חשוב פולק – אלגוריתמים מבוזרים

Distributed Computing (1991) 4:105–110

DISTRIBUTED COMPUTING

Consensus algorithms with one-bit messages*

Amotz Bar-Noy[1] and Danny Dolev[2]

[1] IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY 10598, USA
[2] IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120, USA and the Computer Science Department, Hebrew University, Jerusalem, Israel

Received April 18, 1989/Accepted April 12, 1990

Amotz Bar-Noy received the B.A. degree in mathematics and computer science in 1981, and the Ph.D. degree in computer science in 1987, both from the Hebrew University of Jerusalem, Israel. He was a post-doctoral fellow at Stanford University, California, from October 1987 to September 1989. Since October 1989 he has been a visiting scientist at IBM T.J. Watson Research Center, Yorktown Heights, New York. His research interests include communication networks, distributed algorithms, parallel algorithms and fault-tolerant computing.

Danny Dolev received a B.Sc. in physics from the Hebrew University, Jerusalem, in 1971, an M.Sc. in applied mathematics from the Weizmann Institute of Science, Israel, in 1973, and Ph.D. in computer science in 1979. After two years as a post doctoral fellow at Stanford and a year as a visiting scientist at IBM, he joined the Hebrew University, Jerusalem, in 1982, and IBM Almaden Research Center at 1987. His major research interests are distributed computing, reliability of distributed systems, and algorithms.

Abstract. Three main parameters characterize the efficiency of algorithms that solve the Consensus Problem: the ratio between the total number of processors and the maximum number of faulty processors (n and t, respectively), the number of rounds, and the upper bound on the size of any message. In this paper we present a trade-off between the number of faulty processors and the number of rounds by exhibiting a family of algorithms in which processors communicate by one-bit messages. Let k be a positive integer and let $s=t^{1/k}$. The family includes algorithms where the number of processors is less than $5t^{\frac{(k+1)}{k}}=5\cdot s\cdot t$, and the number of rounds is less than $t+3t^{\frac{k-1}{k}}=1+\frac{3}{s}$. This family is based on a very simple algorithm with the following complexity: $(2t+1)(t+1)$ processors, $t+1$ rounds, and one-bit message size.

Key words: Consensus – Fault tolerance – One-bit messages

* A preliminary version of this paper appeared in the Aegean Workshop on Computing (AWOC), pp 380–390, 1988. This work was carried out while Dr. Bar-Noy was visiting Stanford University. Supported in part by a Weizmann fellowship, by contract ONR N00014-88-K-0166, and by a grant of Stanford's Center for Integrated Systems

Offprint requests to: D. Dolev

1 Introduction

A major task in fault tolerant distributed systems is to agree on common values. The importance of the task stems from the need to overcome the uncertainty that faults introduce. The famous *Byzantine Generals* problem is related to this task and has received much attention in the literature (e.g. [7]). In this paper we deal with a variant of this problem called the *Consensus* problem (first formulated in [7]). This problem is defined as follows. Let $\mathscr{P}$ be a distributed system of n processors where up to t of them might be *faulty*. Each processor has an initial binary value. The goal is to find an algorithm where each non-faulty processor *decides* on a binary value under the following two conditions:

Agreement. All the non-faulty processors decide on the same binary value.

Validity. If the initial value is identical for all the non-faulty processors, then this value will be the decision value.

In this paper a standard and simple model is assumed:

- Processors are synchronized by rounds of communication in which each executes three operations: sending messages, receiving messages, and performing a local computation. By the last round of the algorithm, after the local computation is completed, the processor must *decide*.
- The communication is carried out by a complete and reliable network; each processor can send a message to every other processor and messages arrive unaltered in the same round they are sent.
- The processors are deterministic; no randomized operations are allowed.
- Faulty processors are malicious and might collude in order to prevent reaching valid agreement.
- For ease of exposition, processors' names are assumed to be the numbers $1, \ldots, n$.

Three complexity measured determine the efficiency of a solution: the ratio between n and t; the number of rounds required in the worst case, denoted by r; and the upper bound on the size of any message, denoted by m. Traditionally, researches bounded the total number of messages (or bits) sent during the algorithm, instead of bounding the maximum size of a message. In order to better understand the relationship between m and the other two parameters (n/t and r), it is useful to restrict m by preventing processors from reporting too much information in a single message. In this paper, we restrict the maximum size of any message to be at most one bit, that is, in each round a processor may send at most one bit to each processor. Algorithms in which only one-bit messages are allowed are called *one-bit algorithms*.

In many existing algorithms, messages contain identities of processors. Encoding such algorithms into one-bit algorithms requires many rounds ($\log n$ rounds for each processor identity). Therefore, the existence of one-bit algorithms for the consensus problem, where the number of rounds does not increase as a function of n, teaches us that a solution for the consensus problem does not require processors to relay identities of processors. We view this as a necessary step in understanding the relationship among these three complexity measures.

There are three known lower bounds for these complexity measures. These lower bounds are independent in the sense that they are valid no matter what the values of the other parameters are. The lower bounds are:

- $n \geq 3t+1$ [7].
- $r \geq t+1$ [5].
- $m \geq 1$, (obvious).

The first upper bound appeared in [7] (a simple presentation of this algorithm can be found in [1]). This algorithm optimizes the number of processors and rounds ($n=3t+1$ and $r=t+1$), but messages have size exponential in n. In [6, 3] algorithms are presented where $r=t+1$, is linear in t and m is polynomial in t. Another known result is an algorithm with $n=3t+1$, $r=2t+1$ and m is polynomial in n [8]. Another one-bit algorithm was presented in [2] with $n=4t+1$ and $r=2t+2$.

Recently a family of algorithms with an additional parameter d was presented [4, 1]. Every algorithm in this family has the following complexity: $n=3t+1$, $r=t+\lceil t/d \rceil$, and $m=t^{O(d)}$. In the extreme, where $d=t$ or $d=1$, these algorithms coincide with two of the above algorithms. In this paper we present another family, a family of one-bit algorithms which exhibits a trade-off between the number of faulty processors and the number of rounds. All these families of algorithms optimize one complexity measure at the expense of the other two measures. Thus, the following question arises:

Question. Is there an algorithm that optimizes all three parameters?

We view the results of this paper as a step towards answering this question. We suggest the following problem as a candidate for a negative answer to the above question.

Problem. Is there a one-bit algorithm that terminates after $t+1$ rounds with $n=0(t)$?

The algorithms presented in this paper have other interesting properties. First, the decidsion of when to send a message is a function of the round number only. Second, each processor is required only to compute a majority of values arriving on a set of wires. Therefore, in addition to the fact that the algorithms use only one-bit messages, it is not hard to implement them in hardware.

Our basic one-bit algorithm, $\mathscr{A}_{t,1}$, has complexity $n=(2t+1)(t+1)$ and $r=t+1$. Furthermore, in this algorithm each processor sends a one-bit message exactly once. Using a parameter $k \geq 1$ and assuming $t=s^k$ for some integer $s \geq 2$, we derive from Algorithm $\mathscr{A}_{t,1}$ a family of one-bit algorithms, $\{\mathscr{A}_{t,k}\}_{k \geq 1}$. The complexity of Algorithm $\mathscr{A}_{t,k}$ is:

- $n=4t^{\frac{k+1}{k}}+t^{\frac{k-1}{k}} \leq 5t^{\frac{k-1}{k}}$, and
- $r=\sum_{i=0}^{k} t^{\frac{i}{k}}+t^{\frac{k-1}{k}} \leq t+3t^{\frac{k-1}{k}}$.

As $t=s^k$, it follows that $t^{\frac{k-1}{k}}=\frac{t}{s}$ and that $t^{\frac{k-1}{k}}=s \cdot t$. Therefore, another way to represent the above equation is:

- $n \leq 5 \cdot s \cdot t$, and
- $r \leq \left(1+\frac{3}{s}\right)t$.

Asymptotically, fix $k \geq 1$ and $0<\varepsilon+1$, then, for a sufficiently large t, there exists an algorithm where $r \leq (1+\varepsilon)t$ and n is almost linear in t.

2 Algorithm $\mathscr{A}_{t,1}$

This section describes Algorithm $\mathscr{A}_{t,1}$ which is the basic one-bit algorithm. In this algorithm the number of processors is $n=(2t+1)(t+1)$ and the number of rounds is $r=t+1$.

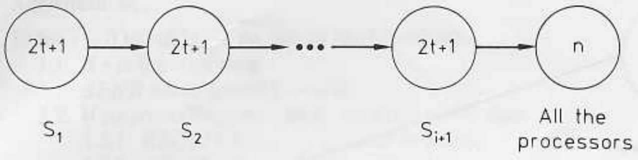

Fig. 1. The partition of the processors and the flow of messages in Algorithm $\mathscr{A}_{t,1}$

Algorithm $\mathscr{A}_{t,1}$;

```
1. for $r := 1$ to $t+1$ do one of the following
   1.1. if $r = round(p) = 1$ then
            $v := val$;
            SEND $v$ to processors $(2t+1)+1, \ldots, 2(2t+1)$;
   1.2. if $1 \le r = round(p) - 1$ then
            RECEIVE the messages of processors
               $(r-1)(2t+1)+1, \ldots, r(2t+1)$;
            $v :=$ the majority value of these $2t+1$ values;
   1.3. if $1 < r = round(p) < t+1$ then
            SEND $v$ to processors $r(2t+1)+1, \ldots, (r+1)(2t+1)$;
   1.4. if $r = round(p) = t+1$ then
            SEND $v$ to all processors;
2. at the end of round $t+1$ do
   2.1. RECEIVE the messages of processors
            $t(2t+1)+1, \ldots, (t+1)(2t+1)$;
   2.2. $v :=$ the majority value of these $2t+1$ values;
   2.3. DECIDE on $v$;
```

Fig. 2. Algorithm $\mathscr{A}_{t,1}$ for processor p with initial value val

Informally, the algorithm can be described as follows. The n processors are partitioned into $t+1$ disjoint sets, each of cardinality $2t+1$. Denote these sets by $S_1, S_2, \ldots, S_{t+1}$. In general, in round i, only processors from set S_i send messages and only to processors from set S_{i+1}. More specifically, in the first round all the processors in S_1 send their initial value to every processor in S_2. In round $i-1, 1<i<t+1$, each processor $p \in S_i$ receives $2t+1$ bits from processors in S_{i-1}. In round i, p sends the majority value of these $2t+1$ values to every processor in S_{i+1}. In the last round $i=t+1$, each processor $p \in S_{t+1}$ sends the majority value to all the processors. The decision value for every processor is the majority value of the $2t+1$ bits it has received from the processors in S_{t+1} in round $t+1$. This informal description is depicted in Fig. 1.

The formal description of the algorithm, for processor p whose initial value is *val*, appears in Fig. 2. In the code, the function *round*(i) is defined to be $\left\lceil \frac{i}{2t+1} \right\rceil$. In the operation *RECEIVE*, 0 is the default value for a message that did not arrive or did not consist of a single bit.

Proof of correctness. For $1 \le i \le t+1$, define

$S_i = \{(i-1)(2t+1)+1, \ldots, i(2t+1)\}$.

Note that if $p \in S_i$ then *round*$(p) = i$.

There are $t+1$ disjoint sets and at most t faulty processors, implying the existence of a set S_i that does not contain any faulty processors. If $i=t+1$, then all the processors receive an identical set of $2t+1$ binary values (Step. 2.1) and therefore decide on the same value (Steps 2.2 and 2.3).

If $i<t+1$, then all the non-faulty processors in S_{i+1} receive the same $2t+1$ bits and compute the same majority value (Step 1.2). In S_{i+1} there are $2t+1$ processors, at least $t+1$ of which are non-faulty. Denote by u the majority value computed by these $t+1$ processors. From that point on in all rounds $j, i<j<t+1$, all the non-faulty processors in S_{j+1} will compute u as their majority value (Step 1.2) and in round $t+1$ all non-faulty processors will compute u as their majority value (Step 2.2). Thus, all non-faulty processors will decide on u (Step 2.3) and the agreement condition holds.

Now assume that all the non-faulty processors have u as their initial value. The same arguments as before prove that u will be the decision value of all non-faulty processors and the validity condition holds. □

The following remarks present possible extensions of the algorithm that will later be used in the more general algorithms.

Remark 1. If $n > (2t+1)(t+1)$, then the processors can be partitioned into disjoint sets of almost equal cardinality (the cardinality of every two sets differs by at most 1). The proof of correctness holds as the non-faulty processors form a majority in each set.

Remark 2. Assume we add the following initial round to Algorithm $\mathscr{A}_{t,1}$. In this round all the processors send their original initial value to all other processors. Then they adopt the majority of the n values they received as their new initial value and apply Algorithm $\mathscr{A}_{t,1}$ with this new initial value. It is not hard to see that if $\left\lceil \left(\frac{n+1}{2}\right) + t \right\rceil$ non-faulty processors have v as their initial value then all non-faulty processors decide on v.

3 The $\mathscr{A}_{t,k}$ family of algorithms

A generalization of Algorithm $\mathscr{A}_{t,1}$ into a family of one-bit algorithms $\{\mathscr{A}_{t,k}\}_{k \ge 1}$ is presented in this Section. Each algorithm in the family depends on two parameters, t and $k, 1 \le k \le t$. For simplicity, in the following presentation we assume that $t = s^k$, for some integer $s \ge 2$. The number of processors is:

$$n(t,k) = 4t^{\frac{k+1}{k}} + t^{\frac{k-1}{k}} \le 5t^{\frac{k+1}{k}},$$

and the number of rounds is:

$$r(t,k) = \sum_{i=0}^{k} t^{\frac{i}{k}} + t^{\frac{k-1}{k}} \le t + 3t^{\frac{k-1}{k}}.$$

In order to present Algorithm $\mathscr{A}_{t,k}$, a tree $T(t,k)$ is defined. The tree is recursively described, but the algorithm itself will be explicit. In this tree, each node represents a subset of the $n(t,k)$ processors.

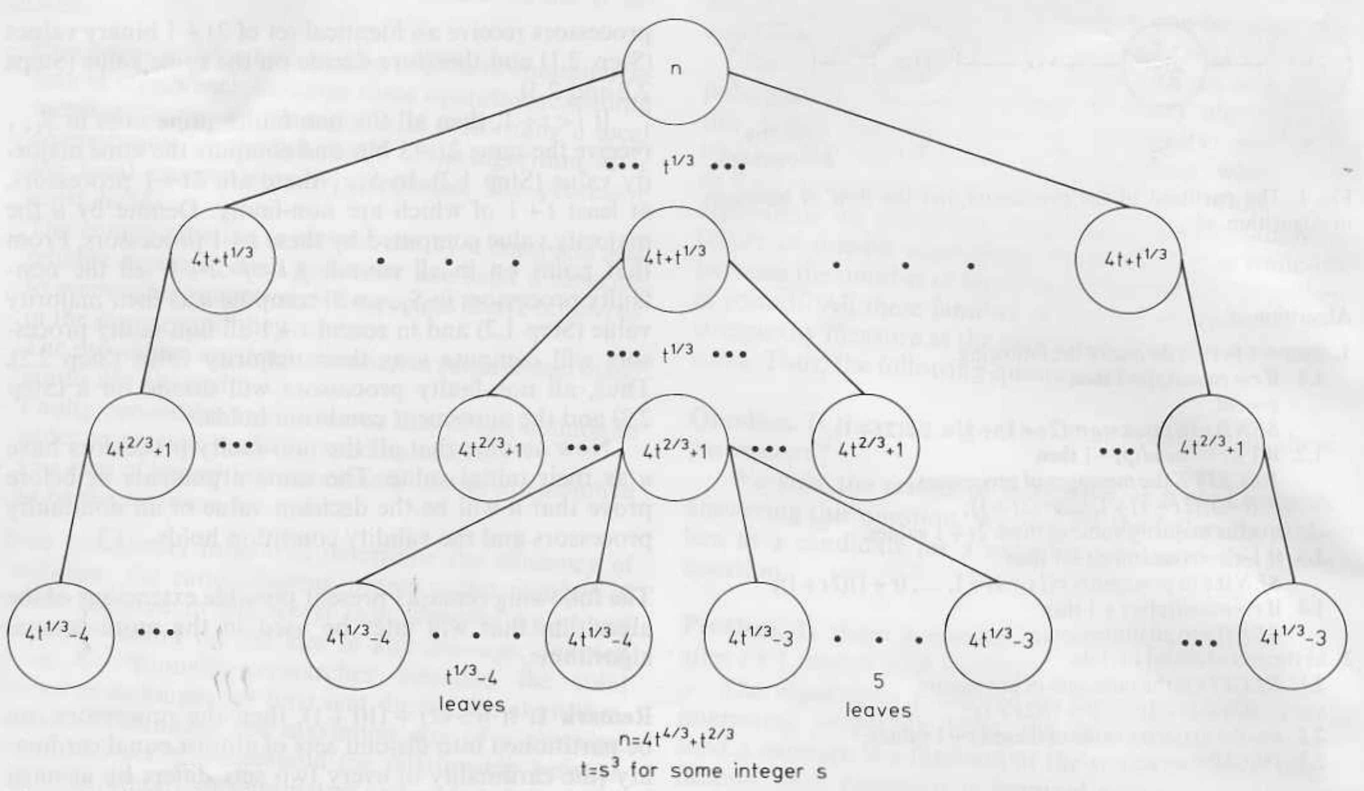

Fig. 3. The description of the tree, $k=3$

- If $k=1$, the n processors are partitioned into $t+1$ disjoint sets of almost equal cardinality (the cardinality of every two sets differs by at most 1). The root of the tree is the set of all n processors and the children are these $t+1$ sets.
- If $k>1$, the processors are partioned into $s=t^{\frac{1}{k}}$ disjoint sets of equal cardinality. Define $t'=t^{\frac{k-1}{k}}$ and $n'=n(t', k-1)$. The root of this tree is the set of all n processors and each child of the root is the root of a $T(t', k-1)$ tree associated with one of these disjoint sets.

Note that the recursive construction can indeed be applied as,

$$n'=4t'^{\frac{(k-1)+1}{k-1}}+t'^{\frac{(k-1)-1}{k-1}}$$
$$=4(t^{\frac{k-1}{k}})^{\frac{k}{k-1}}+(t^{\frac{k-1}{k}})^{\frac{k-2}{k-1}}=4t+t^{\frac{k-2}{k}}=\frac{n}{t^{1/k}}.$$

In Fig. 3, an example of such a tree for $k=3$ is given.

When unfolding the recursion, one can see that for each processor there is a corresponding path in the tree starting at the root and ending at a leaf. Each processor belongs to every set that is represented by the nodes of its corresponding path. The set represented by the root consists of all the processors.

The tree is assumed to be ordered from left to right. The root is numbered 0, and the rest of the nodes are numbered in a post-order fashion. Hereafter, each node in the tree is referred to by its post-order number and the set of processors in node r is denoted by *procs*(r).

For a given tree, the following definitions and notations are used in specifying the rules of the algorithm:

- *Left*: the set of all the nodes in the tree that do not have a left sibling.
- *Right*: The set of all the nodes that do not have a right sibling.
- *Start*: The nodes that have no left siblings and have no ancestors with left siblings (the left edge of the tree).
- *RightMost*(x): The rightmost child of node x (undefined for leaves).
- *LeftMost*(x): The leftmost child of node x (undefined for leaves).
- *Next*(x): A function from the nodes of the tree to the nodes of the tree.
  - $x=0$ (the root): $Next(0)=LeftMost(x)$.
  - $x\notin Right$: $Next(x)=$ the right sibling of x.
  - $x\in Right (x\neq 0)$: $Next(x)=$ the parent of x.
- *Prev*(x): A function from the nodes of the tree to the nodes of the tree.
  - $x\notin Left$: $Prev(x)=$ the left sibling of x.
  - $x\in Start$: $Prev(x)=0$ (the root).
  - $x\in Left \wedge x\notin Start$: $Prev(x)=Prev(y)$, where y is the parent of x. That is, $Prev(x)$ is the left sibling of the closest ancestor that has a left sibling.

In the algorithm, the number of rounds is the same as the number of nodes in the tree. In each round r, $0\leq r<r(t,k)$, processors in *procs*(r) send a one-bit message to the processors in *procs*$(Next(r))$. The following rules determine the value of the bit, denoted by v, that processor $p\in procs(r)$ should send.

- If $r=0$ (node r is the root), then v is the initial value of processor p.

```
Algorithm $\mathscr{A}_{t,k}$;
1. for $r := 0$ to $r(t,k)-1$ do one of the following
   1.1. if $r$ is the root then
        SEND val to procs(Next(r)).
   1.2. if $p \in procs(Next(r))$ and $r \notin Right \setminus \| \{0\}$ then
        1.2.1. RECEIVE $v_1, \ldots, v_w$ from procs(r);
        1.2.2. $v(Prev) :=$ the majority value of $v_1, \ldots, v_w$;
        1.2.3. $support(\ell(w)) := |\{i : v_i = v\}|$
   1.3. if $p \in procs(Next(r))$ and $r \in Right \setminus \{0\}$ then
        1.3.1. RECEIVE $v_1, \ldots, v_w$ from procs(r);
        1.3.2. $v(RightMost) :=$ the majority value of $v_1, \ldots, v_w$;
   1.4. if $p \in procs(r)$ and $r$ is a leaf then
        SEND v(Prev) to procs(Next(r));
   1.5. if $p \in procs(r)$ and $r$ is neither a leaf nor the root then
        1.5.1. if $support(\ell(w)) \geq w - t^{\ell/k}$
               then SEND v(Prev) to procs(Next(r));
                 (* the forwarding rule *)
               else SEND v(RightMost) to procs(Next(r));
                 (* the calculating rule *)
2. at the end of round $r(t,k)$ do
   2.1. RECEIVE $v_1, \ldots, v_{n'}$ from procs(RightMost(0));
   2.2. $v :=$ the majority value among $v_1, \ldots, v_{n'}$;
   2.3. DECIDE on $v$;
```

Fig. 4. Algorithm $\mathscr{A}_{t,k}$ for processor p with initial value val

- If node r is a leaf, then v is the majority of the values that p received from $procs(Prev(r))$. Note that this is well defined, as $procs(Next(Prev(r))) \supseteq procs(r)$.
- Otherwise (node r is neither the root nor a leaf), the set of processors represented by $procs(Prev(r))$ consists of $w = 4^{\ell/k} + t^{(\ell-2)/k}$ processors, for some ℓ that is a function of the level of node $Prev(r)$ in the tree.

The forwarding rule. In case p received some value u from at least $w - t^{\ell/k}$ processors from $procs(Prev(r))$, then $v = u$.

The calculating rule. Otherwise, v is the majority of the values p received from $procs(RightMost(r))$. Again, this is well defined as $Next(RightMost(r)) = r$.

The **decision** value is the majority of the values the processor received from $RightMost(0)$, (the rightmost child of the root).

A more formal description of the algorithm appears in Fig. 4. As in Algorithm $\mathscr{A}_{t,1}$,0 is the default value for any message that did not arrive or includes more than one bit. The root is node 0 and the algorithm distinguishes between leaf and non-leaf nodes. For any w of the form $w = 4t^{(\ell-2)/k}$, define $\ell(w) = \ell - 1$.

Observation 1. If $k=1$, the above description is exactly Algorithm $\mathscr{A}_{t,1}$ after applying the two modifications mentioned in Remarks 1 and 2 at the end of the previous section.

Observation 2. Let r be a node in level $\ell, 0 < \ell < k$, where the level of a leaf is 0 and the level of the root is k. Denote by $\mathscr{B}_r$ the sub-algorithm performed by $procs(r)$ in the sub-tree rooted in r. Then $\mathscr{B}_r$ differs from $\mathscr{A}_{t^{\ell/k},l}$ only in the first round. In $\mathscr{A}_{t^{\ell/k},\ell}$, the processors in $procs(r)$ should send their initial value to $procs(LeftMost(r))$ in the first round. In $\mathscr{B}_r$, processors in $procs(LeftMost(r))$ receive messages from $procs(Prev(r))$. We claim that the correctness of $\mathscr{A}_{t^{\ell/k},\ell}$ implies the correctness of $\mathscr{B}_r$, where the validity condition is related to the initial values of processors in $procs(Prev(r))$. This is true because $|procs(r)| \leq |procs(Prev(r))|$ and processors in $procs(r)$ could have the initial values of some subset of $procs(Prev(r))$.

Proof of correctness. We first prove that the validity condition holds. Recall that $s = t^{\frac{1}{k}}$ and let $r_1, \ldots, r_s$ be the children of the root. The number of processors in $procs(r_i)$ is $4t + t^{(k-2)/k}$, where at least $3t + t^{(k-2)/k}$ of them are non-faulty processors. Therefore, if v is the initial value of all the non-faulty processors, then in each $procs(r_i)$, the processors apply the forwarding rule (Steps 1.2.3 and 1.5.1) when they send messages to $procs(Next(r_i))$. As the root is $Next(r_s)$, all the processors decide on v and the validity condition holds.

The proof for the agreement condition is by induction on k and t. When $k=1$, it follows from the agreement condition stisfied by Algorithm $\mathscr{A}_{1,t}$ and the above Observation 1. Now assume that the agreement condition holds for $1 \leq k' < k$ and $1 \leq t' < t$ and let us prove it for k and t.

A pigeon hole argument implies the existence of a set $procs(r_i)$ with at most $t' = t^{\frac{k-1}{k}}$ faulty processors. Otherwise, there would be a total of more than $t^{\frac{1}{k}} \cdot t' = t$ faulty processors.

If, in this set $procs(r_i)$, there exists a processor that applies the forwarding rule, then this processor must necessarily have received some value u from at least $n'-t$ processors ($n-t$, if $i=1$). In this case, every other non-faulty processor in $procs(r_i)$ receive u from a majority of at least $n' = 2t > n'/2$ processors ($n - 2t > n/2$, if $i=1$). By the above Observation 2 and by the induction hypothesis, the agreement condition satisfied by $\mathscr{A}_{t',k-1}$ implies that all the non-faulty processors send u to $procs(r_{i+1})$ (or to all other processors if $i=s$). The same arguments as in the proof that the validity condition holds, show that this value is preserved until it becomes the decision value.

The remaining case is when all the processors in $procs(r_i)$ apply $\mathscr{B}_{r_i}$ (Observation 2). Then again, by the same arguments as before, it follows that all the non-faulty processors in $procs(r_i)$ send the same value to $procs(r_{i+1})$ (or to all other processors if $i=s$). This value is forwarded until decision is reached on it and the agreement condition holds. □

Number of rounds. Recall that $r(t,k)$, the number of rounds in the algorithm, is also the number of nodes in the tree $T(t,k)$. We prove by induction that

$$r(t,k) = \sum_{i=0}^{k} t^{\frac{i}{k}} + t^{\frac{k-1}{k}}.$$

When $k=1$, the tree has $t+2$ nodes. This is the desired value, because

$$r(t,1) = t^{\frac{0}{1}} + t^{\frac{1}{1}} + t^{\frac{0}{1}} = t+2.$$

When $k>1$, the construction of the tree implies

$$
\begin{aligned}
r(t,k) &= 1+t^{\frac{1}{k}}r(t',k-1)\\
&= 1+t^{\frac{1}{k}}\sum_{i=0}^{k-1} t'^{\frac{i}{k-1}}+t'^{\frac{k-2}{k-1}}\\
&= 1+t^{\frac{1}{k}}\sum_{i=0}^{k-1} (t^{\frac{k-1}{k}})^{\frac{i}{k-1}}+t^{\frac{1}{k}}(t^{\frac{k-1}{k}})^{\frac{k-2}{k-1}}\\
&= 1+t^{\frac{1}{k}}\sum_{i=0}^{k-1} t^{\frac{i}{k}}+t^{\frac{k-1}{k}}\\
&= t^{\frac{0}{k}}+\sum_{i=1}^{k} t^{\frac{i}{k}}+t^{\frac{k-1}{k}}\\
&= \sum_{i=0}^{k} t^{\frac{i}{k}}+t^{\frac{k-1}{k}}.
\end{aligned}
$$

Notice that since

$$\sum_{i=0}^{k-2} t^{\frac{i}{k}}=\sum_{i=0}^{k-2} s^i \leq \frac{s^{k-1}-1}{s-1}\leq s^{k-1},$$

we conclude

$$r(t,k)=\sum_{i=0}^{k} t^{\frac{i}{k}}+t^{\frac{k-1}{k}} \leq t+3t^{\frac{k-1}{k}}. \quad \square$$

Remark 3. In this paper, we present a family of one-bit algorithms depending on two parameters $1\leq k\leq t$. We bind t to be s^k for some integer $s\geq 2$. It is possible to modify the algorithm to work for any choice of $t\geq k$ but then the presentation and the proof are much more tedious. We believe that the idea is well demonstrated even for these restricted values of t.

Remark 4. Assume that the partition of the processors can be encoded in hardware. Then the implementation of all the algorithms presented in this section is straightforward (especially of Algorithm $\mathscr{A}_{t,1}$). The following properties of our algorithms illustrate their simplicity:

- The messages contain only one bit of information.
- The rounds in which processors receive or send messages can be controlled by a clock as they do not depend on the input.
- The processors need only "general majority gates" to compute the message they should send.

Remark 5. Finally, note that unlike some other solutions to the consensus problem, in our algorithms processors participate in "few" consecutive rounds. Consequently, many instances of the consensus problem can be solved in a pipeline fashion. It is not hard to verify that the time complexity of x instances of Algorithm $\mathscr{A}_{t,k}$ is $r(t,k)+x\cdot r(t^{\frac{k-1}{k}},k-1)$ instead of $x\cdot r(t,k)$. In particular, the overall number of rounds of x instances of Algorithm $\mathscr{A}_{t,1}$ is $t+x$.

Acknowledgement. The authors would like to thank Ray Strong for his part in developing the $\mathscr{A}_{t,1}$ algorithm.

References

1. Bar-Noy A, Dolev D, Dwork C, Strong HR: Shifting Gears: Changing algorithms on the fly to expedite Byzantine Agreement. Proc 6th ACM Symp. of Principles of Distributed Computing 1987, pp 42–51
2. Berman P, Garay JA: Asymptotically optimal distributed consensus. Proc Int Conf on Automata, Languages and Programming (ICALP), 1989
3. Berman P, Garay JA, Perry K: Towards optimal distributed consensus. Proc 30th Symp on Foundations of Computer Science, 1989
4. Coan BA: A communication-efficient canonical form for fault-tolerant distributed protocols. Proc. 5th ACM Symp of Principles of Distributed Computing 1986, pp 63–72
5. Fischer, M, Lynch N: A lower bound for the time to assure interactive consistency. Inf Process Lett 14:183–186 (1982)
6. Moses Y, Waarts O: $(t-1)$-round Byzantine Agreement in polynomial time. Proc 29th Symp on Foundations of Computer Science 1988, pp 246–255
7. Pease M, Shostak R, Lamport L: Reaching agreement in the presence of faults, JACM 27:228–234 (1980)
8. Toueg S, Perry KJ, Srikanth TK: Fast distributed agreement, SIAM J Comput 16:445–458 (1987)