designing distributed algorithms it is often impossible to combine different algorithms for the same problem; while the hope is that the strengths reinforce, the reality is that the weaknesses conspire. In this paper we present three Byzantine agreement algorithms, of resilience $\frac{n-1}{3}$, $\frac{n-1}{4}$, and $(n/2)^{1/2}$ respectively, for which it is possible to shift, mid-execution, from one to another, where n denotes the total number of processors in the system. Thus, one may begin an execution using an inefficient but highly resilient algorithm, and, after a predetermined number of rounds of communication, shift to a more efficient algorithm of lower resilience, even though the actual number of faulty processors remains high. Shifting between algorithms of different resiliences is possible in both directions, even if the overall tolerance to faults must remain high. To our knowledge the ability to shift, particularly between algorithms of different resiliences, has not previously been demonstrated.

We have identified three key properties shared by all our' algorithms that in combination capture our intuition of why it is possible to shift between the algorithms. These properties are called "persistence," "fault detection," and "fault masking." At all times during execution of our algorithms each correct processor has a "preferred" candidate decision value. "Persistence" says that if sufficiently many correct processors "prefer" v, then this situation persists, and the eventual decision value will be v. A faulty processor is *detected* if all correct processors have discovered it to be faulty. (We distinguish between *discovery*, which describes the action of a single processor, and *detection*, in which all correct processors have discovered the same faulty processor. These discoveries need not take place simultaneously.) Messages from processors known to be faulty are ignored. Thus the actions of detected processors are essentially "masked". The fault detection and fault masking properties allow us to shift from an algorithm of high resilience down to one of lower resilience if many faults have occurred early in the execution, while the persistence property allows us to shift down if there were few faults early on, despite the fact that more faults may occur later.

Our two algorithms of linear resilience are actually families of algorithms interesting in their own right, as they achieve the message rounds versus number of message bits tradeoff exhibited by Coan's families [C1], but avoid the exponential local computation of his algorithms. In addition, these algorithms and their proofs of correctness are dramatically simpler than those of Coan.

The information transfer, local computation, and rounds of communication required for our three algorithms are stated in Theorems 1-3.

Theorem 1: *For $2 < b \leq t$, Byzantine agreement can be achieved in the presence of $t < n/3$ faults in $2 + \lceil \frac{t-1}{b-2} \rceil b$ rounds of communication, using messages of $O(n^b)$ bits. Moreover, the amount of local computation at each processor is $O(n^{b+1}(\frac{t-1}{b-2}))$.*

Theorem 2: *For $1 < b \leq t$, Byzantine agreement can be achieved in the presence of $t < n/4$ faults in $2 + \lceil \frac{t-1}{b-1} \rceil b$ rounds of communication, using messages of*

$O(n^b)$ bits. Moreover, the amount of local computation at each processor is $O(n^{b+1}(\frac{t-1}{b-1}))$.

Theorem 3([DRS]): *If $n > 2t^2 - 3t + 2$ then t-resilient Byzantine agreement can be achieved in $t + 1$ rounds using messages of $O(n)$ bits. Moreover, the amount of local computation at each processor is at most $O(n^{2.5})$.*

Main Theorem: *It is possible to combine the algorithms used to prove Theorems 1-3 to obtain hybrid algorithms. In particular, for $2 < b \leq t$, there is a hybrid of all three algorithms with resilience $t = \frac{n-1}{3}$ requiring*

$$1 + (\lceil\frac{x-1}{b-2}\rceil + \lceil\frac{x-\sqrt{x}}{b-1}\rceil)b + \lceil\sqrt{x}\rceil$$

rounds of communication, where $x = t/2$. The bounds on information transfer and local computation are as in Theorem 1.

The remainder of this extended abstract is organized as follows. In Section 2 we briefly define the Byzantine agreement problem and specify our model of computation. In Section 3 we provide an exponential (in t) information gathering algorithm very similar to the original algorithm for Byzantine agreement [PSL]. We briefly outline a new proof that this algorithm achieves Byzantine agreement in $t + 1$ rounds in the presence of $t < \frac{n}{3}$ faults. In addition to providing certain lemmas for the correctness proofs of our new algorithms, the new proof provides intuition that was critical in our discovery of the new algorithms. We will show how to modify the exponential algorithm by introducing fault discovery and fault masking so that the resulting algorithm exhibits persistence and fault detection. In Section 4 we introduce *shifting* and apply it to our modified information gathering algorithm to produce our three families of algorithms and the hybrid algorithm of our Main Theorem. Concluding Remarks appear in Section 5.

2. Model Description and Problem Statement

We assume a completely synchronous system of n processors connected by a fully reliable complete network. Each processor has a unique identification number over which it has no control. At any point in the execution of the protocol processors may fail. There is no restriction on the behavior of faulty processors, and we do not assume the existence of authentication mechanisms. However, a correct processor can always correctly identify the source of any message it receives. This is the standard "unauthenticated Byzantine" fault model.

Processing is completely synchronous. Not only do the processors communicate in synchronous rounds of communication, but they all begin processing in the same round. Without loss of generality we refer to this round as Round 1.

In the Byzantine agreement problem one distinguished processor, called the *source*, begins with a single input value v drawn from a finite set V (without loss of generality we assume $0 \in V$). We view $|V|$ as constant. (If $|V|$ is very large we may apply techniques of Coan ([C2]) to reduce the set to two elements, at the cost of two rounds.) The goal is for the source to broadcast v and for all other processors to agree on the value broadcast. That is, at some point in the computation each correct processor must irreversibly *decide* on a value. The requirements are that no two correct processors decide differently, and that if the source is correct then the decision value is the value broadcast by the source.

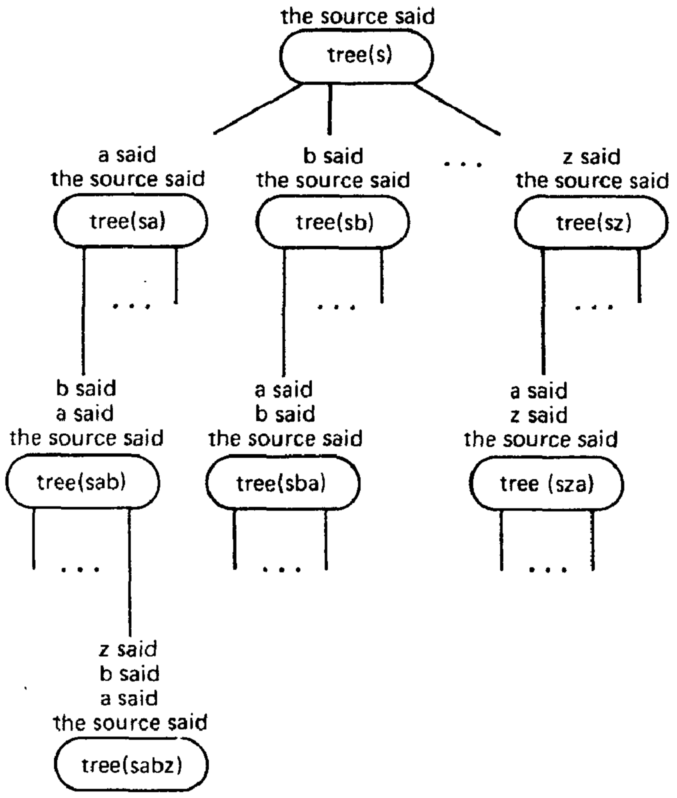
An algorithm for Byzantine agreement has *resilience t* if correct processors following the algorithm are guaranteed to reach Byzantine agreement provided the number of faulty processors does not exceed t. No noncryptographic protocol for Byzantine agreement can tolerate $n/3$ faults [PSL]. Thus, since our results are trivial for $t = 0$, we will assume from now on that the resilience to be achieved is at least 1 and the number of processors is at least 4.

3. The Exponential Algorithm

In this section we describe an algorithm similar to the original Byzantine Agreement algorithm of Pease, Shostak, and Lamport [PSL]. A descriptive, but cumbersome name for our algorithm is "Exponential Information Gathering with Recursive Majority Voting." Henceforth we refer to this algorithm as "the exponential algorithm."

In the exponential algorithm each processor maintains a large tree of height t (each path from root to leaf contains $t + 1$ vertices). The vertices are labelled with processor names as follows. The root is labelled s, for *source*. Let v be an internal node in the tree. For every processor name p not labelling an ancestor of v, v has exactly one child labelled p. With this definition no label appears twice in any path from root to leaf in the tree. Thus, we say this tree is *without repetitions*. (In Algorithm C, described in Section 4, we will extend the tree to include repetitions. In that case all internal nodes have n children.) Note that we may refer to a vertex in the tree by specifying the sequence of labels encountered by traversing the path from the root to the vertex. Let α be such a sequence. The *length* of α is the length of the sequence. The *processor corresponding to vertex* α is the processor whose name labels vertex α, i.e., the last processor name in the sequence α.

In the first round of the information gathering algorithm the source sends its initial value to all $n - 1 \geq 3t$



The information gathering tree

Figure 1:

other processors. When a correct processor p receives its message from the source it stores the received value at the root of its tree (a default value of $0 \in V$ is stored if the source fails to send a legitimate value in V). In each subsequent round each processor broadcasts the level of its tree most recently filled in. With the messages received each processor adds a new level to the tree, storing at vertex $s \ldots bq$ the value that q claims to have stored in vertex $s \ldots b$ in its own tree (again, a default is used if an inappropriate message is received). Thus, intuitively, p stores in vertex $s \ldots bq$ the value that "q says b says ... the source said" (see Figure 1). We refer to this value as $tree_p(s \ldots bq)$, eliminating the subscript p when no confusion will arise.

The value stored in $tree_p(s)$ (i.e., at the root) is called the *preferred* value of p. Information is gathered for $t+1$ rounds, until the entire tree has been filled in. At that point each processor p applies to the tree a recursive data reduction function, called *resolve*, to obtain a new preferred value which we denote $resolve_p(s)$ (we drop the subscript p when no confusion arises).

The value obtained by applying a reduction function to the subtree rooted at a vertex α is called the *reduced value for* α. The specific data reduction function used in the exponential algorithm *resolve*, is essentially a recursive majority vote, and is defined as follows:

$resolve(\alpha) =$

$tree(\alpha)$, if α is a leaf;

the majority value obtained by applying *resolve* to the children of α, if a majority exists;

0, if α is not a leaf and no majority exists.

The entire exponential algorithm is: gather information for $t+1$ rounds; compute the reduced value for s using the data reduction function *resolve*; decide on this reduced value.

We now sketch a proof of correctness for this algorithm.

During the data reduction stage of the algorithm a vertex α is *common* if each correct processor computes the same reduced value for α. Thus the algorithm is correct if and only if

1. in every execution s is common, and
2. if the source is correct, every correct processor reduces s to the value received from s in round 1 ($tree(s)$).

If the source is correct these conditions are guaranteed by the following lemma in the special case $\alpha = s$.

Correctness Lemma: *Any node α in the information gathering tree that corresponds to a correct processor is common and satisfies $resolve_p(\alpha) = tree_p(\alpha)$ for every correct processor p.* □

The proof of the Correctness Lemma, omitted here, relies on the fact that a strict majority of the children of every non-leaf in an information gathering tree correspond to correct processors. This is true because by construction every internal vertex has at least $2t+1$ children, of which at most t are faulty.

There are at most t faulty processors, and every path in the information gathering tree is of length $t+1$, so every path from root to leaf contains a correct processor. It therefore follows by the Correctness Lemma that every path contains a common vertex, independent of whether or not the source is correct. When every root-leaf path contains a common vertex we say the collection of information gathering trees of correct processors has a *common frontier*.

As we have seen, the Correctness Lemma says the algorithm works if the source is correct. We have also

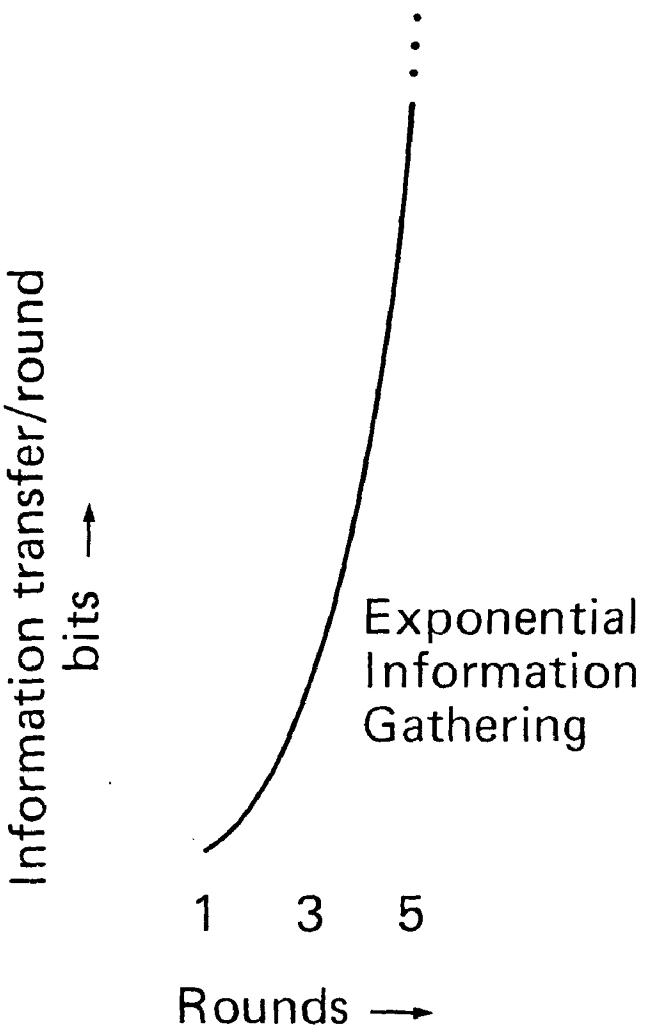


Figure 2:

observed that in every execution in which at most t processors fail there is a common frontier, independent of whether the source is correct. It remains to show that even if the source is faulty the existence of a common frontier guarantees agreement. This is immediate from the the following lemma.

Frontier Lemma: *If there is a common frontier, then s is common.* □

To prove the Frontier Lemma we actually prove the more general claim:

Claim: Let α be a vertex. If there is a common frontier in the subtree rooted at α, then α is common (i.e., α itself constitutes a common frontier of the subtree).

The claim is proved by induction on the height of α (backwards induction on the length of α).

In light of the above discussion we have the following proposition.

Proposition: *The Information Gathering Algorithm reaches Byzantine agreement in $t+1$ rounds tolerating $t < n/3$ faults.* □

We have shown that this simple variant of the original [PSL] algorithm reaches Byzantine agreement in the optimal number of rounds [FL,DS]. In spite of the simplicity of the algorithm, the message size and the amount of local computation required grow exponentially with t (see Figure 2). Later, to bound this growth, we will apply a *shift* operator to reduce message size when it threatens to exceed our bound. However, before we can apply shifting, we must modify the algorithm for fault detection and prove that the modified algorithm exhibits the three important properties, persistence, fault detection, and fault masking, mentioned in the Introduction.

We begin with the *persistence* property. Generally speaking, the persistence property says that if "enough" correct processors share the same preferred value before data reduction, then after reduction s is common. The choice of how many processors constitute "enough" may depend on the particular algorithm involved and the intended application of the persistence property. In our case the requirement is spelled out in the Persistence Lemma below. The intended application is in construction of our hybrid algorithm, discussed in Section 4.

Persistence Lemma: *If before reduction some set of correct processors, sharing the same preferred value v, constitute a strict majority of all processors, then s is common and has reduced value v.* □

The Persistance Lemma follows easily from the Correctness Lemma and the choice of reduction function. The value v described in the Persistence Lemma is called a *persistent* value.

For any $k < t+1$, if information gathering is run for only k rounds, then the Correctness, Frontier, and Persistence Lemmas hold, even though the paths in the information gathering trees contain only k vertices. Moreover, these lemmas hold if the preferred value of each processor is a private initial value rather than the contents of a message from the source. Thus, we could run the information gathering algorithm for k rounds, reduce the resulting tree to produce a single value *resolve*(s), and treat this value as if it had been received directly from the source, storing it in *tree*(s) and continuing with the information gathering algorithm as if it had just finished round 1. It is not difficult to argue that any algorithm constructed along these lines will work correctly if the source is not faulty. However, because $k < t+1$ we are not guaranteed a common frontier if the source is faulty. In this case faults other than the source may be able to collude to prevent the emergence of a persistent value. In order to bound the number of times this can occur we introduce here fault discovery and fault masking rules to be followed by each processor. The intuition we wish to capture is that if a faulty processor is effective at preventing the emergence of a persistent value, then that processor is detected and subsequently ignored.

We modify the exponential algorithm by giving each processor p an extra data structure, L_p (the subscript is omitted when no confusion will arise). L_p, initially empty, contains the names of processors that p has discovered to be faulty by applying the Fault Discovery Rule stated below.

We will need the following definition. For all internal vertices β, a value stored at a strict majority of the children of β is called the *majority value for* β.

Fault Discovery Rule: Let p be a correct processor. During information gathering, a processor b not already in L_p is added to L_p if for some internal vertex αb in $tree_p$

1. there is no majority value for αb, or
2. a majority value for αb exists but other values are stored at more than $t - |L_p|$ children of αb not corresponding to processors already in L_p.

If at most t processors fail and L_p contains only faulty processors, then any processor added to L_p under the Fault Discovery Rule is necessarily faulty.

The Fault Discovery Rule has an extremely useful corollary, the Hidden Fault Lemma. It is helpful to think of the faulty processors as being controlled by an adversary. The Persistence Lemma implies that in order to prevent the occurrence of a persistent value the adversary must arrange to split the correct processors into at least two sets, neither of which has size $\frac{n-1}{2}$, where processors in different sets prefer different values. The Hidden Fault Lemma will be used to show that, if the adversary is successful in preventing a persistent value, then some faulty processor is discovered by all correct processors, i.e., it is detected. Once a faulty processor is detected, all correct processors view it as sending only the default value, so it can never again be used to split the correct processors.

Hidden Fault Lemma: *Let p be a correct processor and let αb be any internal vertex in p's information gathering tree. Let k be the length of αb and let m be the number of children of αb. If all the processors in αb are faulty, but $b \notin L_p$ after round $k+1$ (i.e., after p stores values at the children of αb), then the set of processors corresponding to the children of αb at which the majority value is stored contains at least $m - t + |L_p|$ nodes corresponding to correct processors.* □

Let p be a correct processor and q a faulty processor. Any correct agreement protocol must be able to tolerate any behavior of q, provided the resilience of the protocol is not exceeded. In particular, if q were always to send zeros to p, regardless of what q should be sending, the protocol should still work. This can be proved formally, providing a justification for the following Fault Masking Rule.

Fault Masking Rule: If b is added to L in round r, then any messages from b in round r and any subsequent round are replaced by messages in which each value is the default 0. In other words, once a processor discovers that b is faulty, it "acts as if" b sends only zeros.

Under the Fault Masking Rule, once a processor has been discovered faulty by all correct processors, it is essentially forced to send the same values (zeros) to all correct processors. As we will see, in the new algorithms fault masking will limit the ability of a faulty processor to prevent the correct processors from obtaining a persistent value.

We assume for the rest of this paper that the fault discovery and fault masking rules are applied in each round of information gathering. However, we stress that Fault Masking is never used to fill in the root, $tree(s)$.

Section 4. Shifting

All of our new algorithms are based on applications of shifting to the exponential algorithm with fault discovery and fault masking. We introduce an operator $Shift_j^k$ that uses some conversion process to change the data structures appropriate to the end of round k into those appropriate to the end of round j. This operator can be applied repeatedly to prevent the data structures from growing past the size of those associated with the end of round k (see Figures 3 and 4). In order to specify an algorithm that uses shifting, we need only specify the original algorithm, the points at which shifting is to take place, and the conversion process. When we convert from larger to smaller data structures, we refer to the process as *compression*. We can even specify shifting from one algorithm to another, if we can specify an appropriate compression process. However, there is no guarantee that indiscriminate shifting will result in algorithms that achieve the desired objective of Byzantine agreement.

In this paper we focus on three algorithms, two of which take a parameter that yields a family of algorithms when varied, and show that it is possible to shift among these algorithms.

Let $t_A = \lfloor \frac{n-1}{3} \rfloor$, $t_B = \lfloor \frac{n-1}{4} \rfloor$, and $t_C = \lfloor (n/2)^{1/2} \rfloor$.

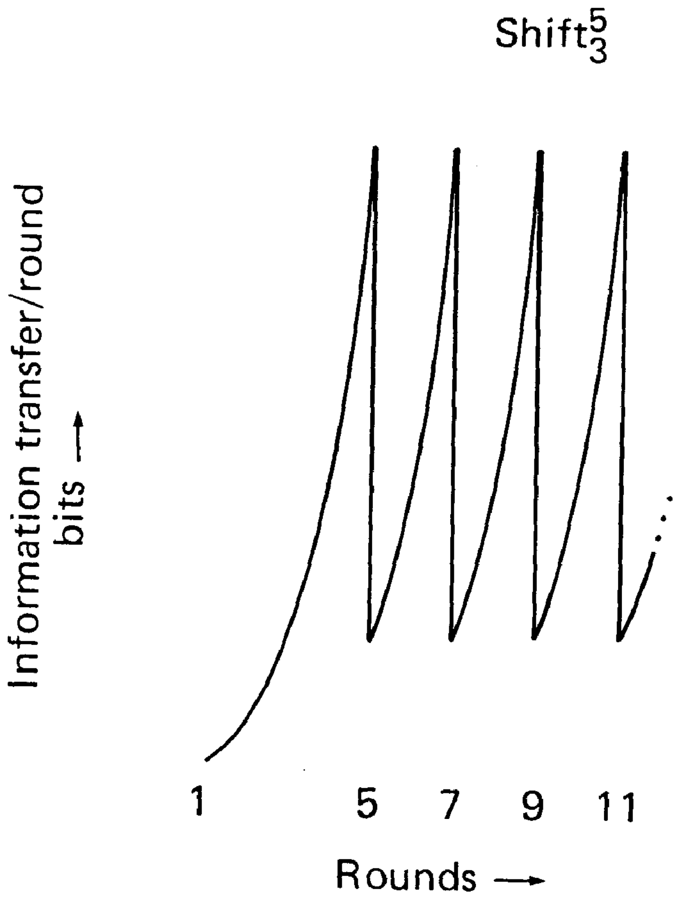


Figure 3:

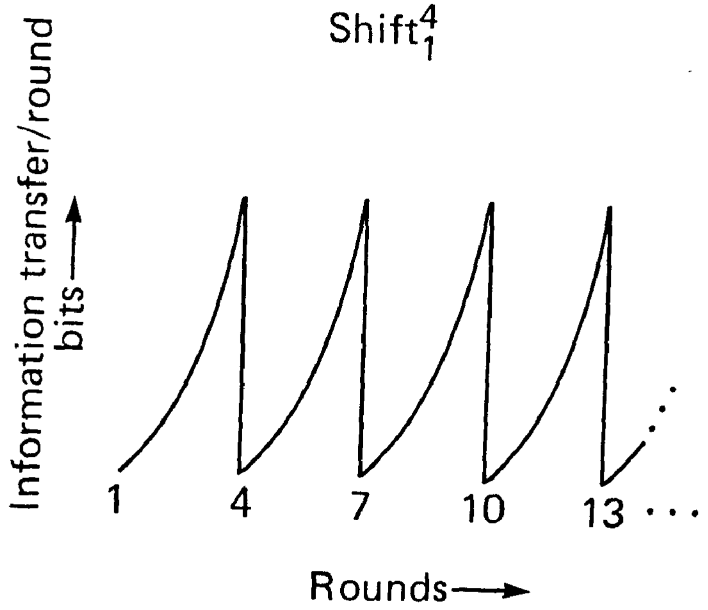


Figure 4:

Algorithm A has resilience t_A. When run with parameter d, $1 \leq d \leq t - 2$, Algorithm A requires $\lceil \frac{t+1}{d} \rceil (d + 2)$ rounds of communication and employs messages of length strictly less than n^{d+1}. The total local processing time expended by each processor is $O(n^{d+2} \lceil \frac{t+1}{d} \rceil)$. (One can do slightly better in the special cases $d = t - 1$ and $d = t$.)

Algorithm B has resilience t_B. When run with parameter d, $1 \leq d \leq t - 1$, Algorithm B requires $\lceil \frac{t+1}{d} \rceil (d + 1)$ rounds of communication and employs messages of length strictly less than n^d. The total local processing time expended by each processor is $O(n^{d+1} \lceil \frac{t+1}{d} \rceil)$. (One can do slightly better in the special case $d = t$.)

Algorithm C has resilience t_C, and is closely related to the early stopping algorithm of Dolev, Reischuk, and Strong [DRS] of the same resilience. This algorithm requires $t_C + 1$ rounds of communication and employs messages of length at most $n - 1$. Local processing time is $O(tn^2)$.

We describe algorithm B first because it is simplest. Algorithm B is simply the repeated application of $Shift_1^d$ to the exponential information gathering algorithm. Figure 4 shows the pattern of information gathering with $d = 4$. Data compression is accomplished by applying the *resolve* function of the previous section to obtain a reduced value for s.

If the number of faults is bounded by t_B then the Hidden Fault Lemma has an important corollary.

Corollary 1: *Let n be fixed and let the number of faults be bounded by t_B. Let αb be an internal vertex in the information gathering tree, and let all processors in αb be faulty. If under the reduction function resolve αb is not common, then all correct processors discover that b is faulty.*

Proof: For the sake of contradiction let us assume correct processors p and q compute different reduced values for αb and that q does not discover b to be faulty. By the Hidden Fault Lemma the majority value for αb in $tree_q$ is stored in at least

$$n - 2t_B + |L_q| \geq n - 2t_B > \frac{n-1}{2}$$

children of αb corresponding to correct processors, contradicting the assumption that αb is not common. (We are using here the fact that $n > 4t_B$.) □

Proposition: *Algorithm B solves Byzantine agreement and achieves the bounds on resiliency, message length, and number of rounds of communication stated in Theorem 2.*

Proof Sketch:

If the source is correct then by definition there is a persistent value, to wit, the value that the source broadcast in round 1. However a persistent value is obtained, the Persistence Lemma implies that at the next application of reduction s will be common.

We now consider the case in which the source is faulty and there is no persistent value. Consider the tree just before reduction. By the Frontier Lemma, if there is a common frontier, then s is common. We therefore need only consider the case in which there is a path ρ from root to leaf containing no common node. By the Correctness Lemma all processors corresponding to vertices in ρ are faulty. By Corollary 1 these faults are all detected.

With the exception of the source, which is repeatedly detected, once a processor is detected, nodes corresponding to it are common. This is because faults other than the source are masked according to the Fault Masking Rule. Thus each block of $d-1$ rounds that produces trees without a common frontier results in the detection of at least $d-2$ new faults in addition to the source. Let us write $t_B - 1 = (d-2)x + y$, where $y < d-2$. Then the number of rounds required by Algorithm B to reach Byzantine agreement is $(d-1)x + y + 2$. □

In order to improve upon the resilience of Algorithm B we modify the data reduction function of the exponential information gathering algorithm and apply $Shift_1^d$ to the resulting algorithm.

The new reduction function, $resolve'$, is defined as follows:

$resolve'(\alpha) =$

$tree(\alpha)$, if α is a leaf;

the unique value occurring at least $t_A + 1$ times among the values obtained by applying $resolve'$ to the children of α, if one exists;

$\perp$, if α is not a leaf and no such unique value exists.

Note that we have introduced a new value, $\perp$. Although used during the reduction process, $\perp$ is never used in the information gathering tree itself. If, at the end of some reduction, $resolve'_p(s) = \perp$ for some correct processor p, then p uses the default value (0) as its new preferred value.

Note that the Persistence Lemma as stated in Section 3 no longer holds when the reduction function used is $resolve'$ and up to $\frac{n-1}{3}$ processors may fail. However, a weaker version of this lemma does hold.

Weak Persistence Lemma: *If before reduction all correct processors prefer the same value v, then after reduction s is common and has reduced value v.* □

The exponential information gathering algorithm solves Byzantine agreement using either of $resolve$ or $resolve'$. Moreover, this holds for any set V of legitimate input values. For technical reasons Algorithm A, obtained by applying $Shift_1^d$ to the exponential information gathering algorithm modified to use $resolve'$ in the reduction process, can only handle sets V of cardinality 2. In order to allow it to handle arbitrary sets V we increase the power of the Fault Discovery Rule by applying it during the reduction process.

Fault Discovery Rule During Reduction: During reduction a processor b not already in L is added to L if for some internal vertex αb corresponding to b

1. there is no majority value among the reduced values for the children of αb, or
2. such a majority value v exists, but for more than $t_A - |L|$ processors $y \notin L$, $resolve'(\alpha b y) \neq v$.

Claim: The proofs of the Correctness, Frontier, and Hidden Fault Lemmas hold when the reduction function $resolve'$ is used in place of $resolve$.

The Hidden Fault Lemma has two new corollaries, one from the new choice of reduction function, and one from the additional fault discovery rule.

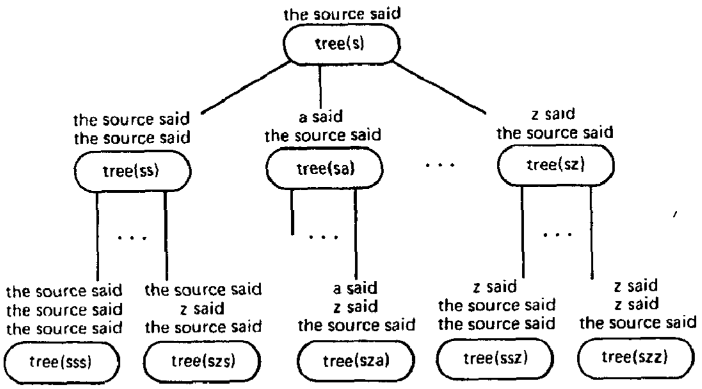
Corollary 1 said that if an internal vertex is not common then its corresponding processor is discovered. However, the proof of Corollary 1 relied on the assumption that the number of faults does not exceed t_B. Something slightly weaker than Corollary 1 holds even if the number of faults reaches t_A. Moreover, this weaker result can be used to show that Corollary 1 does indeed hold for all vertices of height at least 2 in the presence of up to t_A faults.

Corollary 2: *Let αb be an internal vertex in the information gathering tree, and let all processors in αb be faulty. If under the reduction function $resolve'$ two correct processors p and q obtain different reduced values for αb, neither of which is $\perp$, then both p and q discover b to be faulty during reduction of αb.*

Corollary 3: *Let αb be an internal vertex in the information gathering tree that is not the parent of a leaf. If all processors in αb are faulty, and if some correct processor q does not discover b either by the Fault Discovery Rule or the Fault Discovery Rule During Reduction, then αb is common.*

We can now prove the following proposition.

Proposition: *Algorithm A solves Byzantine agreement and achieves the bounds on resiliency, message length, and number of rounds of communication stated in Theorem 1.*



Reordering of the leaves and repetitions for algorithm C.

Figure 5:

Proof Sketch:

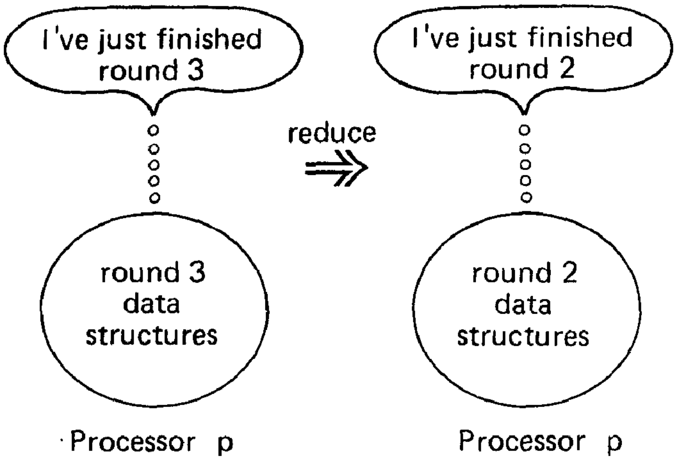
If the source is correct then after round 1 all correct processors prefer the same value, so by the Weak Persistence Lemma s will be common with reduced value v after reduction.

As in the proof of the previous proposition, if there is a common frontier then s is common. We therefore discuss only the case in which the source is faulty and the information gathering tree after d rounds contains a path ρ containing no common nodes. Once again the Correctness Lemma implies that all processors corresponding to vertices on ρ are faulty. Let αb be the label of a node on ρ that is neither a leaf nor the parent of a leaf. Let us assume there exists a correct processor q that has not discovered b to be faulty. By Corollary 3 αb is common. This implies that each block of $d-1$ rounds that produces trees without a common frontier results in the detection of at least $d-3$ new faults in addition to the source. Let us write $t_A - 1 = (d-3)x + y$, where $y < d-3$. Then the number of rounds required by algorithm A to reach Byzantine agreement is $(d-1)x+y+2$. □

We now describe Algorithm C. Consider first the following 3 round algorithm.

- Run information gathering for three rounds building a tree *with repetitions* and performing fault discovery and fault masking at each round.
- Reorder the leaves of the resulting tree of height 2 so that $tree(spq) \leftarrow tree(sqp)$ for all processors p and q (see Figure 5).

After the reordering, the leaves in the subtree rooted at sq are the values received from q in round 3. Algorithm C is the repeated application of $Shift_2^3$ to this



Application of Shift_2^3 at processor p

Figure 6:

3 round algorithm. The compression needed for the shift is achieved by computing reduced values for all parents of leaves according to the function *resolve*. The resulting two level tree is taken as the data structure produced after virtual round 2, $resolve(s)$ being the new preferred value. Thus, beginning with the third actual round of the 3 round algorithm a reduction is performed after each round. Fault Discovery is applied each round to the original tree (leaves ordered as in Figure 1) before reduction.

Proposition: *Algorithm C solves Byzantine agreement and achieves the bounds on resiliency, message length, and number of rounds of communication stated in Theorem 3.*

Proof Sketch (based on [DRS]):

It is easy to show that the Persistence Lemma holds for Algorithm C, so, if the source is correct, then all correct processors will agree on its value.

It remains to show that after the first round of Algorithm C if there is a round in which no new fault is detected during information gathering, then after reduction a persistent value is obtained. Moreover, we also show that at the end of the earliest round in which all t_C faults have been discovered a persistent value is obtained. The second claim is used to show that $t_C + 1$ rounds suffice even if only one fault is discovered in each of rounds 2 through $t_C + 1$.

In round 2, if the source is not detected, then some processor a does not discover the source to be faulty. By the Hidden Fault Lemma, there is a value v stored at at least $n - t_C$ children of s in $tree_a$. Thus at least $n - 2t_C$ correct processors had v as preferred value after round 1.

Recall that we have assumed throughout that $n > 3$. Thus $n - 2t_C$ is a majority of the n processors, and so v is a persistent value. In particular, if a correct processor b were to compute $resolve_b(s)$ after round 2, then it would obtain v as the reduced value. Consider the tree of a correct processor p after reordering, in round 3. The children of sb in p's reordered tree are the values received by p from b. Because b is correct and b would have reduced s to v at the end of round 2, we have that $resolve_p(sb) = v$ at the end of round 3.

By the above discussion, we need only consider the case in which the source is detected in round 2. If the source is the only faulty processor, then, after fault masking, the round 2 trees of all correct processors are identical, so again a persistent value is immediately obtained. Thus we may assume $t_C > 1$. Consider the first round in which no new fault is detected. If a is any correct processor then before reordering all processors agree on $tree(sxa)$, for all processors x. Also, since the source has been detected, all processors have $tree(sxs) = 0$ for all x. Moreover, if a, b, and c are all correct, then all processors have $tree(sbc) = tree(sba)$, before reordering (because b tells the same thing to a as it tells to c, and a and c report these values correctly). Thus, if a and c are correct then, after reordering, the values stored at children of sa corresponding to correct processors must agree with the values stored at the children of sc corresponding to correct processors. It follows that if any of the at most $t_C - 1$ undetected faults had distinguished more than $t_C - 1$ such subtrees from the others (all with roots corresponding to correct processors), it would have been discovered by all processors applying Fault Discovery before reordering. Thus at least $n - t_C - (t_C - 1)^2$ correct processors a correspond to nodes labelled sa with identical children (after reordering) for all processors. By choice of t_C this number is a majority of n, so all correct processors agree on $resolve(s)$.

Finally, consider the earliest round in which the last fault is detected. After fault masking, but before reduction, all the children of s are common, and therefore, s is common. □

We are now ready to discuss shifting from one algorithm to another. Shifting from one algorithm to an algorithm of equal or greater resilience is not difficult. It is shifting down in resilience, even when the total number of faults is unchanged, that requires care. Clearly, shifting can safely occur only after some number of rounds have been executed, as otherwise it would be possible to reach agreement with, say, an $\frac{n-1}{4}$-resilient algorithm in the presence of $\frac{n-1}{3}$ faults. The specific number of rounds after which it is safe to shift between algorithms depends on the particular pair of algorithms involved. Since we are interested in producing a hybrid algorithm for the Main Theorem, we will first describe conditions under which we can shift from Algorithm A to Algorithm B, and then describe shifting from the hybrid to Algorithm C.

Proof Sketch for the Main Theorem:

Assume at most t_A faults. Our idea is to run Algorithm A until either we have a persistent value or k faults have been detected, for some $k < t_A - t_B$ yet to be determined. Note that a value is persistent in Algorithm A only if it preferred by all correct processors (because only the Weak Persistence Lemma holds if the reduction function is $resolve'$). Thus, if a value is persistent in Algorithm A it will persist after the shift to Algorithm B, in which the requirement for persistence is weaker. The intuition is that if after k rounds of Algorithm A a persistent value has not been obtained, then there are fewer than t_B undetected faults, so we should be able to shift into the end of round 1 of Algorithm B rather than into the end of round 1 of Algorithm A. On the other hand, if a persistent value has been obtained then by the Persistence Lemma we should again be able to shift into the end of round 1 of Algorithm B and the value obtained during the first resolution in Algorithm B will be the persistent value.

It remains to determine k so that, if no persistent value has been obtained during execution of Algorithm A, application of the Hidden Fault Lemma once we shift into Algorithm B works as in Algorithm B, even though $t_A > t_B$ processors may actually be faulty. Specifically, after the shift we want that if αb is any internal vertex and some correct processor does not discover b, then αb is common. That is, we want Corollary 1 to hold. This is achieved provided $n - 2t_A + k > \lfloor \frac{n-1}{2} \rfloor$. Thus we must take $k \geq \lfloor \frac{t_A}{2} \rfloor$. Let us write $\lfloor \frac{t_A}{2} \rfloor = x(d-3) + y$, where $y < d - 3$. Then we can run Algorithm A with parameter d for $3 + x(d-1) + y$ rounds and then shift to Algorithm B. We write $\lceil \frac{t_A}{2} \rceil = w(d-2) + z$, where $z < d - 2$. Then after the shift Algorithm B can be run with parameter d for $1 + w(d-1) + z$ rounds to produce a hybrid algorithm with resilience t_A that reaches Byzantine agreement in $4 + (x+w)(d-1) + y + z$ rounds.

It only remains to shift from this algorithm into Algorithm C after either a persistent value has been reached or a sufficiently large number m of faults has been detected. Interestingly, we can shift to the end of round 2 of Algcrithm C (until now we have always been shifting to the end of round 1). In consequence, the number of rounds remaining will be exactly the maximum number of undetected faults remaining. By reasoning similar to the explanation of our choice of k, we find that m must satisfy $n - t_A - (t_A - m)^2 > \frac{n}{2}$. Solving for m, we obtain

$m > t_A - \sqrt{\frac{t_A+1}{2}}$. Let m be the smallest such integer. Let us write $m - \lceil \frac{t_A}{2} \rceil = p(d-2) + q$, where $q < d - 2$. Then the final hybrid of A, B, and C can be run in $4 + (x+p)(d-1) + y + q + t_A - m$ rounds. Conversion to the bounds of the Main Theorem is straightforward. □

5. Concluding Remarks

We have constructed a set of algorithms for which it is possible to shift between any two. However, we do not have explicit necessary or sufficient conditions for an algorithm to be added to this set. We leave as an open question the characterization in general of when it is safe to shift from one algorithm to another with a given overall resilience.

We believe further study of hybrid algorithms may shed new light on the open question of a lower bound on the number of rounds required to reach Byzantine agreement with communication polynomial in the resilience.

References

[C1] Coan, B.A. "A Communication-Efficient Canonical Form for Fault-Tolerant Distributed Protocols," *Proceedings of the Fifth ACM Symposium on Principles of Distributed Computing*, 1986.

[C2] Coan, B.A., *PhD Thesis*, MIT (in preparation).

[DRS] Dolev, D., Reischuk, R., and Strong, H.R., "Early Stopping in Byzantine Agreement," IBM Research Report RJ5406 (55357), 1986.

[DS] Dolev, D. and Strong, H.R., "Polynomial Algorithms for MUltiple Processor Agreement," *Proc. Fourteenth Annual ACM Symposium on Theory of Computing* (1982), pp. 401-407.

[FL] Fischer, M. and Lynch, N., "A Lower Bound for the Time to Assure Interactive Consistency", *Information Processing Letters, 14*(4) (1982), pp. 183-186.

[PSL] Pease, M., Shostak, R., and Lamport, L., "Reaching Agreement in the Presence of Faults," *JACM 27*(2) (1980), pp. 228-234.