Optimal Amortized Distributed Consensus*

Amotz Bar-Noy [†] Xiaotie Deng [‡] Juan A. Garay [§] Tiko Kameda [¶]

April 27, 1994

Abstract

In this paper we study the behavior of deterministic algorithms when consensus is repeatedly needed, say k times. We show that it is possible to achieve consensus with the optimal number of processors ($n > 3t$), and when k is large enough, optimal amortized cost in all other measures: The number of communication rounds r^*, the maximal message size m^*, and the total bit complexity b^*. More specifically, we achieve the following amortized bounds for k consensus instances: $r^* = O(1 + \frac{t}{k})$, $b^* = O(nt + \frac{nt^3}{k})$, and $m^* = O(1 + \frac{t^2}{k})$. When $k \geq t^2$, then r^* and m^* are $O(1)$ and $b^* = O(nt)$, which is optimal.

*A preliminary version of this paper was presented at the *5th International Workshop on Distributed Algorithms*, Delphi, Greece, Sept. 1991.

[†]IBM T. J. Watson Research Center, P. O. Box 704, Yorktown Heights, NY 10598. E-mail: amotz@watson.ibm.com.

[‡]Department of Computer Science, York University, North York, Ontario, Canada M3J 1P3. deng@cs.yorku.ca. This work was carried out while this author was at the School of Computing Science, Simon Fraser University.

[§]IBM T. J. Watson Research Center, P. O. Box 704, Yorktown Heights, NY 10598. garay@watson.ibm.com.

[¶]School of Computing Science, Simon Fraser University, Burnaby, B.C., Canada V5A 1S6. tiko@cs.sfu.ca.

1 Introduction

A major task in fault-tolerant distributed systems is to agree on a common value. The *Consensus* problem (and its variations, e.g., the *Byzantine Generals* problem) [PSL] is related to this task and has received much attention in the literature. The problem is defined as follows. Given is a set P of n processors, a subset T ($|T| = t$) of which may be *faulty*. Each processor starts off with an initial binary value. The goal is to find an algorithm where each *non-faulty* (*correct*) processor *decides* on a 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 is the decision value.

We assume the standard distributed model: Processors are synchronized by rounds of communication in which each sends messages, receives messages, and performs some local computation. By the last round of the algorithm, after the local computation is completed, the (correct) processors must decide. The communication is carried out by a complete and reliable network, i.e., 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 and no randomized operations are allowed. The faulty processors might behave in an arbitrary (*Byzantine*) way, and even collude in order to prevent reaching a valid agreement. For ease of the exposition, processors' names are assumed to be the numbers $1, \ldots, n$.

The efficiency of a solution to the Consensus problem is characterized by three complexity measures: The ratio between n and t; the number of rounds required in the worst case, denoted by r; and the communication complexity, alternatively given by the upper bound on the size of any message, denoted by m, or the total number of transmitted bits, denoted by b. The known lower bounds for these complexity measures are: $n \geq 3t + 1$ [PSL], $r \geq t + 1$ [FL], $m \geq 1$ (obvious), and $b \geq n(t+1)/4$ [DR]. These lower bounds are independent in the sense that they are valid no matter what the values of the other parameters are. The lower bound on the number of rounds, in particular, asserts the difficulty of overcoming potential faults.

One approach to reduce this high price for fault tolerance is that of Dolev, Reischuk and Strong [DRS]. By permitting processors to terminate at different times (rounds), instead of simultaneously, they show that *early-stopping* consensus algorithms are possible. This type of agreement is called *eventual*, and they prove a lower bound of two rounds of termination uncertainty for this case. Such algorithms take a number of rounds proportional to the actual number of failures that actually occur, rather than the worst-case t. (However, if simultaneous termination is required, then the algorithm must run for $t + 1$ even if no faults occur.)

Randomization is another way to circumvent the lower bound for the number of rounds: There exist probabilistic algorithms that reach consensus in expected *constant* number of rounds. Feldman and Micali [FM], in particular, have a probabilistic algorithm that achieves this together with the optimal number of processors.

In this paper we study whether this constant number of rounds can also be achieved on the average in situations where the same set of processors need to perform consensus repeatedly, say k times, and where it is known that at most t will fail over all. In addition, we would like to achieve this constant average number of rounds while obtaining an optimal average performance in the other parameters. We concentrate on the case in which the consensus instances have to be performed far apart, i.e., the next consensus does not start until the previous one has already finished. We call this problem the *Multi-consensus* problem.

Let $R(k)$, $M(k)$, and $B(k)$ denote the overall round complexity, message size complexity, and bit complexity over k instances of the consensus problem, respectively. (Recall that m is defined to be the maximal number of bits a message contains in a specific consensus instance, thus $M(k)$ is the sum of the ms of all k instances.) We introduce the *amortized* counterpart of the efficiency parameters: $r^* = R(k)/k$, $m^* = M(k)/k$, and $b^* = B(k)/k$, namely, the average value of the corresponding parameters over the k consensus instances.

We now establish lower bounds for these parameters. Concerning r^* and b^*, consider the scenario in which the faulty processors behave correctly during the $k-1$ first instances, and maliciously during the last one. Halpern and Hadzilacos have shown a lower bound of $\Omega(nt)$ on the number of bits for a failure-free instance of the problem [HH]. Thus, each of the first $k-1$ instances requires at least one round of communication and $\Omega(nt)$ bits. In the last instance, the faulty processors can force the execution to last for $t+1$ rounds, again using $\Omega(nt)$ bits. Consequently, $R(k) = \Omega(t+k)$, implying $r^* = \Omega(1+\frac{t}{k})$, and $B(k) = \Omega(ntk)$, implying $b^* = \Omega(nt)$. The third lower bound is obviously $m^* \geq 1$.

A related problem to Multi-consensus is when the inputs of all instances are given in advance. Then by concatenating all the input bits together it is possible to achieve $r^* = \frac{r}{k}$, $b^* = b$, and $m^* = m$, for any ordinary consensus algorithm with complexity r, b and m, in rounds, bits, and maximum message size, respectively. However, this is not a very practical solution because of the enormous size of the messages for large k. In [BD], Bar-Noy and Dolev considered solving many instances of the consensus problem in a pipeline fashion by using only single-bit messages ($m^* = 1$). Their generic algorithm achieves $r^* = 1 + \frac{t}{k}$ and $b^* = O(nt^2)$; their solution, however, has the drawback that the number of required processors is quadratic in t. Ben-Or and El-Yaniv [BE] presented a randomized algorithm for solving many consensus instances together in (expected) constant number of rounds, by generalizing the result of [FM] in a non-trivial fashion. However, as in the simple concatenation solution, they use messages the size of which is proportional to the number of instances.

An apparent main requirement to achieve the goal of optimal amortized parameters is that a constant number of rounds is invested per occurring fault. We note that prior to this paper, "technology" already existed to achieve this property, namely the simultaneous application of early stopping and *fault detection* techniques. To the best of our knowledge, the only algorithms in which these techniques are used together to achieve consensus with the optimal number of processors are those of Bar-Noy *et al.* [BDDS] and Berman and Garay [BG].[1] However, using these algorithms as the basis for solving many instances of the consensus problem yield the following parameters: $r^* = O(1 + \frac{t}{k})$, $b^* = O(nt^2 + \frac{nt^3}{k})$, and $m^* = O(t)$ (i.e., non-optimal amortized communication complexity).

The contribution of this paper is to achieve, for $n > 3t$, a constant amortized round complexity and to improve the results implied by [BDDS, BG] in both message size and bit complexity. We thus obtain an optimal amortized solution in all four parameters when k is large enough:

- $n = 3t + 1$,
- $r^* = O(1 + \frac{t}{k})$,
- $b^* = O(nt + \frac{nt^3}{k})$, and
- $m^* = O(1 + \frac{t^2}{k})$.

When $k \geq t^2$ we obtain $r^* = O(1)$, $b^* = O(nt)$, and $m^* = O(1)$. Note that when $k = o(t^2)$ the amortized bit complexity and the amortized message size complexity are not optimal. We leave it as an open problem whether an optimal solution is possible for all values of k. When $n > 4t$, we can do slightly better: $b^* = O(nt + \frac{nt^2 \log t}{k})$, and $m^* = O(1 + \frac{t \log t}{k})$.

The remainder of the paper is organized as follows. In Section 2 we describe the early-stopping *Phase King* algorithm of Berman, Garay and Perry [BGP]. In Section 3, we discuss some modifications to the *Phase King* algorithm that are needed for our Multi-consensus algorithm. Finally, the solution to the Multi-consensus problem and its complexity are presented in Section 4.

2 A Simple Early-Stopping Consensus Algorithm

In this section we present a simple consensus algorithm (*Early-Stopping Phase King*—ESPK for short) due to Berman, Garay and Perry [BGP], that uses messages of constant size and satisfies the early stopping property, i.e., it is able to terminate in a number of rounds

[1] In fact, the algorithm of [BDDS] uses only the fault detection technique. However, the early stopping technique can be readily applied.

```
Procedure ESPK (var V: {0,1});
begin
    for i := 1 to t + 1 do
    begin

            (* universal exchange 1 *)
        broadcast(V);
        for j := 0 to 1 do
            C[j] := # of received j's;

            (* universal exchange 2 *)
        for j := 0 to 1 do
            if C[j] ≥ n − t then broadcast(1) else broadcast(0);
            D[j] := # of received 1's;
        if D[1] > t then V := 1 else V := 0;
        if D[V] ≥ n − t then π := 1 else π := 0;

            (* king's broadcast *)
        broadcast(⟨V, π⟩);
        S := # of received ⟨v, 1⟩'s, for any v;
        if S ≤ t
            then V := v, where ⟨v, w⟩ is received from i
        elseif S > 2t
            then return;

    end for
end;
```

Figure 1: The Early-Stopping Phase King algorithm for processor p.

proportional to the actual number of occurring faults. Let f denote the actual number of faults in a run of the algorithm. The algorithm yields consensus with $n > 3t$, $r = \min\{3(t+1), 3(f+2)\}$, $b = O(nt \cdot \min\{f+2, t+1\})$ bits, and $m = 2$.

Algorithm ESPK is shown in Figure 1. It executes at most $t+1$ phases, each consisting of three communication rounds: Two "universal exchanges," in which all processors broadcast their values, and the "king's broadcast." (The "king's broadcast" is also an universal exchange, but the reason for the name is that in this exchange there is a distinguished processor the processors will pay attention to, in case they are undecided.) Each phase has a distinct, predetermined king. We assume that each processor p has the following local variables: V (the subject of the consensus problem); integer arrays $C[0..1]$ and $D[0..1]$ (the counts of 0's and 1's received in universal exchanges 1 and 2, respectively); π (to indicate whether or not there is a "strong preference" for V after universal exchange 2); and S (which counts the number of processors with strong preferences). In "universal exchange 1," all processors communicate their V values (in phase 1, V contains the processor's initial value). In "universal exchange 2," each processor changes V to what it perceives as the "popular" value. It is shown below that after this exchange there is at most one value v such that $D[v]_p > t$ for any nonfaulty processor p. Each processor also sets π to true if V is strongly favored. During the "king's broadcast" each processor sends the value determined to be popular along with an indication of the strength of its preference for this value. After counting the number of processors with strong preferences, a processor replaces its own V with the king's value if there are too few processors with a strong preference. Should the processor find many processors with a strong preference, it halts.

Remark 2.1 Because some correct processors may halt before others, a convention is adopted to account for messages that are not received during a phase: If no message is received from p during the universal exchange, the receiving processor assumes that the missing value is equal to the value of its own V of the previous phase, and that the silent processor indicates a strong preference during the king's broadcast.

Notice that the algorithm has two exit points; in either case, the value of V is returned. We also assume that the non-faulty processors ignore any spurious or extra messages that may be sent by the faulty processors.

We now address the correctness of the algorithm.

Theorem 2.1 [BGP] *Distributed Consensus is achievable using* $\min\{3(f+2), 3(t+1)\}$ *exchange rounds,* $O(nt \cdot \min\{f+2, t+1\})$ *bits, and messages of size at most two, or* $\min\{5(f+2), 5(t+1)\}$ *exchange rounds,* $O(nt \cdot \min\{f+2, t+1\})$ *bits, using single-bit messages, where* f *is the number of processors that fail during its execution, provided* $n > 3t$. [2]

[2]It is possible to improve the number of rounds of ESPK, while maintaining the same bit complexity and

Proof: We first establish the following three claims:

1. If $n - t$ non-faulty processors begin a phase with the same value of V, say, $v \in \{0, 1\}$, then all the non-faulty processors halt at the end of the phase with V unchanged.

   To see this, first observe that after universal exchange 1 each non-faulty processor p has $C[v]_p \geq n - t$ and $C[\overline{v}]_p \leq t < n - t$. Consequently, in universal exchange 2, each non-faulty processor sends 1 for v and 0 for $\overline{v}$, resulting in each non-faulty processor p assigning v to V_p. Since for each p $D[v]_p \geq n - t$, $\pi = 1$, and thus $S \geq n - t > 2t$, all non-faulty processors ignore the King's broadcast and execute **return** at the end of the phase.

2. If the King of phase K is non-faulty, all non-faulty processors will end phase K with the same value of V.
   There are two possibilities:

   (a) $S \leq t$ for each non-faulty processor p. Then all non-faulty processors replace V with the value received from the King. By assumption, the King is non-faulty so all non-faulty processors assign the same value to V.

   (b) $S > t$ for some non-faulty processor p. Then π is true for at least one correct processor p' so at the end of universal exchange 2, $V_{p'} = v$ and $D[v]_{p'} \geq n - t$. Note that since there are no more than t faulty processors, $|C[v]_q - C[v]_r| \leq t$ and $|D[v]_q - D[v]_r| \leq t$, for every non-faulty q, r, and $v \in \{0, 1\}$. Therefore if some nonfaulty processor q broadcasts 1 for v in universal exchange 2, then no non-faulty processor s can broadcast 1 for $\overline{v}$ (since $C[v]_q \geq n - t$ implies $C[v]_s \geq n - 2t$, $C[\overline{v}]_s$ must be at most $2t < n - t$). By the same logic, $D[v]_{p'} \geq n - t$ implies that the King of phase K also broadcasts v. This can be seen as follows. If $v = 1$, then since every other non-faulty processor q (including the king) has $D[v]_q \geq n - 2t > t$, every non-faulty q sets $V_q = 1$ in universal exchange 2. On the other hand, if $v = 0$, then some non-faulty processor sent a 1 for 0 in the **broadcast** of universal exchange 2, so no non-faulty processor sent a 1 for 1, so $D[1]_q \leq t$ holds for all non-faulty q, and therefore all set $V_q = 0$. As a result, each non-faulty processor q has $V_q = v$ at the end of phase K, either because it ignores the King's value, or because of receiving it.

   In either case, by the first claim and Remark 2.1, all non-faulty processors will halt by phase $K + 1$.

3. If the King of phase K is faulty and the first non-faulty processor p halts in phase K, then all non-faulty processors end phase K with the same value of V.

single-bit messages, to $\min\{4(f + 2), 4(t + 1)\}$ exchange rounds. See [BGP] for details.

By inspection of the code's stopping condition, p halts only if at the end of phase K, $S_p > 2t$ for some $v \in \{0,1\}$. Since at most t processors are faulty, this implies that for all non-faulty q, $S_q > t$. Inspection of the code and the reasoning from 2b imply that all non-faulty q ignore the King's broadcast and end phase K with $V_q = V_p$.
Again, claim 1 and Remark imply that all non-faulty processors will halt by phase $K+1$.

Validity follows from claim 1, since in this case all the non-faulty processors (i.e., at least $n-t$) begin phase 1 with the same value $v \in \{0,1\}$. **Agreement** follows from claim 2, and the fact that at least one out of the $t+1$ Kings is non-faulty. Notice that it is correct for the processors to terminate at the end of phase $t+1$, since in the case that each of the first t kings is faulty, then the last king is non-faulty, and they all finish the phase with the same value. Concerning the running time, first note that the main loop executes $t+1$ times at the most. If $f < t$ failures occur, claims 1-3 taken together imply that all non-faulty processors reach agreement and halt by phase $f+2$.

The message size and bit complexity follow from inspection of the algorithm. By specifying that the **broadcast** operation be performed only by processors $1, 2, \ldots, 3t+1$, the number of sender/receiver pairs is reduced from n^2 to $O(nt)$. This way, the bit complexity becomes $O(nt \cdot \min\{f+2, t+1\})$, instead of $O(n^2 \cdot \min\{f+2, t+1\})$. The correctness proof of the algorithm with this modification is straightforward.

□

In the next section we perform some modifications to ESPK in order to make it suitable for our Multi-consensus construction.

3 Modifications to the ESPK Algorithm

The first modification to ESPK is to prepare it to solve a different version of the Consensus problem, in which the input values for the correct processors are x and $x+1$, for some unknown $0 \leq x < t$. We call this problem the *Unknown Binary Set* Consensus problem (UBC for short). In addition, a stronger Validity condition is required:

- **Strong Validity:** The decision value must be one of the correct processors' initial values.

In the standard version of the problem (i.e., when the values to be decided upon are from $\{0,1\}$), this condition and the original Validity condition stated in Section 1 are equivalent. It is easy to see that a straightforward reduction of UBC to the 0-1 case fails, since the faulty processors would have the power to "introduce" the values $x-1$ and $x+2$. (In the binary version of the problem, the faulty processors cannot introduce any spurious value besides $0, 1$.)

We now show how to solve UBC with the optimal number of processors, and using messages of linear size. For completeness, we remark that when the number of processors is larger, namely, $n > 4t$, the following is possible:

Lemma 3.1 UBC *is solvable using* $n > 4t$ *processors and messages of size at most* $\log x \leq \log t$.

This result can be achieved by running the version of ESPK for $n > 4t$ presented in [BGP] in parallel on each of the bits of the binary representation of x. An initial round of universal exchange is needed, though, in order to filter out values with a low multiplicity. We omit further details, since we are interested here in solutions that use the optimal number of processors. When $3t < n \leq 4t$ we solve UBC using larger messages. In particular, we shall use the *unary* representation of the inputs. Namely, each correct processor will replace its input by x (or $x+1$) ones followed by a single zero. The need for this maneuver is motivated by the following example: Suppose that $x = \langle 1001 \rangle$, and therefore $x + 1 = \langle 1010 \rangle$. A correct outcome of running a "regular" binary algorithm on each of the bits could be, for example, $\langle 1011 \rangle$, which is neither x nor $x + 1$, thus violating the Strong Validity condition of UBC. The processors run at most $x + 3$ instances of ESPK in parallel on the new representation of their input.[3] If a correct processor's initial value is x, it participates only in the first $x + 2$ instances, where its $(x + 2)$nd input is zero. On the other hand, if its initial value is $x + 1$, then it participates in the first $x + 3$ instances and its $(x + 3)$rd input is zero. The decision value is the location of the first zero minus one. Note that a processor with initial value x must participate in the $(x + 2)$nd instance, because in the event that some correct processors have value $x + 1$, it must be ensured that all correct processors agree on 0 there.

Lemma 3.2 ESPK *with the unary representation modification solves* UCB *using* $n > 3t$ *processors and messages of size at most* $x + 3$.

Proof: On each instance the processors run the regular ESPK from the previous section and therefore decide on either zero or one. Since all correct processors have input one in the first x instances, it follows from Validity of ESPK that the first zero location is at least in the $(x + 1)$st position. On the other hand, since all correct processors have input zero in the $(x + 2)$nd instance, it again follows that the first zero location is at most in the $(x + 2)$nd position. Consequently, all correct processors either find the first zero at location $x + 1$ or at location $x + 2$, and therefore decide on either x or $x + 1$. This gives Agreement. Strong Validity also follows from the above reasoning, since if all correct processors start with the same value—same number of 1's, ESPK guarantees that this will be the final value. Clearly

[3]It is easy to see how to achieve consensus on more than one value in parallel. This is achievable in the same number of rounds as for one value, at the cost of a multiplicative increase in communication.

the largest message is of size $x+3$, where $x \leq t$. □

The second modification to ESPK is the addition of a *fault detection* mechanism. One additional observation about ESPK is that only faulty kings can cause the algorithm to run for many rounds. Moreover, doing this forces them to reveal themselves as faulty to all the correct processors. Let F be the set of processors known as faulty by all the correct processors. (More technically, $F \stackrel{\text{def}}{=} \bigcap_{p \in P-T} F_p$, where F_p is the set of processors detected as faulty by correct processor p.) We say a processor in F has been *globally* detected. Assume now that correct processor p terminates the execution of ESPK at phase i. We claim that in this case the set of processors $\{1, \ldots, i-3\} \subseteq F$. This is true because whenever a processor terminates phase $i-1$ and has to proceed the phase i, it realizes that the previous king, $i-2$, must have been faulty. Since no correct processor could terminate before phase $i-1$, it follows that all correct processors detected at least $i-3$ faulty kings. Note, however, that if a processor terminates in phase i, then it knows the king of phase $i-2$ is faulty as well.

Our third and last modification to ESPK concerns the sequence of kings that the algorithm uses. We note that the ascending order of kings used in Figure 1 is not necessary; any set of $t+1$ kings (provided they are known to all the correct processors) would yield the correctness of the algorithm.

We call ESPK augmented with the three modifications described above MOD-ESPK. Pseudocode for the algorithm is presented in Figure 2. In the code, V is a binary vector of length $x+1$. The first bit of the vector, V_0, is the input to the current consensus instance (while the rest of the vector is the unary representation of the number of faults detected by the previous instance of MOD-ESPK, as will be explained in the next section). The variables fk and lk are the identities of the first and the last processors to be used as kings in this instance, respectively. In addition to the decision value, MOD-ESPK also outputs the number of processors detected as faulty during the current invocation; that is the purpose of variable f. The algorithm imitates ESPK on each one of the $x+1$ components of the vector V. To control these $x+1$ parallel runs, the algorithm maintains a vector *decide* of length $x+1$, and terminates when each of this vector's components is *true* meaning that ESPK terminated on each of the $x+1$ binary values. The rest of the code is similar to the code of algorithm ESPK, where all the operations are performed on components of binary vectors instead of binary values. The halting decision reflects the fact that if processor p terminates in phase i, then p knows that processors $fk, \ldots, i-2$ are faulty. Therefore, p detects $i-2-(fk-1) = i-1-fk$ faulty processors. For simplicity, a processor that has already halted on some component(s) of the vector V continues to broadcast the whole vector. This is fine since, as dictated by the code (variable $decide_\ell$), processors that have already decided on these components will ignore them. In the next section, we use MOD-ESPK as the building block of our solution to the Multi-consensus problem.

```
Procedure MOD-ESPK ( var V : {0,1}^{x+1};  var f : {0,...,t};  fk, lk : {1,...,n} );
begin
    for ℓ := 0 to x do decide_ℓ := false;
    for i := fk to lk do
    begin

            (* universal exchange 1 *)
        broadcast(V);
        for j := 0 to 1 do
            for ℓ := 0 to x do if decide_ℓ = false then
                C_ℓ[j] := # of received j's in the ℓth component;

            (* universal exchange 2 *)
        for j := 0 to 1 do
            for ℓ := 0 to x do if decide_ℓ = false then
                if C_ℓ[j] ≥ n − t then M_ℓ[j] := 1 else M_ℓ[j] := 0;
        for j := 0 to 1 do
            broadcast(M);
            for ℓ := 0 to x do if decide_ℓ = false then
                D_ℓ[j] := # of received 1's in the ℓth component;
        for ℓ := 0 to x do if decide_ℓ = false then
            if D_ℓ[1] > t then V_ℓ := 1 else V_ℓ := 0;
            if D_ℓ[V_ℓ] ≥ n − t then π_ℓ := 1 else π_ℓ := 0;

            (* king's broadcast *)
        broadcast (⟨V_0, π_0⟩, ..., ⟨V_x, π_x⟩);
        for ℓ := 0 to x do if decide_ℓ = false then
            S_ℓ := # of received ⟨v, 1⟩'s, in the ℓth component;
            if S_ℓ ≤ t then
                V_ℓ := v, where ⟨v, w⟩_ℓ is the ℓth component of the message
                received from i
            elseif S_ℓ > 2t then decide_ℓ := true;

            (* halting decision *)
        if for all 0 ≤ ℓ ≤ x decide_ℓ = true then
            f := max{i − 1 − fk, 0};      (* # of detected faulty processors *)
            return
    end for
end;
```

Figure 2: The Modified Early-Stopping Phase King algorithm for processor p.

4 Optimal Amortized Consensus

In this section we describe an algorithm for k consensus instances, which for k large enough achieves optimal amortized parameters. In this paper we assume that there is enough time to finish each instance before the next one starts. We also assume that all the starts of all instances are synchronized. Our goal is to use for each consensus instance an early-stopping algorithm satisfying the following two properties:

1. Its round complexity is $r = \beta_r + \alpha_r f$, where α_r and β_r are constants and f is the number of faulty processors that actually occurred in the run and these faulty processors were globally detected. Since in later consensus instances at least f faulty processors can be ignored,[4] the round complexity of the Multi-consensus algorithm is at most $R(k) = \beta_r k + \alpha_r t$, and hence $r^* \leq \beta_r + \alpha_r \frac{t}{k}$. When $k \geq t$, we have $r^* \leq \beta_r + \alpha_r = O(1)$.

2. Its bit complexity is $b = \beta_b nt + \alpha_b nf$, where α_b and β_b are constants. Again, since the f globally detected faults will be ignored in later consensus instances, it follows that the bit complexity of the Multi-consensus algorithm is at most $B(k) = \beta_b ntk + \alpha_b nt$, and hence $b^* \leq \beta_b nt + \alpha_b \frac{nt}{k}$. Thus, $b^* = O(nt)$ for all k.

The consensus algorithms of Bar-Noy *et al.* [BDDS] and Berman and Garay [BG] achieve the first property but not the second: The bit complexity for globally detecting f faulty processors is $\Omega(fnt^2)$. On the other hand, algorithm ESPK, which always uses processors $1, \ldots, t+1$ as kings, achieves the two properties for a non-constant $\alpha_b = O(t)$. However, there is a problem in using this algorithm straightforwardly. Implicit in ESPK is the assumption that the processors know in advance the sequence of kings to be used during the run of the algorithm. When the number of consensus instances is only one or two, this can be easily achieved by having all processors agree *a priori* on the sequences of kings. For example, the first instance could use kings $1, \ldots, t+1$, while the second instance could use kings $n-t, \ldots, n$. However, what would the sequence of kings for the third consensus instance be? The main problem in obtaining a common agreement among the correct processors about the next set of kings is the fact that the consensus instances we are talking about here are of the *eventual* type [DRS], and therefore the processors cannot immediately agree on what the last round in the previous instance was. On the other hand, if faulty kings were to be "re-used," then it is easy to see that the desired amortized complexity would not be achieved.

To overcome this problem we will use the following mechanism. In even consensus instances the correct processors will use processors from $\{1, \ldots, t+1\}$ as kings, while in odd consensus instances they will use processors from $\{n-t, \ldots, n\}$. (Thus, the sets of kings will be alternating

[4]For example, by using a *fault masking* technique (e.g., [BDDS, DRS]), or, in the case of ESPK, by "skipping" kings globally known as faulty.

between the "lower" and "upper" regions of the names' domain.) In addition, during the jth consensus instance the processors will also agree on the first king to be used in the $(j+1)$st consensus. Initially, in the first two instances, the starting kings will be 1 and $n-t$, respectively. In subsequent instances, as some processors are globally detected as faulty, the starting king could be any processor whose name is between 1 and $t+1$, or between $n-t$ and n, respectively. To obtain an agreement on the starting king, and therefore on the sequence of kings to be used, the input for each instance will consist of a pair of numbers: The consensus input bit, and the name of a would-be starting king for the next instance.

Such a solution implies an amortized bit complexity of $b^* = O(nt \log t)$, because the size of each message is multiplied by $\log t$. To get rid of the $\log t$ factor, instead of agreeing on the first king to be used in the next instance, the processors will agree instead on the number of globally detected faulty processors. This number can be as large as t, but t is also an upper bound for the sum of these numbers over all instances. We will show later that the overhead bit complexity for this mechanism is $O(knt + nt^3)$.

We now proceed to describe the algorithm more formally; please refer to Figure 3. In the algorithm, for each consensus instance j, processor p gets the input bit in V_0, and returns its decision before getting the next input bit. Processor p maintains two parameters, l and u, which will contain the names of the first kings in the odd and even instances, respectively. In the odd consensus instances the kings will be $l, l+1, \ldots$, while in the even consensus instances the kings will be $u, u+1, \ldots$ Let l_j and u_j denote the values of these parameters at the beginning of the jth instance; initially, $l_1 = 1$ and $u_1 = n - t$. Another parameter maintained by p is f, which is the number of *new* faulty processors detected by p after the termination of a call to MOD-ESPK. Let f_{j-1} denote the value of f at the beginning of the jth instance; initially, $f_0 = 0$. The vector of input bits for the jth instance of MOD-ESPK is V (the consensus bit) followed by the unary representation of f_{j-1}. The vector of output bits of the jth instance of MOD-ESPK contains the decision bit, followed by the number of globally detected faults, again in unary representation. At the end of an odd consensus instance j, p sets $l_{j+1} = l_j$ and $u_{j+1} = u_j + f_{j-1}$, while at the end of an even consensus instance j, p sets $l_{j+1} = l_j + f_{j-1}$ and $u_{j+1} = u_j$. The value of f_j, which is required for the next instance, is set to be f_{new}, which is the number of faults detected by processor p during the jth instance. Note that since this number is agreed upon by all processors, by induction, $f_0, f_1, \ldots$ are identical for all processors. In the sequel, we shall use them without referring to a specific processor.

The following lemma shows that before each call to MOD-ESPK, all the correct processors agree on the set of kings to be used in the next instance. The proof of this lemma follows by a straightforward induction on j.

Lemma 4.1 *At the beginning of an odd (respectively, even) instance j, all the correct processors have the same value for l_j (respectively, u_j).*

```
Algorithm MULTI-CONSENSUS;
begin
    l := 1;    u := n − t;    f := 0;
    for j := 1 to k do
    begin
        input(V_0);
        V_1 = V_2 = ··· = V_f := 1;    V_{f+1} = V_{f+2} := 0;
        if j is odd then begin
            Call MOD-ESPK(V, f_new, l, t + 1);
            (* # of globally detected faults *)
            f := min_{ℓ>0}{ℓ|V_ℓ = 0} − 1;
            (* compute identity of first king for next instance *)
            u := u + f;
            f := f_new
        end;
        if j is even then begin
            Call MOD-ESPK(V, f_new, u, n);
            (* # of globally detected faults *)
            f := min_{ℓ>0}{ℓ|V_ℓ = 0} − 1;
            (* compute identity of first king for next instance *)
            l := l + f;
            f := f_new
        end;
        output(V_0)
    end for
end;
```

Figure 3: The Multi-consensus algorithm for processor p.

The following lemma follows from the fault detection property of MOD-ESPK:

Lemma 4.2 *For all j, processors $1, \ldots, l_j - 1 \in F$ and $n-t, \ldots, u_j - 1 \in F$.*

Proof: Algorithm MOD-ESPK guarantees that at the beginning of each round j, there can be at most two values, x and $x+1$, for f_{j-1}. In the $(j+1)$st instance, the processors agree on either x or $x+1$ detected faulty processors. In any case, at least x faulty processors are globally detected. These x faulty processors are the first x kings used by the jth invocation of MOD-ESPK. Thus, if the jth instance of algorithm MOD-ESPK used kings $l_j, l_j+1, \ldots$ (or $u_j, u_j+1, \ldots$), then processors $l_j, \ldots, l_j+x-1$ (respectively, $u_j, \ldots, u_j+x-1$) are faulty processors. The proof is completed by a straightforward induction on j, where the induction step is described above. □

We are now ready to assert the amortized complexity of our Multi-consensus algorithm:

Lemma 4.3 *The round complexity of the* MULTI-CONSENSUS *algorithm is $R(k) = O(k+t)$, and therefore $r^* = O(1+\frac{t}{k})$.*

Proof: The early stopping property of MOD-ESPK implies that the jth instance takes at most $r_j = f_j + 3$ rounds. It follows from Lemmas 4.1 and 4.2 that a globally detected faulty processor can play the role of a king at most once, and therefore $\sum_{j=1}^{k} f_j \leq t$. Consequently, $\sum_{j=1}^{k} r_j = O(k+t)$. □

Lemma 4.4 *The bit complexity of the* MULTI-CONSENSUS *algorithm is $B(k) = O(ntk+nt^3)$, and therefore $b^* = O(nt + \frac{nt^3}{k})$.*

Proof: Recall that the input to MOD-ESPK is a pair of numbers: The first is the input bit and the second is a number f, $0 \leq f \leq t$. First we analyze the bit complexity of MULTI-CONSENSUS related to the first component of each such pair. In algorithm MOD-ESPK the bit complexity of each phase is $O(nt)$. The early stopping property of MOD-ESPK implies that the jth instance took at most f_j+2 phases. Therefore, the bit complexity of the jth instance is $b_j^{'} = O(ntf_j + nt)$. It follows from Lemmas 4.1 and 4.2 that a globally detected faulty processor can play the role of a king at most once, and therefore $\sum_{j=1}^{k} f_j \leq t$. Consequently,

$$\sum_{j=1}^{k} b_j^{'} \leq O(nt^2 + ntk).$$

We now analyze the bit complexity related to the second component of each input pair. This bit complexity is the same as that of the first part for all cases when $f_j = 0$. Let $i_1, \ldots, i_h$ be the instances for which $f_{i_g} \geq 1$. The agreement on f_{i_g} is actually done in instance $i_g + 1$. Since instance i_g+1 takes $f_{i_g+1}+2$ rounds each with messages of length $3+f_{i_g}$, it follows that the bit complexity is at most $b_{i_g+1}^{''} = O(nt(3+f_{i_g})(f_{i_g+1}+2)$. Thus, the bit complexity for the cases where $f_j > 0$ is at most

$$\sum_{g=1}^{h} b''_{i_g+1} \leq \sum_{g=1}^{h} cnt f_{i_g+1} f_{i_g} \leq cnt^2 \sum_{g=1}^{h} f_{i_g+1} \leq cnt^3,$$

for some constant c. The above inequalities rely on Lemma 3.2. □

We note that if $n > 4t$, then we can use Lemma 3.1 instead of Lemma 3.2 for the last inequality to obtain a bit complexity $b^* = O(nt + \frac{nt^2 \log t}{k})$.

Lemma 4.5 *The average message size of* MULTI-CONSENSUS *is* $m^* = O(1 + \frac{t^2}{k})$.

Proof: Using the same reasoning as in Lemma 4.4, we obtain that in most instances the messages are of constant size, whereas in at most t instances the messages are of size greater than 3, and in those instances, the messages are of size $O(t)$. Thus $M(k) = k \cdot O(1) + t \cdot O(t)$, which in turn implies $m^* = O(\frac{k+t^2}{k})$. □

Again, note that if $n > 4t$ we get $m^* = O(1 + \frac{t \log t}{k})$.

We are now ready to state our main result. Its correctness follows from Lemmas 4.1 and 4.2, while Lemmas 4.3, 4.4 and 4.5 yield the complexity measures.

Theorem 4.1 *The* MULTI-CONSENSUS *algorithm is correct for all k consensus instances, provided $n > 3t$, and has the following amortized complexity:*

1. $r^* = O(1 + \frac{t}{k})$,
2. $b^* = O(nt + \frac{nt^3}{k})$, *and*
3. $m^* = O(1 + \frac{t^2}{k})$.

Acknowledgments

The authors would like to thank Vassos Hadzilacos for pointing out a flaw in an early version of the Multi-consensus algorithm. The third author would also like to thank Ken Perry for discussions on multi-valued agreement. Last but not least, the authors specially thank the two referees for their thorough readings and many useful suggestions, which greatly improved both the presentation and the accuracy of the paper. Work of the second and fourth authors was partially supported by the NSERC (Natural Sciences and Engineering Research Council of Canada) International Postdoctoral Fellowship, and research grants from the NSERC and the Advanced Systems Institute of British Columbia.

References

[BD] A. Bar-Noy and D. Dolev, "Consensus Algorithms with One-Bit Messages," *Distributed Computing,* Vol 4, pp. 105-110, 1991.

[BDDS] A. Bar-Noy, D. Dolev, C. Dwork, and H. R. Strong, "Shifting Gears: Changing Algorithms on the Fly to Expedite Byzantine Agreement," *Information and Computation,* Vol 97, pp.205-233, 1992.

[BE] M. Ben-Or and R. El-Yaniv, "Interactive Consistency in Constant Time," unpublished manuscript.

[BG] P. Berman and J.A. Garay, "Efficient Distributed Consensus with $n = (3+\varepsilon)t$ Processors," *Proc. 5th International Workshop on Distributed Algorithms*, LNCS Vol. 579, pp. 129-141, 1991.

[BGP] P. Berman, J.A. Garay, and K.J. Perry, "Optimal Early Stopping in Distributed Consensus," *Proc. 6th International Workshop on Distributed Algorithms*, LNCS Vol. 647, pp. 221-237, 1992.

[DR] D. Dolev and R. Reischuk, "Bounds of Information Exchange for Byzantine Agreement," *Journal of the ACM*, Vol. 32, pp. 191-204, 1985.

[DRS] D. Dolev, R. Reischuk, and H.R. Strong, "Early Stopping in Byzantine Agreement," *Journal of the ACM*, Vol. 37, pp. 720-741, 1990.

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

[FM] P. Feldman, and S. Micali, "Optimal Algorithms for Byzantine Agreement," *Proc. 20th Symposium on Theory of Computing*, pp. 148-161, 1988.

[HH] V. Hadzilacos and J. Halpern, "Message-Optimal Protocols for Byzantine Agreement," *Mathematical Systems Theory*, Special Issue on Fault-Tolerant Distributed Algorithms (ed. Ray Strong), Vol. 26, No. 1, pp. 41-102, 1993. Preliminary version appeared in *Proc. 10th Annual Symp. on Principles of Distributed Computing*, pp. 309-324, 1991.

[PSL] M. Pease, R. Shostak, and L. Lamport, "Reaching Agreement in the Presence of Faults," *Journal of the ACM*, Vol. 27, pp. 228-234, 1980.