Shifting Gears: Changing Algorithms On the Fly to Expedite Byzantine Agreement

Amotz Bar-Noy * Danny Dolev † Cynthia Dwork ‡ H. Raymond Strong §

August 8, 1990

Abstract

We describe several new algorithms for Byzantine agreement. The first of these is a simplification of the original exponential-time Byzantine agreement algorithm due to Pease, Shostak, and Lamport, and is of comparable complexity to their algorithm. However, its proof is very intuitively appealing. A technique of *shifting* between algorithms for solving the Byzantine agreement problem is then studied. We present two families of algorithms obtained by applying a $shift$ operator to our first algorithm. These families obtain the same rounds to message length trade-off as do Coan's families but do not require the exponential local computation time (and space) of his algorithms. We also describe a modification of an $O(\sqrt{n})$-resilient algorithm for Byzantine agreement of Dolev, Reischuk, and Strong. Finally, we obtain a *hybrid* algorithm that dominates all our others, by beginning execution of an algorithm in one family and shifting first into an algorithm of the second family and finally shifting into an execution of the adaptation of the Dolev, Reischuk, and Strong algorithm.

*IBM T. J. Watson Research Center, P. O. Box 704, Yorktown Heights, NY 10598. This work was carried out while this author was a Ph.D. student of the Hebrew University, Jerusalem, Israel.

†IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120 and the Computer Science Department, Hebrew University, Jerusalem, Israel.

‡IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120.

§IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120.

1 Introduction

In designing fault tolerant 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 roughly $\frac{n-1}{3}$, $\frac{n-1}{4}$, and $(n/2)^{1/2}$ respectively, for which it is possible to shift, mid-execution, from one to another. Here, n is the total number of processors in the system, and the *resilience* of an algorithm is the maximum number t such that the algorithm runs correctly despite arbitrary (faulty) behavior of t processors. Using our three algorithms, we construct a hybrid algorithm tolerating $\frac{n-1}{3}$ faults, which begins by executing the relatively inefficient $\frac{n-1}{3}$-resilient algorithm, and then, after a predetermined number of rounds of communication, shifts to an execution of a more efficient algorithm of resilience $\frac{n-1}{4}$, and finally shifts into an execution of an optimally efficient $\sqrt{\frac{n}{2}}$-resilient algorithm. The first two algorithms are actually families of algorithms; the third algorithm is an adaptation of an algorithm of Dolev, Reischuk, and Strong [7]. Specifically, we prove the following theorem.

Theorem 1 (Main Theorem) *Let t satisfy $n \geq 3t+1$. For $2 < b \leq t$, there is a t-resilient algorithm for Byzantine agreement among n processors requiring*

$$t + 2\left\lfloor \frac{t/2-1}{b-2} \right\rfloor + \left\lfloor \frac{t/2 - \sqrt{(t+1)/2}+1}{b-1} \right\rfloor + 4$$

rounds of communication, using messages of $O(n^b)$ bits. Furthermore, the amount of local computation time at each processor is $O\left(n^{b+1}\left(\frac{t-1}{b-2}\right)\right)$.

Thus, for any choice of b in the given range, the algorithm requires at most

$$t + \frac{t}{b-2} + \frac{t}{2(b-1)} - O\left(\frac{\sqrt{t}}{b-1}\right) = t + O\left(\frac{t}{b}\right)$$

rounds of message exchange.

We assume the reader is familiar with the Byzantine agreement problem. An informal description of the model and statement of the problem appear in the next section. The rest of the paper is organized as follows. In Section 3, we describe a new algorithm for Byzantine agreement. We call this algorithm *Exponential Information Gathering with Recursive Majority Voting*, or simply, the *Exponential Algorithm*. The Exponential Algorithm is a simplification of the original Byzantine agreement algorithm due to Pease, Shostak, and Lamport [11] and is of comparable complexity to their algorithm. However, its proof is very intuitively appealing. The data structures defined in this new algorithm are the same as those used in our later algorithms and hybrids. 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. We prove that a simple modification of the Exponential Algorithm enjoys all three properties. In Section 4, we present two families

of linearly resilient algorithms interesting in their own right, as they achieve the rounds versus number of message bits trade-off exhibited by Coan's families [4, 5] but avoid the exponential local computation of his algorithms. These families are essentially obtained by applying a *shift* operator to the local states of the processors executing the Exponential Algorithm. We also describe a modification of an $O(\sqrt{n})$-resilient algorithm for Byzantine agreement of Dolev, Reischuk, and Strong [7]. Recent developments are mentioned in Section 5. Concluding remarks appear in Section 6.

2 Description of the Model and Statement of the Problem

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. The processor identifiers are common knowledge. 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 at the same round. We refer to this round as round 1.

In the Byzantine agreement problem, one distinguished processor, called the *source*, begins with a single initial (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 [5] to convert 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 initial value of the source.

An n-processor 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 $\lceil \frac{n}{3} \rceil$ faults [11]. Thus, since the problem is 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 [11]. 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." The algorithm requires $n \geq 3t + 1$.

In the Exponential Algorithm each processor maintains a dynamic tree of height at most t (each path from root to leaf contains at most $t + 1$ nodes), called the *Information Gathering Tree*. The nodes of the Information Gathering Tree are labelled with processor names as follows. The root is labelled s, for *source*. Let α be an internal node in the tree. For every processor name p not labelling an ancestor of α, α 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*. Henceforth, a *sequence* is an ordered list of at most $t + 1$ distinct processor names, beginning with s. We use Greek letters to denote (possibly empty) sequences and Roman letters to denote individual processors. We often refer to a node in the tree by specifying the sequence of labels encountered in traversing the path from the root to the node. Let α be such a sequence. The *length* of α, denoted $|\alpha|$, is the number of names in the sequence. Note that if α is an internal node then α has $n - |\alpha| \geq 2t + 1$ children. The *processor corresponding to node* α is the processor whose name labels node α, i.e., the last processor name in the sequence α. The Information Gathering Tree maintained by processor p is called $tree_p$.

The exponential Algorithm is split into two phases: Information Gathering, and Data Conversion.

Information Gathering:

In the first round of the Exponential Algorithm the source sends its initial value to all $n - 1$ other processors, decides on this value, and halts. We now describe the protocol for processors other than the source. For each $1 \leq h \leq t + 1$, the Information Gathering Tree at the end of round h is called the *round h tree*, and is of height $h - 1$.[1]

Let p be a correct processor. When p receives its message from the source, it stores the received value at the root of its tree. A special default value of $0 \in V$ is stored if the source failed to send a legitimate value in V. For $1 \leq h \leq t$, at round $h + 1$ processor p broadcasts the leaves of its round h tree. Upon receipt of these messages, each processor adds a new level to its tree, storing at node $s \ldots qr$ the value that r claims to have stored in node $s \ldots q$ in its own tree. Again, the default value 0 is used if an inappropriate message is received. Thus, intuitively, p stores in node $s \ldots qr$ the value that "r says q says ... the source said" (see Figure 1). We refer to this value as $tree_p(s \ldots qr)$, eliminating the subscript p when no confusion will arise. Information is gathered

[1] By convention, the height of an empty tree is -1.

Figure 1: The Information Gathering Tree

for $t+1$ rounds. This completes the description of the Information Gathering phase. The value stored in $tree_p(s)$ (i.e., at the root) is called the *preferred* value of p.

Data Conversion:

During this phase each correct processor p applies a recursive function to $tree_p$ to obtain a new preferred value. The value obtained by applying the conversion function to the subtree rooted at a node α is called the *converted value for* α. The specific data conversion function, *resolve*, used in the Exponential Algorithm is essentially a recursive majority vote and is defined as follows for all sequences α:

$$resolve(\alpha) = \begin{cases} tree(\alpha) & \text{if } \alpha \text{ is a leaf;} \\ v & \text{if } v \text{ is the majority value of } resolve \text{ applied to the children of } \alpha; \\ 0 & \text{if } \alpha \text{ is not a leaf and no majority exists.} \end{cases}$$

The value obtained by processor p in computing $resolve(\alpha)$ is denoted $resolve_p(\alpha)$. We occasionally drop the subscript p when no confusion will arise. Summarizing, we have:

The Exponential Algorithm:

1. Gather information for $t+1$ rounds;

2. Compute the converted value for s using the data conversion function *resolve*;

3. Decide on this converted value.

We now give a proof of correctness for this algorithm. After data conversion, a node α is said to be *common* if each correct processor computes the same converted value for α. Thus, the algorithm is correct if and only if both of the following conditions are satisfied:

- node s is common in every execution, *i.e.*, for all correct p and q, $resolve_p(s) = resolve_q(s)$;
- when the source s is correct, $resolve_p(s) = tree_p(s)$ for every correct processor p.

Recall that if p is correct, then $tree_p(s)$ is precisely the value received from the source during the first round. Thus, the second condition implies that if the source is correct then all correct processors, including the source, decide on the source's initial value.

The following lemma is more general than is necessary to prove correct the Exponential Algorithm so that we can use it later, in the proofs of correctness of our families of algorithms.

Lemma 1 (Correctness Lemma) *For any $1 \leq h \leq t+1$, consider the Information Gathering Tree after h rounds of Information Gathering. Let $\alpha = \beta q$ be a sequence of length at most h in which $|\beta| \geq 0$ and q is correct. If data conversion is applied to the h-round tree, then there is a value v such that α is common with converted value v and, for every correct processor p, $tree_p(\alpha) = v$.*

Proof: Throughout the proof, let p be a correct processor. Note that since q is correct, $tree_p(\alpha) = tree_q(\beta)$. If β is empty then $q = s$. In this case, we interpret $tree_s(\beta)$ to be the source's initial value.

The lemma is proved by reverse induction on the length of α. If $|\alpha| = h$ then, since α is a leaf, $resolve_p(\alpha) = tree_p(\alpha)$ for all correct processors p. Thus, α is common.

Assume the lemma for sequences of length k, where $1 < k \leq h$. Let α be a sequence of length $k-1$. Let $r \notin \alpha$ be a correct processor. By induction, $resolve_p(\alpha r) = tree_p(\alpha r)$. Moreover, since p, q, and r are all correct,

$$tree_p(\alpha r) = tree_r(\alpha) = tree_q(\beta) = tree_p(\alpha).$$

Thus, all but t of the children of α in $tree_p$ have common converted value equal to $tree_p(\alpha)$. However, since α is internal it has at least $2t+1$ children, hence $resolve_p(\alpha) = tree_p(\alpha)$. This completes the proof. ∎

From the Correctness Lemma (with $h = t+1$ and $\alpha = s$) we immediately have

Claim: *If the source is correct then after data conversion s is common and $resolve_p(s) = tree_p(s)$ for all correct processors p.* ∎

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 node, independent of the correctness of the source. When every root-to-leaf path contains a common node we say the Information Gathering Tree has a *common frontier.* It remains to show that the existence of a common frontier guarantees agreement. This is immediate from the following lemma.

Lemma 2 (Frontier Lemma) *If there is a common frontier, then s is common.*

Proof: To prove the Frontier Lemma, we prove the following, more general, claim:

Claim: *Let α be a node. 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).*

To prove the Claim, suppose it failed in some execution of the algorithm and suppose α were a counterexample of maximal length: thus α would not be common but the subtree rooted at α would have a common frontier. If the subtree rooted at a leaf has a common frontier, then the leaf is common. Hence, α cannot be a leaf. If a subtree has a common frontier, either its root is common or the subtree rooted at each child of its root has a common frontier. Hence, by the length maximality of α, each of its children is common. But then every correct processor computes the same value for $resolve(\alpha)$, and α is common, contradicting the assumption that α is not common. ∎

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

Proposition 1 *The Exponential Algorithm reaches Byzantine agreement in $t+1$ rounds provided $n \geq 3t+1$.* ∎

We have shown that this simple variant of the original algorithm of Pease, Shostak, and Lamport reaches Byzantine agreement in the optimal number of rounds [9]. Despite the simplicity of the algorithm, the message size and the amount of local computation required grow exponentially in t. More specifically, for any $1 \leq h \leq t+1$, the h-round Information Gathering Tree has $O(n^{h-1})$ leaves, yielding messages of size $O(n^{h-1})$ in round $h+1$. Later, we will show how, for any chosen bound n^k on message length, $2 \leq k \leq t$, we can modify the Exponential Algorithm to require message length, local space, and local computation at most $O(n^k)$. The main idea is to run the Exponential Algorithm for k rounds, and then to apply a *shift* operator involving data conversion to reduce message size and tree size back to $O(1)$. This is repeated roughly $\frac{t}{k-O(1)}$ times. Thus, there is a penalty paid in time, but message length, local storage, and local computation time are all bounded by n^k. However, before we can apply shifting, we must modify the algorithm and prove that the modified algorithm exhibits three important properties, *persistence, fault detection,*

and fault masking, which we now describe.

Recall that $tree_p(s)$ is called the *preferred value* of processor p. In the Exponential Algorithm, a processor's preferred value changes at most once. However, this will not be the case in our later algorithms. *Persistence* says that if sufficiently many correct processors prefer v, then this situation persists, and moreover, the eventual decision value will be v. In the case of the Exponential Algorithm this says merely that if sufficiently many correct processors prefer a value v then v will be the decision value.

During the execution of our algorithms, it may sometimes be possible for a correct processor p to deduce that some other processor q is faulty. In the modified Exponential Algorithm and in all our future algorithms, each processor p maintains a list L_p of processors known to be faulty. Exact details of when processors are added to these lists are discussed below. We say a faulty processor is *globally detected* if all correct processors have discovered it to be faulty. (We distinguish between *discovery*, which describes the action of a single processor, and *global detection*, in which all correct processors have discovered the same faulty processor. These discoveries need not take place simultaneously.) A fault that has not been globally detected is said to be *undetected.*

Processor p ignores messages from processors it knows to be faulty (*i.e.*, processors in the list L_p). Thus, the actions of globally detected processors are essentially *masked.* Later we will see that the global 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, despite the fact that more faults may occur later.

Lemma 3 (Persistence Lemma) *For any h such that $1 \leq h \leq t+1$, if all correct processors have the same preferred value v, then after conversion is applied to the h-round tree, s is common and has converted value v.*

Proof: By the Correctness Lemma, children of s corresponding to correct processors with preferred value v will be common with converted value v after conversion using *resolve*. Thus, all correct processors will compute $resolve(s) = v$. ∎

The value v described in the Persistence Lemma is called a *persistent* value. We remark that, due to the specific choice of conversion function *resolve*, the Persistence Lemma holds even if the correct processors sharing the same preferred value simply constitute a majority of all processors. We refer to this fact as the **Strong Persistence Lemma**. The Strong Persistence Lemma will be used in proving correct our hybrid algorithms.

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 note that if a processor misbehaves in a way not discussed in the Fault Discovery Rule, then

p can "observe" this fact; our algorithms simply do not take advantage of this extra information.

Let p be any processor. For every internal node β in $tree_p$, 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 r not already in L_p is added to L_p if for some internal node αr in $tree_p$

- there is no majority value for αr or
- a majority value for αr exists but values other than the majority value are stored at more than $t - |L_p|$ children of αr corresponding to processors $q \notin L_p$.

Processor r is added to L_p in the round in which the conditions of the Fault Discovery Rule are first satisfied, specifically, at the end of round $|\alpha r| + 1$ (before round $|\alpha r| + 2$ of Information Gathering).

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. Since the L_p are initially empty, no correct processor p ever puts the name of a correct processor into L_p.

Lemma 4 (Hidden Fault Lemma) *Let p be a correct processor and let αr be any internal node in $tree_p$. Let $k = |\alpha r|$. If all the processors in αr are faulty, but $r \notin L_p$ after round $k + 1$ (i.e., after p stores values at the children of αr), then there exists a majority value for αr, and the set of children of αr at which its majority value is stored contains at least $n - 2t + |L_p|$ nodes corresponding to correct processors.*

Proof: By assumption, correct processor p does not put r into L_p after it stores values at the children of αr. Hence p has a majority value v for αr, and v is stored at all children of αr not corresponding to processors in L_p, except for at most $t - |L_p|$ processors. Since all processors in αr are faulty, there are at least $n - t$ correct processors corresponding to children of αr. Since no correct processor is in L_p, there are at most $t - |L_p|$ children αrq of αr such that q is correct and $tree_p(\alpha rq) \neq v$. Thus the number of children of αr with stored value v that correspond to correct processors is at least $n - t - (t - |L_p|) = n - 2t + |L_p|$. ∎

Let p be a correct processor and q a faulty processor. An 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. We exploit this property in the following Fault Masking Rule.

Fault Masking Rule: If q is added to L in round k, then any messages from q in round k and any subsequent round are replaced by messages in which each value is the default 0.

It is worth commenting on the relative ordering of Fault Discovery and Fault Masking. When the round k messages are received, messages from processors added to L_p *before* round k are masked. Fault Discovery is performed on the resulting round k tree. Newly discovered faulty processors are added to L_p. Finally, the round k messages of these newly discovered processors are also masked. Note that although p masks the round k messages of q, it does not change any portion of its round $k-1$ tree. (Thus, it only changes the portion of its tree that it has not yet sent to other processors.)

Intuitively, once a processor p discovers that q is faulty, it "acts as if" q sends only zeros. Under the Fault Masking Rule, once a processor has been globally detected, it is essentially forced to send the same values (zeros) to all correct processors. We assume for the rest of this paper that the fault discovery and fault masking rules are applied in each round of Information Gathering. Since s never sends after round 1, Fault Masking is never used to fill in the root, $tree(s)$ (although if s fails to send in round 1 the default value is assumed).

4 Shifting

Roughly speaking, *shifting* involves changing, mid-execution, from running one algorithm to running a different algorithm, or to running the same algorithm but from a different point or round number. Thus,

Definition 1 *A* shifting *is an operator $shift_{k \to j}$ that uses some conversion process to change a subset of the data structures appropriate to the end of round k of one algorithm into those appropriate to the end of round j of another (possibly the same) algorithm.*

To specify a shift operation $shift_{k \to j}$ we need only specify the original algorithm, the conversion process, and the target algorithm. Each of our families of algorithms is obtained by repeated shifting into the same algorithm. The hybrid algorithms are obtained by shifting from a member of one family to a member of the second family.

Henceforth, by "Exponential Algorithm" we mean the Exponential Algorithm modified for fault discovery and fault masking. All our algorithms are derived from the Exponential Algorithm by applying the shifting technique. In each case there is a principal data structure, the Information Gathering Tree, and some auxiliary data structures (*e.g.* the lists L_p of faulty processors discovered by p), all of which tend to get larger as execution progresses. The strategy is to run the Exponential Algorithm for some specified number k of rounds (k is usually a parameter of the algorithm), building the principal and auxiliary data structures. The round k principal data structure (generally large) is then converted to a round j data structure, where $j < k$, so that during conversion the structure shrinks. The auxiliary data structures generally remain unchanged. After conversion, we execute the Exponential Algorithm from round $j+1$ but using the

unconverted auxiliary data structures, possibly shifting again when the principal data structure again is as in round k.

Our algorithms are proved correct by showing that in any *phase* (*i.e.*, between successive shifts), either a persistent value is obtained or some number of new faults are globally detected and thereafter masked. This part makes use of the Hidden Fault Lemma and some of its corollaries. Intuitively, the algorithm need only be run for sufficiently many phases for all t faults to be detected, since globally detected faults cannot prevent the emergence of a persistent value. Persistence Lemmas are used to show that, once obtained, a persistent value remains persistent.

Our two families of algorithms require $n \geq 3t + 1$ and $n \geq 4t + 1$, respectively, to tolerate t faults. Since we wish to keep n fixed, we define

$$t_A = \left\lfloor \frac{n-1}{3} \right\rfloor \quad ; \quad t_B = \left\lfloor \frac{n-1}{4} \right\rfloor \quad ; \quad t_C = \left\lfloor \sqrt{\frac{n}{2}} \right\rfloor .$$

Theorem 2 *For $2 < b \leq t$, Byzantine agreement can be achieved in the presence of $t \leq t_A$ faults in $t + 2 + 2\left\lfloor \frac{t-1}{b-2} \right\rfloor$ rounds of communication, using messages of $O(n^b)$ bits. Furthermore, the amount of local computation time at each processor is $O\left(n^{b+1}\left(\frac{t-1}{b-2}\right)\right)$.*

Theorem 2 is proved by constructing a family of algorithms, indexed by b, by repeatedly applying $shift_{b+1 \to 1}$ to the Exponential Algorithm. In this family, if a persistent value is not obtained in a given block of b rounds, then the number of new faults globally detected is at least $b - 2$ (this is why the time bound is infinite when $b = 2$, since in this case there is no guarantee of progress). In the family of algorithms constructed for the next theorem, the number of faults globally detected in a block of b rounds is at least $b - 1$ (the theorem therefore only holds for $b > 1$), but the overall resilience is lower.

Theorem 3 *For $1 < b \leq t$, Byzantine agreement can be achieved in the presence of $t \leq t_B$ faults in $t+1+\left\lfloor \frac{t-1}{b-1} \right\rfloor$ rounds of communication, using messages of $O(n^b)$ bits. Furthermore, the amount of local computation time at each processor is $O\left(n^{b+1}\left(\frac{t-1}{b-1}\right)\right)$.*

In Subsection 4.3, we will describe a third algorithm, based largely on the algorithm of Dolev, Reischuk, and Strong [7], but recast in terms of the techniques developed in this paper. Blocks of this third algorithm consist of single rounds, and at each round if no new fault is globally detected, then something that functions like a persistent value is obtained. However, the resilience is very low.

Theorem 4 *Byzantine agreement can be achieved in the presence of $2 < t \leq t_C$ faults in $t + 1$ rounds of communication, using messages of $O(n)$ bits. Furthermore, the amount of local computation time at each processor is at most $O(n^{2.5})$.*

We first describe Algorithm B, the family of algorithms of Theorem 3, since it is most easily obtained from the Exponential Algorithm. Algorithm A, the family of Theorem 2, is similar, but its analysis is more subtle.

Algorithm B(b):

Execute the Exponential Algorithm for 1 round;

DO $\left\lfloor \frac{t-1}{b-1} \right\rfloor$ times $\rightarrow$

Execute rounds 2 through $b+1$ of the Exponential Algorithm;

{ **comment** $shift_{b+1 \rightarrow 1}$ }

$tree(s) = resolve(s)$;

OD

IF $b-1$ does not divide $t-1$ then

Execute rounds 2 through $1 + t - (b-1)\left\lfloor \frac{t-1}{b-1} \right\rfloor$ of the Exponential Algorithm; FI

Decide $resolve(s)$;

Figure 2: Algorithm B with Parameter $b < t$ (t and n Fixed)

4.1 Algorithm B

Let $t \leq t_B = \left\lfloor \frac{n-1}{4} \right\rfloor$. Algorithm B has parameter b, which is the maximum number of rounds (after round 1) in any block. We require $1 < b \leq t$. If $b = t$, then Algorithm B is just the Exponential Algorithm. Henceforth, we assume $b < t$. In this case, Algorithm B is simply the repeated application of $shift_{b+1 \rightarrow 1}$ to the Exponential Algorithm until the total number of rounds of communication completed is $t+1+\left\lfloor \frac{t-1}{b-1} \right\rfloor$ (one fewer round is needed when $(b-1)|(t-1)$). The algorithm appears in Figure 2. Data conversion is accomplished by applying the *resolve* function of the previous section to obtain a converted value for s.

After the initial round, Algorithm B repeats blocks of b rounds. We could use $\left\lceil \frac{t}{b-1} \right\rceil$ such blocks; but we optimize the number of rounds in the last block so that Algorithm B uses an initial round, $\left\lfloor \frac{t-1}{b-1} \right\rfloor$ blocks of b rounds, and, if $b-1$ does not divide $t-1$, one final block with $t-(b-1)\left\lfloor \frac{t-1}{b-1} \right\rfloor$ rounds. The total number of rounds is therefore in the worst case

$$1 + b\left\lfloor \frac{t-1}{b-1} \right\rfloor + t - (b-1)\left\lfloor \frac{t-1}{b-1} \right\rfloor = 1 + t + \left\lfloor \frac{t-1}{b-1} \right\rfloor .$$

Claim 1 *The proofs of the Correctness, Frontier, Persistence, and Hidden Fault Lemmas (Lemmas 1-4 respectively) hold verbatim for executions of Algorithm B.* ∎

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

Corollary 1 *Let αr be an internal node in an Information Gathering Tree of Algorithm B and let q be a correct processor. If all processors in αr are faulty, and if, when the children of αr are stored in $tree_q$, q does not discover r by the Fault Discovery Rule, then αr is common.*

Proof: If correct processor q does not discover r to be faulty, then, by the Hidden Fault Lemma, there is a majority value for αr in $tree_q$ and it is stored in at least

$$n - 2t_B + |L_q| \geq n - 2t_B > (n-1)/2$$

children of αr corresponding to correct processors, (recall, $n > 4t_B$). Thus, by the Correctness Lemma and the definition of *resolve*, αr must be common. ∎

Later, in constructing the hybrid algorithms, we will want Corollary 1 to hold after shifting into Algorithm B. The hybrid algorithm must tolerate up to $t_A > t_B$ faults, but the proof of the Corollary relies on the fact that $t \leq t_B$. Note, however, that if $|L_q|$ is sufficiently large, then $n - 2t_A + |L_q| > (n-1)/2$. The trick will be to shift into Algorithm B only after sufficiently many faults have been globally detected (and hence, the sets L_q are large), or a persistent value has already been obtained. In the latter case the Strong Persistence Lemma will ensure that the persistent value remains so after the shift.

Proposition 2 *Algorithm B achieves Byzantine agreement with the bounds on resiliency, message length, local computation time, and number of rounds of communication stated in Theorem 3.*

Proof: The bound on number of rounds is proved in the discussion immediately preceding Claim 1. We now argue correctness. Consider the $(b+1)$-round Information Gathering Tree of any block. If there is a common frontier, then, by the Frontier Lemma, after conversion s is common and its converted value is persistent. By the Persistence Lemma, a persistent value remains so even in subsequent blocks. In particular, if the source is correct then at the end of round 1 all correct processors prefer the same value, to wit, the value that the source broadcasts in round 1, so this value is persistent.

If, in the $(b+1)$-round Information Gathering Tree of some block there is no common frontier, then by definition there is a path ρ from root to leaf containing no common node. By the Correctness Lemma, all processors corresponding to nodes in ρ are faulty. By Corollary 1, the faults corresponding to internal nodes of ρ are all globally detected.

Except for the source, which is repeatedly detected, once a processor is globally detected, nodes corresponding to it in the Information Gathering Trees of subsequent blocks will be common. This is because faults other than the source are masked according to the Fault Masking Rule.

Note that after the initial round, Algorithm B repeatedly runs b rounds of Information Gathering followed by conversion using *resolve*. Each block of b rounds that produces trees without a common frontier results in the global detection of at least $b-1$ new faults besides the source.

There are two cases, according to whether or not $b-1$ divides $t-1$. If $b-1$ does not divide $t-1$ then let us write $t-1 = (b-1)x + y$, where x and y are nonnegative integers and $y < b-1$. (Note that $x = \left\lfloor \frac{t-1}{b-1} \right\rfloor$ is the number of iterations of the main loop.) After the first $1+bx$ rounds of Algorithm B, if no persistent value has been obtained, then $1+(b-1)x$ faults have been globally

detected. Rewriting the last equation we have $t - y = 1 + (b-1)x$. Thus, after $1 + bx$ rounds, if no persistent value has been obtained then $t - y$ faults have been globally detected. Therefore, after an additional $y + 1$ rounds followed by conversion using *resolve*, s must be common by the Persistence and Frontier Lemmas. Since in this case Algorithm B uses exactly $2 + bx + y$ rounds, algorithm B reaches Byzantine agreement in the presence of up to t faults.

Suppose $b - 1$ divides $t - 1$, and consider the final iteration of the loop in an execution of Algorithm B. If on beginning this iteration the source is already common, then by the Persistence Lemma s will be common at the end of the iteration. If the source is not common, then at most $b-1$ faults other than the source are not globally detected before the final iteration begins. Thus, including the source (which is undetected if $b = t$ and will be redected otherwise) there can be at most b undetected faults just before this last iteration begins. But the tree constructed in the last iteration is an $h + 1$ round tree, and thus will have a common frontier, whence by the Frontier Lemma s will be common after data conversion at the end of this last iteration.

To prove the bounds on message length and local computation time, note that messages sent at the beginning of the last round of each block of b rounds (the largest messages) carry information about trees with $O(n^b)$ leaves. Thus messages for Algorithm B carry at most $O(n^b)$ bits. For each block of b rounds, the algorithm requires $O(n^{b+1})$ local computation, so the entire execution requires $O\left(n^{b+1}\left(\frac{t-1}{b-1}\right)\right)$ local computation at each processor. ∎

4.2 Algorithm A

Let $t \leq t_A = \left\lfloor \frac{n-1}{3} \right\rfloor$. Algorithm A has parameter b, which is the maximum number of rounds of Information Gathering (after round 1) in any block. We require $2 < b \leq t$. If $b = t$, then Algorithm A is exactly the Exponential Algorithm with a different data conversion function, *resolve'*, described below. Henceforth, we assume $b < t$. In this case, Algorithm A is the repeated application of $shift_{b+1 \to 1}$ to the Exponential Algorithm, using data conversion function *resolve'*, defined as follows:

$$resolve'(\alpha) = \begin{cases} tree(\alpha) & \text{if } \alpha \text{ is a leaf;} \\ v & \text{if } v \text{ is the unique value in V occurring at least } t+1 \text{ times in applying} \\ & resolve' \text{ to the children of } \alpha; \\ \perp & \text{if } \alpha \text{ is not a leaf and no such unique value exists.} \end{cases}$$

The definition of *resolve'* introduces a new value, $\perp \notin V$. Although used during the conversion process, $\perp$ is never used in the Information Gathering phase itself. If, at the end of some conversion, $resolve'_p(s) = \perp$ for some correct processor p, then p uses the default value (0) as its new preferred value.

Algorithm A uses an initial round, $\left\lfloor \frac{t-1}{b-2} \right\rfloor$ blocks of b rounds, and, if $b-2$ does not divide $t-1$,

one final block with $t+1-(b-2)\left\lfloor\frac{t-1}{b-2}\right\rfloor$ rounds, for a total of $t+2+2\left\lfloor\frac{t-1}{b-2}\right\rfloor$ rounds.

Claim 2 *The Correctness, Frontier, Persistence, and Hidden Fault Lemmas (Lemmas 1-4 respectively) hold for executions of Algorithm A.*

Proof: The proofs of the Correctness, Frontier, and Persistence Lemmas given above hold for Algorithm A with "*resolve*" replaced by "*resolve'*". The proof of the Hidden Fault Lemma requires no changes. ∎

Remarks:

1. Using the Correctness and Frontier Lemmas, it is easy to show that the Exponential Algorithm reaches Byzantine agreement in t+1 rounds with *resolve'* instead of *resolve* as conversion function.

2. Since $\perp$ is never used during Information Gathering, if p and q are correct then by the Correctness Lemma, $resolve'_p(\alpha q) \neq \perp$. In other words, the converted value of a node corresponding to a correct processor is never $\perp$.

Recall that Corollary 1 to the Hidden Fault Lemma said that if an internal node is not common then its corresponding processor is globally detected. 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 hold for all nodes of height at least 2 in the presence of up to t_A faults. Specifically, we have the following corollary to the Hidden Fault Lemma.

Corollary 2 *Let $2 < h \leq t+1$. Let αr be an internal node in an h-round Information Gathering Tree of Algorithm A, and let all processors in αr be faulty. If, when data conversion is applied to the h-round tree, two correct processors p and q obtain different converted values for αr, neither of which is $\perp$, then $r \in L_p \cap L_q$ at the end of round $|\alpha r|+1$.*

Proof: Assume correct processors p and q obtain different converted values for αr, neither of which is $\perp$. Suppose, for the sake of contradiction, that $r \notin L_p$ at the end of round $|\alpha r|+1$. By the Hidden Fault Lemma, there is a value v such that for at least $n-2t+|L_p|$ correct processors w, $tree_p(\alpha r w) = v$. By the Correctness Lemma, these children are common and all correct processors (including p and q) have converted value v for each of them. Since $t \leq t_A$, there are at least t+1 such children w. Thus $resolve'_q(\alpha r) \in \{v, \perp\}$, contradicting the assumption that $resolve'_q(\alpha r) \notin \{resolve'_p(\alpha r), \perp\}$. ∎

In order to obtain the result of Corollary 1 for internal nodes of height at least 2 (in the presence of t_A faults), we increase the power of the Fault Discovery Rule by applying it during the conversion process.

Fault Discovery Rule During Conversion: Let p be a correct processor. During conversion, p adds $r \notin L_p$ to L_p if, for some internal node αr corresponding to r,

- there is no majority value among the converted values for the children of αr or
- a majority value v exists, but for more than $t - |L_p|$ processors $q \notin L_p$, $resolve'_p(\alpha rq) \neq v$.

Corollary 3 *Let αr be an internal node, but not the parent of a leaf, in an Information Gathering Tree of Algorithm A. If all processors in αr are faulty, and if some correct processor q does not discover r either by the Fault Discovery Rule or the Fault Discovery Rule During Conversion, then αr is common.*

Proof: Since q does not discover r during Conversion, there is a value v such that $resolve'_q(\alpha r) = v$ and q has at most $t - |L_q|$ children of αr corresponding to processors not in L_q that have converted values other than v. Thus, for at most t children w of αr we have $resolve'_q(\alpha rw) \neq v$. Let $z \notin L_q$ be such that $resolve'_q(\alpha rz) = v$, and let p be any correct processor different from q. If z is faulty, then by Corollary 2 $resolve'_p(\alpha rz) \in \{v, \perp\}$, while if z is not faulty then by the Correctness Lemma $resolve'_p(\alpha rz) = v$. Thus, for at most t children w of αr we have $resolve'_p(\alpha rw) \notin \{v, \perp\}$. Thus, p sees at most t support for any non-$\perp$ value $u \neq v$. Moreover, by the Hidden Fault and Correctness Lemmas, all correct processors, including p, see at least $t + 1$ support for v, so $resolve'_p(\alpha r) = v$. ∎

Proposition 3 *Algorithm A achieves Byzantine agreement with the bounds on resiliency, message length, local computation time, and number of rounds of communication stated in Theorem 2.*

Proof: If the source is correct then after Round 1 all correct processors prefer the same value, so by the Persistence Lemma, s will be common with converted value v after conversion.

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 $b + 1$-round Information Gathering Tree contains a path ρ containing no common nodes. Once again the Correctness Lemma implies that all processors corresponding to nodes on ρ are faulty. By Corollary 3, for every internal node αr of ρ other than the parent of the leaf, processor r is globally detected. As before, except for the source, which is repeatedly detected, once a processor is globally detected, nodes corresponding to it in the Information Gathering Trees of subsequent blocks will be common with converted value 0, the default value used in Fault Masking.

In analyzing the running time of Algorithm A there are two cases, according to whether or not $b - 2$ divides $t - 1$. Since the analysis is so close to that of Algorithm B, we discuss only the case in which $b - 2$ does not divide $t - 1$. After the initial round, Algorithm A repeatedly runs b rounds of Information Gathering followed by conversion using $resolve'$. Each block of b rounds that produces trees without a common frontier results in the global detection of at least $b - 2$

new faults besides the source. For $t \leq t_A$, let us write $t - 1 = (b-2)x + y$, where x and y are nonnegative integers and $y < b - 2$. (Note that $x = \left\lfloor \frac{t-1}{b-2} \right\rfloor$.) After $1 + bx$ rounds of Algorithm A, either s is common or there are at most y faults not globally detected. After an additional $y + 2$ rounds followed by conversion by *resolve'*, s must be common by the Persistence and Frontier Lemmas. Since Algorithm A uses exactly $3 + bx + y$ rounds, Algorithm A reaches Byzantine agreement in the presence of up to t faults.

To prove Theorem 2, let $2 < b \leq t \leq t_A$. Then algorithm A with parameter b achieves Byzantine agreement in at most

$$1 + b\left\lfloor \frac{t-1}{b-2} \right\rfloor + t + 1 - (b-2)\left\lfloor \frac{t-1}{b-2} \right\rfloor = t + 2 + 2\left\lfloor \frac{t-1}{b-2} \right\rfloor$$

rounds of communication. Messages sent at the beginning of round $b+1$ carry information about trees with $O(n^b)$ leaves. Thus messages for Algorithm A carry at most $O(n^b)$ bits. For each block of b rounds, the algorithm requires $O(n^{b+1})$ local computation, so the entire execution requires $O\left(n^{b+1}\left(\frac{t-1}{b-2}\right)\right)$ local computation time at each processor. ∎

4.3 Algorithm C

In this subsection we modify a theorem of Dolev, Reischuk, and Strong, and recast their (modified) theorem in terms of shifting. Our proof of the modified theorem follows the lines of the original proof [7].

Let $t_C = \left\lfloor \sqrt{n/2} \right\rfloor$. We will presently describe Algorithm C which, if $t \leq t_C$, achieves the bounds of Theorem 4. Algorithm C will use an Information Gathering Tree in which each internal node has exactly n children, one labelled with each processor name. Thus, in Algorithm C vertices in the tree are sequences of processor names *with repetitions*, beginning always with s. Faults are discovered using the Fault Discovery Rule during Information Gathering, which does not change to reflect this new tree structure.

Consider first the following 3-round algorithm.

- Run Information Gathering for three rounds building a tree *with repetitions*, applying the Fault Discovery Rule during rounds 2 and 3, and masking faults as soon as they are discovered (*i.e.*, if faulty processor p is discovered in round i, then mask its round i messages).
- Reorder the leaves of the resulting 3 level tree by swapping the values stored in $tree(spq)$ and $tree(sqp)$, for all $q \neq p$.

After the reordering, the leaves in the subtree rooted at sq contain the values received from q in round 3, unless q was discovered to be faulty, in which case these leaves contain the default value 0 given by fault masking. Algorithm C is the repeated application of $shift_{3 \to 2}$ to this

3-round algorithm until the entire execution has run for exactly $t+1$ rounds. The conversion is achieved by setting $tree(sq) = resolve(sq)$ for all processors q. This results in a two-level tree. A final application of $shift_{2\to 1}$, this time by setting $tree(s) = resolve(s)$, yields the decision value. Note that the Information Gathering Tree never has more than three levels. The children of the root are referred to as *intermediate vertices.*

Claim 3 *The Frontier and Hidden Fault Lemmas hold for Algorithm C when applied to the 3-round Information Gathering Tree before reordering. The Frontier Lemma also holds after reordering.*

Proof: The proof of the Frontier Lemma holds verbatim for Algorithm C, before and after reordering. (Although the Frontier Lemma holds before and after reordering, we will only apply it to the Information Gathering Tree after reordering, since we only apply data conversion after reordering.)

The proof of the Hidden Fault Lemma requires only minor modifications. The modifications are needed because in Algorithm C the Information Gathering Tree is constructed with repetitions; the proof is even easier in this case. (In fact, for Algorithm C the Hidden Fault Lemma holds even without the condition that all processors in αr be faulty.) ∎

The Correctness Lemma does not hold for Algorithm C, principally because of the reordering. However, we do have a weaker version.

Lemma 5 (Correctness for Intermediate Vertices) *For $k \geq 3$, let p be a correct processor and let $\alpha = sp$ be an intermediate node. Then at the end of round k $resolve_q(\alpha) = resolve_r(\alpha)$ for all pairs of correct processors q, r. That is, all correct processors compute the same converted value for α at the end of round k.*

Proof: Consider the round k Information Gathering Tree of processor q after reordering. The children of α are the values received from p in the current round. Since p is correct it sent the same values to r, and since r is correct these values are stored correctly, after reordering, in $tree_r$. Thus, before conversion, for all w (correct or faulty), $tree_q(\alpha w) = tree_r(\alpha w)$ and the lemma follows. ∎

Henceforth, by "preferred value at the end of round k" we mean the value that would be obtained by computing $resolve(s)$ just after reordering at round k, even though the algorithm does not call for this computation. Accordingly, we say "s is common at the end of round k" if all correct processors hold the same preferred value for s at the end of round k. Note that after applying $shift_{3\to 2}$ processors do not broadcast their preferred values, but only the values stored at the intermediate vertices. Thus, the Persistence Lemma as stated above does not hold. However, we have an analogue.

Lemma 6 *Let p be correct. Let k be the round number of any round of the algorithm. Consider*

the following propositions:

1. *at the end of round k, there exists a value v such that for strictly more than $n/2$ correct processors r, we have $tree_p(sr) = v$;*
2. *at the end of round k, s is common with value v;*
3. *if $k+1$ is a round of the algorithm, then at the end of round $k+1$ there exists a value v such that $tree_p(sq) = v$ for all correct processors q;*
4. *if $k+1$ is a round of the algorithm, then at the end of round $k+1$ there exists a value v such that for strictly more than $n/2$ correct processors r we have $tree_p(sr) = v$.*

The following implications hold: (1) implies (2), (2) implies (3), and (3) implies (4).

Proof: Note that "the end of round k" means "after $tree(s)$ is obtained" if $k = 1$, "after information gathering" if $k = 2$, and "after conversion" if $k > 2$.

Observe that (4) follows immediately from (3). Note that if k is not the last round then (4) is simply (1) for round $k+1$. We now prove that (1) implies (2) and that (2) implies (3). Once this is done, we will have that if (1) holds for round k and $k+1$ is also a round of the algorithm, then (1) holds also for round $k+1$.

Assume (1). By assumption there exist v and more than $n/2$ correct processors r such that $tree_p(sr) = v$ at the end of round k. (Note that we must have $k > 1$.) Thus, it is immediate that v is the preferred value of p at the end of round k. We now show that (1) holds with p replaced by any correct processor q, from which will follow (2).

Fix one of the correct processors r given in the statement of the Lemma. Let q be any correct processor. There are two cases to consider, according to whether or not $k = 2$. If $k = 2$, then $tree_p(sr)$ is the value received from r in round k. Since r is correct, it sent to q the same value, so at the end of round k, $tree_q(sr) = v$. If $k \neq 2$, then the value stored in $tree_p(sr)$ was obtained by computing $resolve_p(sr)$ at the end of round k, and by Lemma 5 (with $\alpha = sr$), $resolve_q(sr) = resolve_p(sr)$. Applying the above argument to each of the more than $n/2$ correct processors r such that $tree_p(sr) = v$, we see that at the end of round k indeed (1) is satisfied for q. Thus, v is the common preferred value for s at the end of round k, proving (2).

Now, assume (2) and assume that $k+1$ is a round of the algorithm. If $k = 1$, then every correct processor sends v in round 2 and (3) holds at the end of round 2. Assume $k > 1$. For all correct q, after reordering at the end of round $k+1$, the leaves of the subtree of $tree_p$ rooted at sq contain precisely the values stored at the intermediate nodes of $tree_q$ at the end of round k. Since v is the common preferred value for s at the end of round k, a majority of these values are v, whence $resolve_p(sq) = v$, proving (3). ∎

The value v described in Lemma 6 is called a "persistent" value. Note that Lemma 6 actually yields an analogue to the Strong Persistence Lemma, since its conditions merely require that the value v be stored at more than $n/2$ intermediate vertices corresponding to correct processors (rather than $n - t_C$ such nodes). This will be important when we shift into Algorithm C in the hybrid algorithm.

Proposition 4 *Algorithm C achieves Byzantine agreement with the bounds on resiliency, message length, and number of rounds of communication stated in Theorem 4.*

Proof: That Algorithm C has the stated running time, message length, and local computation bounds is clear by inspection.

We will show that, after the first round of Algorithm C, if there is a round in which no new fault is globally detected during Information Gathering, then a persistent value is obtained. We will also show that a persistent value is obtained by the end of the earliest round in which all t faults are globally detected. The second claim is used to show that $t + 1$ rounds suffice even if only one fault is discovered in each of rounds 2 through $t + 1$.

In round 2, if the source is not globally detected, then some processor p does not discover the source to be faulty. By the Hidden Fault Lemma, at least $n - 2t + |L_p|$ correct processors q had $tree_q(s) = v$ after round 1, for some value v. For each of these $n - 2t + |L_p|$ correct processors q, $tree_r(sq) = v$ at the end of round 2, for every correct processor r. The conditions on t ensure that $n - 2t + |L_p|$ is a majority of the n processors, so condition (1) of Lemma 6 is satisfied. In this case (in particular, if s is correct and has initial value v), then from round 2 on, s is common with value v.

We now turn to the case in which the source is globally detected in round 2. Consider the first round $k \geq 3$ in which no new fault is globally detected. Let p be correct. Since the source is globally detected in round 2, $|L_p| \geq 1$. Let w be a faulty processor not discovered by p by the end of round k. Then there exists a value v such that for at least $n - (t - |L_p|) \geq n - (t - 1)$ processors q, $tree_p(swq) = v$ before reordering in round k. Thus, there exist at most $t - |L_p| \leq t - 1$ correct processors r such that $tree_p(swr) \neq v$. Fix such an r, if one exists. Note that after reordering, the subtree of $tree_p$ rooted at sr differs from the subtrees of $tree_p$ rooted at a majority of the intermediate vertices. This is because after reordering, for most q, $tree_p(sqw) = v$, while $tree_p(srw) \neq v$. In this case we say that "w distinguishes the subtree rooted at sr from the majority."

We have already observed there can be at most $t - |L_p| \leq t - 1$ processors r such that undiscovered w can distinguish the subtree rooted at sr from the majority. Moreover, since s is globally detected, there are at most $t - |L_p| \leq t - 1$ undetected faults w. Thus, the undetected faulty processors together can distinguish from the majority at most $(t - |L_p|)^2 \leq (t - 1)^2$ subtrees rooted at vertices sr.

In particular, if q, r are correct and if, after reordering, the subtrees rooted at sq and sr are not distinguished from the majority, then for all w, $tree_p(sqw) = tree_p(srw)$, so $resolve_p(sq) = resolve_p(sr)$. Thus for some value v for at least

$$n - t - (t - |L_p|)^2 \geq n - t - (t-1)^2$$

correct processors q, $resolve_p(sq) = v$ for all correct p. Since $t \leq t_C$, this number is strictly greater than $n/2$, so condition (1) of Lemma 6 is satisfied and a persistent value is obtained.

Finally, consider the earliest round k in which all t faults have been globally detected. After fault masking and reordering, all leaves are common. If $k = t + 1$, then by the Frontier Lemma, s is common so all correct processors decide on the same value. If $k < t + 1$, then the children of s are common. This means s is common so condition (2) of Lemma 6 is satisfied. Thus by Lemma 6, this common value persists from round k on. ∎

4.4 Shifting Between Algorithms

Although we defined shifting in full generality, we have so far only used the technique for shifting between rounds of a single algorithm. In this section, we construct a t_A-resilient agreement algorithm by shifting from Algorithm A to Algorithm B and from there to Alglorithm C. We call this new algorithm the *hybrid* algorithm. The existence of the hybrid algorithm is surprising, since the resiliencies of Algorithms B and C are strictly less than t_A, the actual number of faults tolerated by the hybrid. The hybrid algorithm is faster and requires less local computation time than Algorithm A, but has identical resilience, maximum message length, and space requirements.

The hybrid algorithm has one parameter, b, the maximum number of rounds (after round 1) in any block. We will later compute two special values, t_{AB}, and t_{AC}. t_{AB} will be chosen so that it is "safe" to shift (from Algorithm A) into Algorithm B once t_{AB} faults have been globally detected or a persistent value has been obtained. Specifically, t_{AB} is chosen so that if t_{AB} faults are detected before shifting into Algorithm B, then Corollary 1 to the Hidden Fault Lemma holds after shifting into Algorithm B.[2] Similarly, t_{AC} will be chosen so that it is "safe" to shift into Algorithm C once t_{AC} faults have been globally detected or a persistent value has been obtained.

Let k_{AB} denote the minimum number of rounds, such that after k_{AB} rounds of Algorithm A, either a persistent value has been obtained or at least t_{AB} faults have been globally detected.

Let $t_{BC} = t_{AC} - t_{AB}$, and let k_{BC} denote the minimum number of rounds, such that after k_{BC} rounds of Algorithm B (after the shift into Algorithm B), either a persistent value has been obtained or an additional t_{BC} faults have been globally detected.

[2] Recall, this corollary shows that if all processors in a sequence αr are faulty but r is not detected, then αr is common, but the corollary is proved only for up to t_B faults.

The HybridAlgorithm(b):

{ **comment** Begin with Algorithm A }

Run Algorithm A with parameter b for exactly k_{AB} rounds;

{ **comment** Shift to Algorithm B}

$tree(s) = resolve'(s)$;

Run Algorithm B with parameter b for exactly k_{BC} rounds

beginning with round 2;

{ **comment** Shift to Algorithm C }

$tree(s) = resolve(s)$;

Run Algorithm C for exactly $t - t_{AC} + 1$ rounds beginning with round 2;

Decide $resolve(s)$;

Figure 3: The Hybrid Algorithm with Parameter b (t and n Fixed)

If we shift into the end of round 1 of Algorithm C with at most $t_A - t_{AC}$ undetected faults, then only an additional $t_A - t_{AC} + 1$ rounds are needed to obtain a common value (including one extra round in which the source is rediscovered). The total running time of the hybrid algorithm will be $k_{AB} + k_{BC} + t_A - t_{AC} + 1$.

The hybrid algorithm appears in Figure 3. We will first show how to compute k_{AB} and k_{BC} as functions of t_{AB} and t_{BC}, respectively. We will then compute t_{AB} and t_{AC} (and therefore t_{BC}).

We now show how to choose k_{AB} and k_{BC} as functions of t_{AB} and t_{BC}, respectively. We begin with k_{AB}. The goal is to find the minimum number of rounds such that after execution of rounds 1 through k_{AB} of Algorithm A, either a persistent value has been obtained or at least t_{AB} faults have been detected. Let us write

$$t_{AB} - 1 = (b-2)x + y,$$

where x and y are nonnegative integers and $y < b - 2$. Note that $x = \left\lfloor \frac{t_{AB}-1}{b-2} \right\rfloor$. After the first $1 + bx$ rounds of Algorithm A, if no persistent value has been obtained, then $1 + (b-2)x$ faults have been globally detected. Rewriting the last equation we have

$$t_{AB} - y = 1 + (b-2)x.$$

Thus, after $1 + bx$ rounds, if no persistent value has been obtained then $t_{AB} - y$ faults have been globally detected. To detect an additional y faults (or obtain a persistent value), it suffices to run Algorithm A for an additional $y + 2$ rounds. Thus,

$$\begin{aligned} k_{AB} &= 1 + bx + y + 2 \\ &= 3 + bx + y \end{aligned}$$

$$
\begin{aligned}
&= 3 + bx + t_{AB} - 1 - (b-2)x \\
&= 2 + t_{AB} + 2x \\
&= 2 + t_{AB} + 2\left\lfloor \frac{t_{AB}-1}{b-2} \right\rfloor .
\end{aligned}
$$

We now turn to k_{BC}. Recall that $t_{BC} = t_{AC} - t_{AB}$. Intuitively, our goal is to shift into Algorithm C with a persistent value or with at least t_{AC} faults detected. If there is no persistent value, then when we leave Algorithm A at least t_{AB} faults have been globally detected, so while in Algorithm B either a persistent value must be obtained or t_{BC} new faults must be detected. The analysis is similar to the one just performed for k_{AB}, only this time we shift into the *end* of round 1 of Algorithm B, while we began execution of Algorithm A at the *beginning* of round 1. Thus, we write

$$t_{BC} = (b-1)x' + y',$$

where x' and y' are nonnegative integers and $y' < b-1$. Note that $x' = \left\lfloor \frac{t_{BC}}{b-1} \right\rfloor$. Recall that t_{AB} is chosen such that if t_{AB} faults are detected (and thereafter masked) during execution of Algorithm A, then Corollary 1 to the Hidden Fault Lemma holds after shifting into Algorithm B. On the other hand, the Strong Persistence Lemma will guarantee that a persistent value found in Algorithm A remains persistent after the shift. Thus, after bx' rounds of Algorithm B, if no persistent value has been obtained, then $(b-1)x'$ new faults have been globally detected. Running Algorithm B an additional $y'+1$ rounds yields

$$
\begin{aligned}
k_{BC} &= bx' + y' + 1 \\
&= bx' + t_{BC} - (b-1)x' + 1 \\
&= 1 + t_{BC} + x' \\
&= 1 + t_{BC} + \left\lfloor \frac{t_{BC}}{b-1} \right\rfloor .
\end{aligned}
$$

Proof of the Main Theorem:

When a faulty processor is globally detected, its messages are masked, so its ability to prevent emergence of a persistent value is destroyed. Thus, the intuition behind being able to shift from Algorithm A to Algorithm B (and, later, down to Algorithm C) is that if, after some number k_{AB} of rounds, a persistent value has not been obtained, then the number of undetected faults is sufficiently small that we can safely shift into the end of round 1 of Algorithm B. On the other hand, if a persistent value has been obtained, then, by the Strong 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 conversion in Algorithm B will be the persistent value.

It remains to determine the least t_{AB} such that, when no persistent value has been obtained during execution of Algorithm A, but t_{AB} faults have been globally detected, then once we shift

into Algorithm B, Corollary 1 to the Hidden Fault Lemma can be applied as in the proof of correctness of Algorithm B. This Corollary is critical because, speaking informally, it guarantees that the adversary must allow $b-1$ new faults to be detected in each block of b rounds that does not result in a persistent value. The difficulty is that the Corollary is proved only for $t_B < t_A$ faults, but t_A processors may actually be faulty.

Specifically, after the shift we require that, if αr is any internal node (with all processors in αr faulty) and some correct processor does not discover r once it receives values for the children of αr, then αr is common. As we now explain, this is achieved provided $n - 2t_A + t_{AB} > \left\lfloor \frac{n-1}{2} \right\rfloor$. First, note that if t_{AB} faults are globally detected then for every correct processor p, $|L_p| \geq t_{AB}$. Let p be correct and let αr be an internal node. If all processors in αr are faulty but $r \notin L_p$, then by the Hidden Fault Lemma there is some value v such that v is stored in at least $n - 2t_A + |L_p| \geq n - 2t_A + t_{AB}$ children of αr corresponding to correct processors. By the Correctness Lemma, these children are common. To guarantee that αr is common with converted value v, it suffices that these children form a strict majority of the ($\leq n-1$) children of αr. Thus, we need

$$n - 2t_A + t_{AB} > \left\lfloor \frac{n-1}{2} \right\rfloor,$$

so we must take $t_{AB} \geq \left\lfloor \frac{t_A}{2} \right\rfloor$.

Now we consider the shift after $k_{AB} + k_{BC}$ rounds of A and B to the end of round 1 of Algorithm C. First suppose a persistent value has been obtained by the end of round $k_{AB} + k_{BC}$. This means that s is common in the converted round 1 tree and condition (2) of Lemma 6 is satisfied. Note that Lemma 6 holds provided the faulty processors are a strict minority. Thus the persistent value will be preserved through the end of the hybrid algorithm. Alternatively, suppose that there are t_{AC} detected faults at the end of round $k_{AB} + k_{BC}$. The proof of Proposition 4 for the case in which the source is globally detected in round 2 holds when $n - t - (t - |L_p|)^2 > \frac{n}{2}$ for any correct processor p. Since t_{AC} is a lower bound on $|L_p|$ for all correct processors p, it suffices for t_{AC} to satisfy

$$n - t_A - (t_A - t_{AC})^2 > \frac{n}{2}.$$

Solving for t_{AC}, we obtain $t_{AC} > t_A - \sqrt{\frac{n-2t_A}{2}}$. Recall that $n \in \{3t_A+1, 3t_A+2, 3t_A+3\}$ so it suffices for t_{AC} to satisfy

$$t_{AC} \geq \left\lfloor t_A - \sqrt{\frac{t_A+1}{2}} \right\rfloor + 1.$$

We shift into Algorithm C only after either a persistent value has been obtained or at least t_{AC} faults have been globally detected. Thus we run Algorithm C only an additional $t_A - t_{AC} + 1$ rounds. In the first round after the shift (corresponding to round 2 of Algorithm C) the source s may be redetected. Thereafter, at least one new fault is detected per round until a persistent value is obtained. If all remaining $t_A - t_{AC}$ undetected faults become globally detected in round k, then a persistent value is obtained by round k (see the proof of Proposition 4).

Summarizing, let $t = t_A$, and let t_{AB}, and $t_{BC} = t_{AC} - t_{AB}$ be as in the above discussion. The number of rounds required by the hybrid algorithm, when run with parameter b, is

$$\begin{aligned} & k_{AB} + k_{BC} + t - t_{AC} + 1 \\ = \; & 2 + t_{AB} + 2\left\lfloor \frac{t_{AB} - 1}{b-2} \right\rfloor + 1 + t_{BC} + \left\lfloor \frac{t_{BC}}{b-1} \right\rfloor + t - t_{AC} + 1 \\ = \; & t + 2\left\lfloor \frac{t_{AB} - 1}{b-2} \right\rfloor + \left\lfloor \frac{t_{BC}}{b-1} \right\rfloor + (t_{AB} + t_{BC} - t_{AC}) + 4 \\ = \; & t + 2\left\lfloor \frac{t/2 - 1}{b-2} \right\rfloor + \left\lfloor \frac{t/2 - \sqrt{(t+1)/2} + 1}{b-1} \right\rfloor + 4. \end{aligned}$$

5 Recent Results

Since the first appearance of these results [1], a number of agreement algorithms superseding our algorithms have appeared in the literature. All those of which we are aware make use of some of the techniques used in this paper. We briefly note some of these here.

Moses and Waarts obtained the first linearly resilient agreement algorithm with polynomial (in n) communication complexity and local computation time, and requiring only $t+O(1)$ rounds [10]. Indeed, their algorithm runs in exactly $t + 1$ rounds. They have since improved the resiliency slightly, with no cost in these measures. Although not based on the technique of shifting, their algorithms rely heavily on the ideas presented in this paper, together with a new technique they call "Coordinated Traversal." Waarts informs us that it is possible to shift into both of the Moses and Waarts algorithms [12].

More recently, Berman, Garay, and Perry have obtained a $t + 1$-round agreement algorithm requiring only $n > 4t$ processors [2]. Considerably less complex than the Moses and Waarts algorithms, Berman, Garay, and Perry's algorithm uses a strengthening of the fault masking technique presented here, together with a simple and elegant technique of "Cloture Agreement." As the name suggests, cloture agreement is a primitive that makes it possible to close debate and force an output at an early time. Since the underlying algorithm in Berman, Garay, and Perry's work is again the Exponential Algorithm, it may be possible to shift into the Berman and and Garay algorithm. Finally, Coan and Welch [6] have obtained optimally resilient agreement algorithms running in time $t + o(t)$ using a "modular" approach similar in spirit to our shifting technique. Berman, Garay, and Perry have exhibited similar bounds [3].

6 Concluding Remarks

We have constructed Algorithms A, B, and C and shown that it is possible to shift from A to B and from B to C with resulting improvements in speed and resilience. When can we shift

from one algorithm to another in a way that provides a better combination of our performance measures than that of either of the algorithms separately? We do not have explicit necessary or sufficient conditions for such a successful application of shifting. 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.

Acknowledgements

The authors thank the referees for their in-depth reading of the paper and their numerous suggestions.

References

[1] Bar-Noy, A., Dolev, D., Dwork, C., and Strong, H.R., "Shifting Gears: Changing Algorithms on the Fly to Expedite Byzantine Agreement," *Proceedings of the Sixth ACM Symposium on Principles of Distributed Computing*, pp. 42-51, 1987.

[2] Berman, P. and Garay, J., and Perry, K. J., "Towards Optimal Distributed Consensus," *Proceedings of the Thirtieth Symposium on Foundations of Computer Science*, pp. 410-415, 1989.

[3] Berman, P. and Garay, J., and Perry, K. J., "Recursive Phase King Protocols for Distributed Systems," *Technical Report 89-24*, Department of Computer Science, The Pennsylvania State University, August 1989.

[4] Coan, B.A., "A Communication-Efficient Canonical Form for Fault-Tolerant Distributed Protocols," *Proceedings of the Fifth ACM Symposium on Principles of Distributed Computing*, pp. 63-72, 1986.

[5] Coan, B.A., "Achieving Consensus in Fault-Tolerant Distributed Computer Systems: Protocols, Lower Bounds, and Simulations," *PhD Thesis*, MIT, June 1987.

[6] Coan, B.A., and Welch, J.L., "Modular Construction of Nearly Optimal Byzantine Agreement Protocols," *Proceedings of the Eighth ACM Symposium on Principles of Distributed Computing*, pp. 295-305, 1989.

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

[8] Dolev, D. and Strong, H.R., "Authenticated Algorithms for Byzantine Agreement," *SIAM Journal of Computing 12*(4) (1983), pp. 656-666.

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

[10] Moses, Y. and Waarts, O., "$(t+1)$-Round Byzantine Agreement in Polynomial Time," *Proceedings of the 29th Symposium on Foundations of Computer Science*, pp. 246-255, 1988.

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

[12] Waarts, O., private communication.