Gap Theorems for Distributed Computation *

Revised Version, December 1991

Shlomo Moran **
Department of Computer Science, the Technion,
Haifa 32000, Israel.

Manfred K. Warmuth ***
Department of Computer and Information Sciences,
University of California,
Santa Cruz, CA 95064.

* A preliminary version of this paper appeared in [MW86].

** Part of this work was done while this author was at IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598.

***This research was supported by Faculty Research funds granted by the University of California, Santa Cruz, and by ONR grants N00014-86-K-0454 and N00014-91-J-1162.

KEY WORDS:

Distributed algorithms, networks, asynchronous, message complexity, bit complexity, ring of processors, lower bounds, gap theorem.

ABSTRACT:

Consider a bidirectional ring of n identical processors that communicate asynchronously. The processors have no identifiers and hence the ring is called anonymous. Each processor receives an input letter, and the ring is to compute a function of the circular input string. If the function value is constant for all input strings, then the processors do not need to send any messages. On the other hand, we prove that any deterministic algorithm that computes any non-constant function for anonymous rings requires $\Omega(n \log n)$ bits of communication for some input string. We also exhibit non-constant functions that require $O(n \log n)$ bits of communication for every input string. The same gap for the bit complexity of non-constant functions remains even if the processors have distinct identifier, provided that the identifiers are taken from a large enough domain.

When the communication is measured in messages rather than bits, the results change. We present a non-constant function that can be computed with $O(n \log^* n)$ messages on an anonymous ring.

1. INTRODUCTION

There has been an extensive amount of research on studying computation on a ring of n asynchronous processors. The ring topology is in a sense the simplest distributed network that produces many typical phenomena of distributed computation. In this model the processors may communicate by sending messages along the links of the ring, which are either unidirectional or bidirectional. All the messages sent reach their targets after a finite, but unpredictable and unbounded delay. Numerous algorithms [ASW88, DKR82, P82] have been found for this *asynchronous ring* model. All these algorithms require the transmission of $\Omega(n \log n)$ bits. This is not surprising in view of the results of this paper. We establish a gap theorem for asynchronous distributed computation on the ring which says that either the function computed is constant and no messages need to be sent, or the function is non-constant and there is an input string requiring $\Omega(n \log n)$ bits.

In our model there is no leader among the processors. All processors receive a letter from some input alphabet as their input and run the same deterministic program which may depend on the ring size. We first treat the case where the ring is unidirectional and the processors have no identifiers (the anonymous model of [ASW88]). Then we show that the lower bound holds also for bidirectional anonymous rings, and for rings of processors with distinct identifiers, provided the set of possible identifiers is sufficiently large. Note that in the anonymous ring without a leader it is necessary that the processors "know" some bound on the ring size. Otherwise the processors cannot determine when to terminate [ASW88].

Let us contrast the anonymous ring with the model of a ring with a leader (which is also the initiator of the algorithm). If we assume unidirectional communication then any non-constant function that depends on the input of any processor other than the input of leader requires $\Omega(n)$ bits. In the bidirectional case, there are simple non-constant functions for any bit complexity In the bidirectional case, there are simple non-constant functions for any bit complexity $\Theta(b(n))$. If $b(n) \geq n^2$, such a function is achieved using non-constant size input alphabet. We now show how to achieve such a function with $b(n) < n^2$. In this case the input letters are bits. The input to the ring is denoted as an n-bit cyclic string ω, the i-th bit ω_i of ω being the input letter of the i-th processor. Using a crossing sequence argument on pairs of links one can show that the following function is a non-constant function of bit complexity $\Theta(b(n))$: $f(\omega)$=1 if and only if ω contains a palindrome of $2\left\lceil \sqrt{b(n)} \right\rceil$ +1 bits centered at the leader (the same function was first described in [MZ87], where it was used for a similar purpose). Thus there is no gap for rings with a leader. Our gap theorem for anonymous rings clearly quantifies the price one has to pay for having no distinguished processor.

In the model where there is a leader but the size of the ring is unknown, a different sort of "gap" has been found by Mansour and Zaks [MZ87]. They showed that a language is accepted in this model in $O(n)$ bit complexity if and only if it is regular, and every non-regular language requires $\Omega(n \log n)$ bits. These results of [MZ87] are analogous to the classical results for one-tape Turing machines [T64, H68]. Note that the bit complexity of non-regular languages coincides with the bit complexity required for computing the size of the ring. As discussed above, if the size of the ring is known, then for any bit complexity, there is a language with that complexity on rings with a leader.

The lower bound of $\Omega(n\log n)$ for anonymous rings proven here does not hold if one counts *messages* (of arbitrary length) instead of bits. In fact, we show in this paper that if the input alphabet is of size at least n, then there are simple non-constant functions that can be computed in $O(n)$ messages, for arbitrary ring size. Considering the message complexity is more interesting in the case when the input alphabet is of constant size, independent of the ring size n. In [ASW88] a non-constant function was presented that is computable in $O(n)$ messages on an anonymous ring when the inputs are bits. However, this function is only defined for rings of odd size. It is easy to find similar functions for rings whose sizes are not divisible by some fixed integer. When the smallest non-divisor of the ring size is large, the ring contains more inherent symmetry, and it is hard to find non-constant functions of low message complexity. We exhibit a non-constant function with binary inputs for arbitrary ring size that can be computed[1] in $O(n\log^* n)$ messages. For some values of n, a matching lower bound of $\Omega(n\log^* n)$ was proven for computing any non-constant function [DG87]. The lower bound assumes that the input alphabet is finite.

Our lower bound proofs rely heavily on the asynchronous nature of the computation. We use the fact that the result of a computation must be independent of the specific delay times of the links, and impose certain delays on the links that assure that $\Omega(n\log n)$ bits are sent. In contrast, on synchronous anonymous rings, the Boolean *AND* can be computed with $O(n)$ bits [ASW88], and $\Omega(n)$ is also a trivial lower bound for an arbitrary non-constant function.

Our research opens many challenging problems concerning gap theorems in distributed computation. Given an asynchronous network of anonymous processors, define the *distributed bit (message) complexity* of the network to be the smallest bit (message) complexity of a non-constant function when computed on that network. Intuitively, this complexity measures the minimum effort needed to coordinate the processors of the network in any sensible way[2]. This coordination should be more difficult if the network is highly symmetric. What parameters of the network correspond to this complexity? How does this complexity depend on the connectivity, diameter, or other properties of the network? Our results show that the distributed bit complexity of a ring of n processors is $\Theta(n\log n)$. The distributed bit complexity of the torus was recently shown to be linear in the number of processors [BB89].

In the next section we summarize the model of computation. We first prove the $\Omega(n\log n)$ lower bound on the bit complexity for unidirectional rings with no identifiers (Section 3), and then we generalize the result to bidirectional rings (Section 4) and to rings with identifiers (Section 5). In the last section we show that for each n, there are non-constant functions that can be computed on a ring of n anonymous processors by algorithms of $O(n\log n)$ bit complexity and $O(n\log^* n)$ message complexity. The latter function is defined by interleaving

---

[1]The function $\log^* n$ grows very slowly (≤ 5 for $n \leq 2^{65536}$). It is the number of iterations of the function $\log_2$ required to get the value n down to 1 or below.

[2]This complexity might also depend on the size of the input alphabet over which the functions are defined.

de Bruijn sequences of various lengths. We also show there that if we allow the alphabet size to be $\Omega(n)$, then there is a non-constant function (defined uniformly for all ring sizes) of linear message complexity.

The Gap Theorem with distinct identifiers assumes that the identifiers are chosen from a set of double exponential size. Similar theorems for small identifier sets are considered in a subsequent paper [BMW91]. Also, Gap Theorems for probabilistic models have been recently shown in [AAHK89].

2. THE MODEL

Our computational model is a ring R of n processors, $p_1, p_2, \cdots, p_n$. The processors are *anonymous*, i.e., they do not have identifiers, and they run the same deterministic program which may depend on the ring size. In particular, the program of p_i does not depend on its index i. Consecutive processors (as well as the last processor p_n and the first processor p_1) are connected by communication *links* and thus each processor has two *neighbors*. A processor can distinguish between its two neighbors, one is called the *left* and the other one the *right* neighbor. An *orientation* of the ring is a particular assignment of left and right to the links of each processor. If the notions of left and right of all processors are consistent then we say the ring is *oriented*. In the *bidirectional* ring connected processors can send and receive messages to and from both neighbors, respectively. In the *unidirectional* ring we assume that the ring is oriented and that messages can only be sent to the right and received from the left, that is, messages can only travel in one direction around the ring.

We assume that messages are encoded as non-empty bit strings. The messages sent along a fixed direction of a link arrive in the order in which they were sent. The communication is *asynchronous* meaning that messages arrive with an unpredictable but finite delay time. We assume that any non-empty subset of the processors may wake up spontaneously and start running the algorithm. Processors that do not wake up spontaneously are awakened upon receiving a message from a neighbor.

In our model, the input of each processor is a letter of an arbitrary alphabet I, i.e., the functions have the domain I^n. We shall assume that I contains the letter 0. An *execution* of an algorithm on a unidirectional or bidirectional ring consists of (i) an input assignment to each processor, (ii) in the case of a bidirectional ring an orientation of the ring, and (iii) a combined schedule of how the algorithm is run at each processor. The schedule includes wake up times for each processor, the exact times for each step taken by the individual runs of the algorithm and times for when each message was sent and received. The schedule has to satisfy the requirements of asynchronous ring computations, as described above.

An algorithm *computes* a function f (which is defined for a particular ring size[3] n) if for every input string ω in I^n and each execution on the ring labeled with ω every processor

[3]In Section 6 (the last section) and only in that section we present algorithms for computing non-constant functions that are defined for more than one ring size. In that case we give the algorithm the ring size as an argument.

outputs $f(\omega)$. For any non-constant function f there is a string $\omega \in I^n$ such that $f(\omega) \neq f(0^n)$. For any algorithm that computes f we say for convenience that if the input is ω (respectively 0^n) then all processors running *AL* reach an *accepting* (respectively *rejecting*) state.

Note that functions computed on a ring without a leader must be invariant under circular shifts of the input string and in the case of (unoriented) bidirectional ring also under reversal of the input string [ASW88]. The *bit* (*message*) *complexity* of an algorithm is the maximal number of bits (messages) sent over all possible executions of this algorithm. The bit (message) complexity of a function f is the minimal possible bit (message) complexity of any algorithm that computes f. The bit (message) complexity of a given network is the minimal possible bit (message) complexity of any non-constant function when computed on that network. The lower bound we prove on the bit complexity of bidirectional rings also holds when the bidirectional rings are oriented. All algorithms presented in this paper are for unidirectional rings. We discuss how they can be converted to algorithms of similar bit and message complexities that work on unoriented bidirectional rings.

Proof method for the lower bounds: We consider an arbitrary algorithm *AL* that computes a non-constant function on a ring of size n, and we construct executions of *AL* on lines of processor of various lengths. By cutting and pasting such executions we get an execution of *AL* on a ring of size n in which at least $cn \log n$ bits are sent, for some constant c independent of n. We first present the lower bound for the unidirectional case since it helps understand the more complicated bidirectional case.

3. THE LOWER BOUND FOR UNIDIRECTIONAL RINGS

Theorem 1: The bit complexity of a unidirectional ring of n anonymous processors is $\Omega(n \log n)$.

The proof of Theorem 1 will follow from some lemmas, given below. We will give lower bounds on the worst case complexity of any algorithm *AL* that accepts some string $\omega \in I^n$ and rejects 0^n. The first lemma (which holds for both the unidirectional and bidirectional case) is similar to Theorem 5.1 of [ASW88]. We repeat its proof here, to make the paper more self contained. Since the function value is independent of the delay times of the asynchronous computation, we may choose any delay times for the proofs: we assume that the ring is oriented, all processors start at time zero, internal computation at a processor takes no time and links are either *blocked* (very large delay) or are *synchronized* (it takes exactly one time unit to traverse the link). For the lemma below we assume that all links are synchronized. Intuitively, this keeps the execution symmetric and causes the most messages to be sent.

Lemma 1: Let *AL* be an algorithm for a unidirectional or bidirectional ring of n processors. If *AL* rejects 0^n and accepts $0^z \cdot \tau$ for some τ, then *AL* requires at least $n \lceil z/2 \rceil$ messages on input 0^n.

Proof: Consider the synchronized execution of *AL* with input string 0^n. Since all processors have the same input letter and run the same algorithm, at any given time all the processors are in the same state of the algorithm. Hence, at least one message is sent by each processor at each integral time until some time T at which no message is sent. From then on the processor are inactive because no new messages are received. Thus the processors terminate at time T after sending at least nT messages altogether. Now consider a second execution with input string $0^z \cdot \tau$. If $T < \lceil z/2 \rceil$ then the processor $p_{\lceil z/2 \rceil}$ is in the same state of the algorithm at time step T in both executions. Thus in both cases $p_{\lceil z/2 \rceil}$ terminates with the same result which is a contradiction. We conclude that at least $n \lceil z/2 \rceil$ messages are sent in the execution with 0^n as the input string. □

Consider an execution of the algorithm AL on ring R with ω as input string and all links synchronized. The sequence of messages sent by an anonymous processor in such an execution is uniquely determined[4] by its input letter and the sequence of message received by the processor which we call the ''history'' of the processor. Suppose all processors terminate before some time t. For $0 \le s \le t$ and for $1 \le i \le n$, we define the *history* $h_i(s)$ as the string $m_i(1)L \cdots Lm_i(r_s)$, where L is a separating symbol, and $m_i(1),..,m_i(r_s)$ are the messages (non-empty bit strings) received by p_i until (and including) time s, in chronological order. Note that r_s might be smaller than s. $H_i = h_i(t)$ is the *total history* of p_i in this execution. Since the messages are non-empty bit strings, the total length of H_i is less than twice the number of bits received by p_i. Thus, a lower bound on the bit complexity of AL is implied by a lower bound on the sum of the lengths of the total histories of the processors in a certain execution of AL. The lower bound on the sum of the lengths will follow from the fact that during a certain execution of AL the number of distinct histories of the processors is $\Omega(n)$, and therefore the lemma below implies the $\Omega(n \log n)$ bound.

Lemma 2: Let $H_1, \cdots, H_l$ be l distinct strings over an alphabet of size $r > 1$. Then $|H_1| + |H_2| + \cdots + |H_l| \ge (l/2)\log_r(l/2)$.

Proof: Represent the H_i with an ordered r-ary tree, such that each H_i corresponds to a path from the root to an internal node or a leaf of the tree. In the tree each leaf is responsible for some H_i. Assume the overall length of the strings H_i is minimized and the leaves at the lowest level of the tree are as far to the left as possible. Then in the corresponding tree each leaf and each internal node is responsible for some H_i. Furthermore, all but possibly one internal node have out-degree r and hence at least half of all the nodes are leaves. The lemma is implied by the fact that the average height of the leaves in an r-ary tree with v leaves is at least $\log_r v$. □

Outline of the proof of Theorem 1: An execution of AL is constructed for which either Lemma 1 or Lemma 2 implies the lower bound of $\Omega(n \log n)$ bits. In the first case a processor accepts an input string that contains $\log n$ consecutive zeros. Thus Lemma 1 implies that $\Omega(n \log n)$ messages are required for input 0^n. In the second case there will be an execution with more than $n - \log n$ processors with distinct histories and Lemma 2 gives an $\Omega(n \log n)$ bits lower bound.

---

[4] This does not hold for arbitrary executions. In this paper we only need to consider histories of specific executions.

Let k be an integer such that all the processors have terminated before time $t = kn$ in the synchronized execution of AL on ω, and let C be a line of kn processors, denoted by $p_{1,1}, p_{2,1}, .., p_{n,1}, p_{1,2}, .., p_{n,k}$. Informally, C consists of k copies of the ring R of n processors that were cut at the link $p_n - p_1$ and then concatenated to form one line of kn processors. Thus, processor $p_{i,j}$ in C corresponds to the processor p_i in the jth copy of R. We make C a ring by connecting $p_{n,k}$ with $p_{1,1}$ by a link which is blocked. Note that even though every processor in C acts as if it is on a ring, the block on the link $p_{n,k} - p_{1,1}$ makes the global behavior of C to be that of a line of processors.

Let ω^k be the input string to C, where $p_{i,j}$ receives the input letter ω_i, and consider the execution of AL on C in which all links are synchronized except for the block on the link $p_{n,k} - p_{1,1}$. For $0 \le s \le t$, the histories $h_{i,j}(s)$ and $H_{i,j}$ of the processor $p_{i,j}$ in C (with input string ω^k) are defined similarly to the histories of the processors in R (with input ω) given above. Recall that all processors of R terminate at time $t-1$ or before. Using an argument similar to the "shifting scenario" argument of [FLM85] we show that p_n and $p_{n,k}$ act alike.

Lemma 3: Processor $p_{n,k}$ in C accepts.

Proof: Recall that C consists of k identical copies of R. Assume for a moment that there is no block on the link $p_{n,k} - p_{1,1}$. It is easy to see that in that case $h_{i,j}(s) = h_i(s)$, for all i, j and for $0 \le s \le t$. If we now restore the block on $p_{n,k} - p_{1,1}$, then by time s, the block can only effect the s leftmost processors. Thus at time $t-1$ processor $p_{n,k}$ has exactly the same history that p_n has at time $t-1$ and $p_{n,k}$ accepts because p_n does so. □

Next, we define a subsequence $\overline{C}$ of C such that all of its processors have distinct histories at the end of the execution. First we use C to construct a directed graph, G, and then construct $\overline{C}$ from G.

The vertices of G are the processors of C, and there is a directed edge from p to q if q is the rightmost processor having the same history as the processor to the right of p. It is easy to see that there is exactly one edge leaving every processor except the last processor, $p_{n,k}$, and that G contains no cycles. Thus, G is a directed tree rooted at $p_{n,k}$. $\overline{C}$ is now the sequence of processors on the unique path that starts at $p_{1,1}$ and ends at $p_{n,k}$.

Lemma 4: No two processors of $\overline{C}$ have the same history in the execution on C.

Proof: The first processor in $\overline{C}$, $p_{1,1}$, is the only processor that receives no messages during the execution described above, due to the block on the link entering it. For the other processors the lemma follows from the fact that for each history H there is at most one rightmost processor p in C with $H_p = H$. □

Let the sequence of processors $\overline{C}$ defined above be $\overline{C} = (p_{i_1, j_1}, \cdots, p_{i_m, j_m})$, and let τ be the input string $\omega_{i_1} \cdots \omega_{i_m}$ of length m. We now run AL on $\overline{C}$ with all links synchronized except for the link $p_{i_m, j_m} - p_{i_1, j_1}$ ($= p_{n,k} - p_{1,1}$) which is blocked.

Lemma 5: In the execution of AL on $\overline{C}$ with input string τ, the history of processor p_{i_l, j_l}

($1 \le l \le m$) of $\overline{C}$ is the same as the history of p_{i_l, j_l} in the corresponding execution of AL on C with input string ω^k. In particular, processor p_{i_m, j_m} ($= p_{n,k}$) of $\overline{C}$ accepts.

Proof: This follows by a "cut and paste" argument, using the way $\overline{C}$ is constructed from C: $p_{1,1}$ ($= p_{i_1, j_1}$) clearly has the same history in both executions, and the history of any other processor p in both executions is completely determined by its input letter and the history of the preceding processor. □

Corollary 1: For any $1 \le l \le m$, the number of bits received by l distinct processors of $\overline{C}$ in the execution described above is at least $(l/4)\log_3(l/2)$.

Proof: Let P be a set of l processors in $\overline{C}$, and let $p \in P$. By Lemma 5, the history of p in the execution of $\overline{C}$ is the same as its history in the execution of C. This implies, by Lemma 4, that no two processors in P have the same history. Thus, by Lemma 2, the sum of the lengths of the histories of the processors in P is at least $(l/2)\log_3(l/2)$. The corollary follows from the observation made earlier, that this sum is less than twice the number of bits received by these processors. □

Proof of Theorem 1: Let τ' be the first n letters of $\tau \cdot 0^n$. Let $m' = min(\{m, n\})$ (recall that $|\tau| = m$). Consider a execution of AL on R with input string τ', in which the first m' processors have exactly the same history as the first m' processors in the execution of $\overline{C}$ on τ described above, and no message sent by the remaining processors is ever received. We distinguish two cases depending on the length m of $\overline{C}$.

Case $m \le n - \log n$: By Lemma 5, τ' will be accepted by at least one processor in R. Since τ' ends with $\log n$ zeros, Lemma 1 guarantees that $\Omega(n \log n)$ messages are required for the input string 0^n and the theorem holds.

Case $m > n - \log n$: By Corollary 1, the total number of bits received by the first m' processors is $(m'/4)\log_3(m'/2)$ which is $\Omega(n \log n)$. This completes the proof of Theorem 1. □

4. THE LOWER BOUND FOR BIDIRECTIONAL RINGS

The proof of the gap theorem for bidirectional rings follows the same general outline. However, the corresponding "cut and paste" construction is more subtle, and requires a more careful analysis.

Theorem 1': The bit complexity of a bidirectional ring of n anonymous processors is $\Omega(n \log n)$. This bound holds even if the ring is oriented.

Proof: Let AL be an algorithm for an oriented bidirectional ring R of size n that accepts ω and rejects 0^n and consider a synchronized execution of AL on ω. The history of a processor p_i at time s in such a execution is a string $h_i(s) = d_i(1)m_i(1) \cdots d_i(r_s)m_i(r_s)$, where $d_i(j)$ is either R (for right) or L (for left), and the $m_i(j)$'s are the distinct messages (non-empty bit strings) received by p_i up to (and including) time s, in chronological order; $m_i(j)$ is received from direction $d_i(j)$, and when two messages arrive at the same time, we assume that the left one is received before the right one. Note that the length of H_i is at most two times larger than the number of bits received by p_i.

Assume that AL accepts ω in less than t time units, where $t{=}nk$. For each positive integer $b \le k$, define a line D_b of $2nb$ processors as follows: Let C_b be a line $(p_{1,1}, p_{2,1}, \cdots, p_{n,b})$ of nb processors, and let C'_b be the line obtained by replacing each $p_{i,j}$ in C_b by $p'_{i,j}$. D_b is constructed by connecting the last processor $p_{n,b}$ of C with the first processor $p'_{1,1}$ of C'. As before make D_b a ring by connecting $p'_{n,b}$ to $p_{1,1}$ via a blocked link.

For each $b \le k$, we construct a particular execution, E_b, of AL on D_b with input string ω^{2b}. Again internal computation of a processor takes no time and a message requires exactly one unit to traverse a link. A processor is *blocked* at time s if it receives no messages at time s or later. In execution E_b we impose the following schedule: $p_{1,1}$ and $p'_{n,b}$ are blocked at time 1, $p_{2,1}$ and $p'_{n-1,b}$ are blocked at time 2, and in general at time s $(1 \le s \le bn)$, the s leftmost and the s rightmost processors of D_b are blocked. Since $p_{1,1}$ and $p'_{n,b}$ are blocked at time 1 and since the execution is synchronized, no message sent on the link $p'_{n,b}{-}p_{1,1}$ is ever received and thus D_b acts as a line of processors.

The following stronger version of Lemma 3 states that at the end of the execution E_b, the history of the sth leftmost [rightmost] processor in D_b is equal to that of the corresponding processor in the synchronized execution of AL on R after $s-1$ time units.

Lemma 6: Let $p_{i,j}$ [$p'_{i,j}$] be the sth leftmost [rightmost] processor in D_b $(1 \le s \le bn)$, and let $h_{i,j}$ [$h'_{i,j}$] denote the history of processor $p_{i,j}$ [$p'_{i,j}$] in execution E_b. Then $h_{i,j}(t){=}h_i(s-1)$ [$h'_{i,j}(t){=}h_i(s-1)$] and in the execution E_k on D_k, both $p_{n,k}$ and $p'_{1,1}$ accept. □

Since every n consecutive processors in D_b correspond to the n processors of R, Lemma 6 implies:

Corollary 2: Let R' be a set of n consecutive processors in D_b. Then the sum of the lengths of the histories in E_b is not larger than the sum of the lengths of the histories of the processors in the synchronized execution on R. □

As in the proof for the unidirectional case, we associate with the sequence D_b described above a digraph as follows: The vertices are the processors in $D_b = C_b{\cdot}C'_b$. There is an edge from each processor p in C_b to the rightmost processor in C_b which has the same history as the right neighbor of p, and there is an edge from each processor p' in C'_b to the leftmost processor in C'_b which has the same history as the left neighbor of p'. We also add an extra edge from the leftmost processor in C'_b, $p'_{1,1}$, to the rightmost processor of C_b, $p_{n,b}$. It is easily observed that this graph is a directed tree rooted at $p_{n,b}$. Let $\overline{C_b}$ be the path in that tree from $p_{1,1}$ to $p_{n,b}$, and $\overline{C'_b}$ be the path from $p_{n,b}$ to $p'_{n,b}$. $\overline{D_b}$ is the concatenation $\overline{C_b}{\cdot}\overline{C'_b}$.

Lemma 7: There is an execution $\overline{E_b}$ on $\overline{D_b}$ in which every processor has the same history as it has in the execution E_b on D_b.

Proof: Consider two processors p and q that are adjacent on $\overline{D_b}$ but not on D_b (q is to the right of p). Now let $\hat{D}$ be the line obtained from D_b by connecting p and q with a link e, and deleting all the processors between p and q. We now define an execution $\hat{E}$ on $\hat{D}$ in which the histories of all the processors in $\hat{D}$ are the same as the corresponding histories in the execution E_b on D_b.

Assume without loss of generality that in E_b, p sends a message to its right neighbor before it receives a message from it. Then the execution $\hat{E}$ is initiated by simulating the execution E_b on the processor p and the processors to its left, and delaying the computations in all other processors. Such a simulation is possible since no processor to the left of p ever receives a message from a processor to the right of p (here we use the fact that the link between $p'_{n,b}$ and $p_{1,1}$ is blocked). Continue the simulation as long as p sends messages to its right neighbor without receiving a message from it, up to a point where p receives a message from its right neighbor. At this point, continue the execution $\hat{E}$ by making a similar simulation on q and the processors to its right. This procedure can be repeated until the desired execution $\hat{E}$ is constructed.

Once the execution $\hat{E}$ is constructed, select in $\hat{D}$ two processors that are adjacent in $\overline{D_b}$ but not in $\hat{D}$, connect them by a link as above, and repeat the procedure above. By repeating this for every link in $\overline{D_b}$, we get the desired execution $\overline{E_b}$. □

The construction of $\overline{C_b}$ and $\overline{C'_b}$ guarantees that no two processors in $\overline{C_b}$ and no two processors in $\overline{C'_b}$ have the same history in the execution $\overline{E_b}$ on $\overline{D_b}$ described above. Hence no three processors in $\overline{D_b}$ have the same history in $\overline{E_b}$. This implies, by Lemma 2, that the number of bits received by any l distinct processors in $\overline{D_b}$ is larger than $(l/8)\log_4(l/4)$.

Let m_k be the length of $\overline{D_k}$ (recall that k is defined by the equality $t{=}nk$). In the case where $m_k \le n$ we can pad $\overline{D_k}$ with $n-m_k$ processors that receive input zero. As in the proof Theorem 1, the messages sent by these $n-m_k$ processors never reach their target. According to Lemma 6, $p_{n,k}$ accepts, and hence by Lemma 7 the corresponding processor in $\overline{D_k}$ accepts too. By distinguishing two cases depending on whether $m_k \le n-\log n$ or not, one can complete the proof of Theorem 1' as in Theorem 1, provided that $m_k \le n$.

However if $m_k > n$ then one cannot proceed as in the unidirectional case. If we "cut" the n leftmost processors of $\overline{D_k}$, then the n-th processor does not receive the proper messages from the right, and the proof requires modification. This is where we use the histories $\overline{E_b}$ for $b < k$.

Lemma 8: Let m_b be the length of $\overline{D_b}$ and let $m_0{=}0$. For any b such that $1 \le b \le k$, either the last n processors of C_b or the first n processors of C'_b contain at least $\dfrac{m_b - m_{b-1}}{2}$ distinct histories in E_b.

Proof: For notational convenience assume that processor lines with subscript 0 are empty. For a line Q let $|Q|$ denote the number of processors in it. Recall that D_b is obtained by inserting a line of $2n$ processors between the left half, C_{b-1}, and the right half, C'_{b-1}, of D_{b-1}. Since $m_b = |\overline{D_b}| = |\overline{C_b}\,\overline{C'_b}|$ and $m_{b-1} = |\overline{D_{b-1}}| = |\overline{C_{b-1}}\,\overline{C'_{b-1}}|$, either $|\overline{C_b}| - |\overline{C_{b-1}}| \ge \dfrac{m_b - m_{b-1}}{2}$ or $|\overline{C'_b}| - |\overline{C'_{b-1}}| \ge \dfrac{m_b - m_{b-1}}{2}$. Assume w.l.o.g. the former is the case. It suffices to show that the last $\frac{m_b - m_{b-1}}{2}$ processors of $\overline{C_b}$ are from the last n processor of C_b. Observe that $\overline{C_b}$ is constructed by taking a prefix of $\overline{C_{b-1}}$ and appending it with a sequence of distinct processors from the last n processors of C_b. Thus the processors at position $|\overline{C_{b-1}}|+1$ through position $|\overline{C_b}|$ of the line $\overline{C_b}$ must be from the last n processors of C_b.□

We are now ready to prove the Theorem 1' for the remaining case that $m_k > n$ (recall that $t = kn$). Let b be the smallest integer such that $m_b > n$. Clearly either $m_b - m_{b-1} \geq n/2$ or $n/2 < m_{b-1} \leq n$.

In the former case by Lemma 8 there are n consecutive processors in D_b which have at least $n/4$ distinct histories the lower bound follows from Lemma 2 and Corollary 2. In the second case, let τ be the string composed of the concatenation of the input letters to the processors in $\overline{D_{b-1}}$. Then $n/2 < m_{b-1} = |\tau| \leq n$. Consider the execution on a line with input string τ' consisting of the execution $\overline{E_{b-1}}$ on the first m_{b-1} processors and assume that the remaining processors are never awakened. Since this execution has more than $n/2$ distinct histories, the result follows. This completes the proof of Theorem 1'. □

5. LOWER BOUNDS FOR RINGS WITH IDENTIFIERS

Interestingly, the gap remains for rings with identifiers if the set of possible identifiers is large enough relative to the ring size n. Now the algorithm AL that each processor runs is given two inputs, a symbol of the input alphabet and an *identifier* of the processor. Different processors must receive distinct identifiers. An *execution* of an algorithm includes a labeling of the ring with distinct identifiers in addition to an assignment of an input symbol to each processor, an orientation of the ring, and a combined schedule of how the algorithm is run at each processor. The function value may only depend on the labeling of the processors with input symbols and must be invariant over all labelings of the ring with distinct identifiers, all orientations, and all possible schedules. The *bit* (*message*) *complexity* of an algorithm is again the maximal number of bits (messages) sent over all possible executions of this algorithm and the bit (message) complexity of a function f is the minimal possible bit (message) complexity of any algorithm that computes f. Note that this notion of complexity is worst case over all possible labelings of the ring with distinct identifiers.

To prove the lower bound of $\Omega(n \log n)$ on the bit complexity for rings with identifiers we make the following simplifying assumptions. Assume that AL rejects 0^n and accepts an input string ω, as before. Let Σ denote the set of letters in the string $0 \cdot \omega$. Observe that the function computed by AL is non-constant even for input strings over the restricted input alphabet Σ^n. For each $a \in \Sigma$ let $e(a)$ be an encoding of a by at most $\log n + 2$ bits. For a specific execution of AL, in which all the input letters are from Σ, define $eh_i(s)$, the *extended history* of the processor p_i at time s in this execution, to be a string $e(a) t_i(1) d_i(1) m_i(1) \ldots\ t_i(r_s) d_i(r_s) m_i(r_s)$, where a is the input of p_i in this execution, the $d_i(j)$s and the $m_i(j)$s are, as before, directions and messages, and the $t_i(j)$s are either I if the corresponding message was received by p_i, or O if it was sent by it.

Theorem 2: The bit complexity of a bidirectional oriented ring of n processors with distinct identifiers is $\Omega(n\log n)$, provided that the set of identifiers that can be assigned to the processors is large enough.

Proof: Assume that there is an algorithm *AL* of bit complexity[5] $o(n\log n)$ computing a non-constant function and let Z be the set of possible identifiers, with $|Z| \geq n2^{n^n}$. Let $\boldsymbol{H}_z$ be the set of possible extended histories that can occur at a processor with identifier $z \in Z$ when *AL* is executed on an input from Σ^n. Then the definition of extended history implies that the program of a processor with identifier z in executions on input from Σ^n is uniquely determined by $\boldsymbol{H}_z$, and hence that if $\boldsymbol{H}_z = \boldsymbol{H}_{z'}$ then two processors with identifiers z and z' behave identically in all executions on inputs from Σ^n. Since, apart from the $\log n + 2$ bits required to encode the input, the length of the extended history of a processor is at most three times the number of bits received or sent by that processor, the assumption on the bit complexity of *AL* implies that the set of all possible extended histories $\boldsymbol{H} = \bigcup_{z \in Z} \boldsymbol{H}_z$ is of cardinality $o(2^{n\log n}) = o(n^n)$.

Denote $z \equiv_{AL} z'$ if $\boldsymbol{H}_z = \boldsymbol{H}_{z'}$. The relation $\equiv_{AL}$ partitions Z into at most $2^{|\boldsymbol{H}|} = o(2^{n^n}) = o(\frac{|Z|}{n})$ equivalence classes, where all the processors in each class behave identically on inputs from Σ^n. Thus, there must be n distinct identifiers in Z which belong to the same equivalence class. By restricting the identifiers to this equivalence class, the proof of the anonymous case implies a lower bound of $\Omega(n\log n)$ on the bit complexity of *AL*, which contradicts the assumption. □

6. FUNCTIONS COMPUTABLE WITH A SMALL NUMBER OF MESSAGES

In [ASW88] a non-constant function was presented that is computable in $O(n)$ messages and $O(n\log n)$ bits. That function is the characteristic function for the subset of $\{0,1\}$ which consists of all cyclic shifts pattern $0(01)^*$. The pattern requires that the ring size n is odd. As we will show here, for any non-divisor k of n there are similar patterns that can be recognized in $O(kn)$ messages and $O(kn + n\log n)$ bits. Since for any ring size n there is a non-divisor of size $O(\log n)$ this leads to a non-constant function defined for all ring sizes with message and bit complexity $O(n\log n)$. This shows that the $\Omega(n\log n)$ bounds on the bit complexity given in the previous sections are tight.

As the main result of this section we exhibit a function with ‘‘almost linear’’, i.e. $O(n\log^* n)$, message complexity, and this function is defined for arbitrary ring size. The lower bound proven in [DG] for specific ring sizes matches this upper bound. The upper bounds proven in this section assume alphabets of constant size at least two. In fact, we show that if the alphabet size is at least n, then there are non-constant functions that can be computed with $O(n)$ messages (The $\Omega(n\log^* n)$ lower bound of [DG] assumes constant alphabet size).

---

[5] We write $f(n) = o(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

All algorithms presented in this section work on the unidirectional ring and output one if and only if the input is a cyclic shift of a string θ. It is easy to derive bidirectional versions of these algorithms working on bidirectional rings of arbitrary orientation such that the message complexity increases by a at most a factor of two. Recall that algorithms working on unidirectional rings assume that the ring is oriented and they only receive messages from the left and send messages to the right. Let U be such and algorithm. In the bidirectional version each processor runs two non-interacting versions of U, one that receives messages from the left neighbor and sends messages to the right neighbor, and a second one that receives messages from the right neighbor and sends to the left neighbor. Here left and right refer to the local orientation of the processor. The local orientations between processors do not have to be consistent with each other. After both algorithms terminate each processor outputs one if and only if one of the algorithms outputs one. It is easy to see that the bidirectional version computes one if and only if the input string or the reverse of the input string is a cyclic shift of θ.

To get an easy introduction to the unidirectional algorithms presented in this section, we will first present a simple generalization of an algorithm of [ASW88], which is also used in our algorithm. This generalization, which we call *NON−DIV*, is defined for two parameters: the ring size n, and an integer k which does not divide n ($n > k$). It accepts the (cyclic shifts of) the pattern $\pi = 0^{n \bmod k}(0^{k-1}1)^{\lfloor n/k \rfloor}$. The message complexity of *NON−DIV* is $O(kn)$ and its bit complexity is $O(kn+n\log n)$. Algorithm *BRUIJN*, the second algorithm described, follows the same outline as *NON−DIV* but is used to recognize more complicated patterns. Finally, *BRUIJN* is used iteratively in Algorithm *STAR* for recognizing a pattern in $O(n\log^* n)$ messages.

Algorithm *NON−DIV*(k,n)**:**
%% it is assumed that $n \bmod k = r \neq 0$. %%
FOR all n processors in parallel DO

N1. Set your status to *passive* and send your bit to the right neighbor. Forward $k+r-3$ bits received from the left to the right and wait until you receive $k+r-2$ bits from the left.

N2. Let ψ be the concatenation of the $k+r-2$ bits received from the left with your own bit (ψ contains $k+r-1$ bits).

IF ψ is not a cyclic substring of $\pi = 0^r(0^{k-1}1)^{\lfloor n/k \rfloor}$ THEN send a *zero−message* and terminate with output zero.

IF $\psi = 0^{k+r-1}$ THEN send a *size−counter* message with count one to the right and change your status to *active*.

Wait for a message from the left.

%% This must be either a *zero−message* or a *size−counter*. %%

N3. IF the message received is a *zero−message* THEN forward it and terminate with output zero.

IF the message received is a *size−counter* THEN

IF your status is *passive* THEN increment the *size−counter* by one and forward it to the right.

ELSE (i.e., your status is *active*):

IF the value of the *size−counter* is n THEN send a *one−message* to the right and terminate with output one.
ELSE send a *zero−message* to the right and terminate with output zero.
ELSE (The message received must be either *one−message* or a *zero−message*)

N4. Forward the message you received and terminate with output zero if it was a *zero−message* and with output one otherwise.

We sketch below a proof that *NON−DIV* indeed recognizes the string $\pi = 0^r(0^{k-1}1)^{\lfloor n/k \rfloor}$, where $r = n \bmod k$.

Case 1: The input string contains a substring of length $k+r-1$ which is not a substring of π. Then the processor with the rightmost bit of this substring will initiate a *zero−message* in Step $N2$, and will never forward a *size−counter*. This latter fact means that no *size−counter* will make a complete traversal of the ring, and hence that no *one−message* can be initialized. The *zero−messges* (possibly more than one) produced in Step N2 will eventually cover the whole ring and all processors will output zero.

Case 2: This case is the complement of Case 1. Thus no *zero−message* will be initiated in Step $N2$. A simple case analysis shows that in this case the input string must contain a substring of $k+r-1$ successive zeroes, and that it contains exactly one such substring if and only if the input string is a cyclic shift of π. This means that at least one *size−counter* will be initiated in Step $N2$, and exactly one *size−counter* will be initiated if and only if the input string should be accepted. The proof is completed by observing that output one is obtained only if some *size−counter* traverses the whole ring which is only possible if exactly one such counter was initiated. If more than one *size−counter* is initiated, then eventually some processor will generate a *zero−message* and all processor will output zero.

Each processor sends at most $2k$ messages in an execution of *NON−DIV*, so that the total number of messages is $O(kn)$. Counters cost at most $\lfloor \log n \rfloor + 1$ bits, so the total bit complexity of *NON−DIV* is $O(kn + n\log n)$.

Now it is easy to derive an algorithm computing a non-constant function of $O(n\log n)$ bit complexity uniformly for all ring sizes: First each processor determines the smallest non-divisor k of the ring size n and then runs *NON−DIV*(k,n). Since k is $O(\log n)$ we get an algorithm for a non-constant function whose bit complexity matches the lower bounds of the previous sections.

Lemma 9: There is a non-constant function for binary input alphabet defined for all ring sizes n with bit complexity $O(n\log n)$ on the unidirectional ring.

Before proceeding we note that if the input alphabet has size at least n, then there are non-constant functions of $O(n)$ message complexity: let $\sigma_0, \cdots, \sigma_{n-1}$ be n letters of the alphabet. The cyclic shifts of $\sigma = \sigma_0\sigma_1 \cdots \sigma_{n-1}$ can be accepted as follows. Send your input letter to your right neighbor as in Step $N1$ of *NON−DIV*, and concatenate the letter received with your own letter as in $N2$ to form ψ; if ψ is not of the form $\sigma_i\sigma_{i+1 \bmod n}$ $(i = 0, \cdots, n-1)$ then initiate a *zero−message* and terminate with output 0. The pattern $\psi = \sigma_{n-1}\sigma_0$ initiates a counter message. Thus we have:

Lemma 10 (Hans Bodlaender): If the input alphabet is of size at least n, then the distributed message complexity of ring of n processors is $O(n)$. □

The above lemma can be generalized to alphabet size εn for arbitrary positive constant ε. Moreover, similar technique can be used to show a linear distributed message complexity of other networks, provided the alphabet size is linear in the network size.

The final goal of this section is to present an algorithm called $STAR(n)$ that computes a nontrivial function in $O(n\log^* n)$ messages for arbitrary ring size n. So far we have shown that if n is not divisible by some integer $k{=}O(log^*n)$ then $NON-DIV(k,n)$ is such an algorithm. However it is much harder to find algorithms of low message complexity if n has no small non-divisors.

$STAR(n)$ recognizes patterns based on the classical de Bruijn Sequences [B46]. A *de Bruijn Sequence* β_k is a sequence of 2^k bits with the property that each binary string of length k occurs in β_k exactly once as a cyclic substring [B46]. For each $k{\geq}1$ there are such sequences (see e.g. [E79]). From now on assume that β_k denotes a fixed such sequence with the property that its first k bits are zeroes, and that the first zero is barred. That means that our input alphabet has three letters 0, 1 and $\bar{0}$, which we call bits for simplicity. We discuss later how to get the corresponding results for the two-letter case.

One way to construct a de Bruijn sequence β_k is the following: start with $\bar{0}0^{k-1}$; bit i ($k{+}1{\leq}i{\leq}2^k$) is one if the string of length k formed by the bits $i{-}k{+}1$, $i{-}k{+}2$, $\cdots$ $i{-}1$ appended by a one does not appear yet in the sequence; otherwise bit i is zero. The sequences for $k{=}1$, 2, 3, 4 are $\bar{0}1$, $\bar{0}011$, $\bar{0}0011101$, $\bar{0}000111101100101$, respectively.

We will use prefixes of strings in $(\beta_k)^*$ to construct patterns recognizable with a small number of messages. Note that in these prefixes, each new copy of β_k starts with $\bar{0}$. Let $\pi_{k,n}$ (for $k{\leq}n$) be the first n bits of $(\beta_k)^n$. For example, $\pi_{3,21}{=}\bar{0}0011101\bar{0}0011101\bar{0}0011$. The Algorithm $STAR(n)$ will essentially recognize cyclical shifts of a word formed by interleaving a number of patterns of the form $\pi_{k',n'}$, for various choices of k' and n'.

At times the algorithm checks whether the parts of the input string fulfill a local condition of legality w.r.t. some $\pi_{k',n'}$, which we define now. Let $\theta{=}\theta_0 \cdots \theta_{n-1}$ be a cyclic string of length n representing the input of the n processors. Then θ_i is *legal* w.r.t. $\pi_{k,n}$ if the k bits to the left of θ_i appended with θ_i produces a string that occurs as a cyclic substring in $\pi_{k,n}$. The following lemma shows that if all bits of θ are legal w.r.t. $\pi_{k,n}$ then this string must be equal to $\pi_{k,n}$ or a close relative thereof.

Let τ be any bit string and σ be a cyclical substring if τ. A bit b is called a *successor* of σ in τ if σb is a cyclical substring of τ. Let the last k bits of a string $\alpha_0,\alpha_1,\cdots,\alpha_{l-1}$ of length $l{\geq}k$ be the string $\alpha_{l-k},\alpha_{l-k+1},\cdots,\alpha_{l-1}$. Let ρ be the last k bits of $\pi_{k,n}$. Every length k cyclic substring $\sigma{\neq}\rho$ of $\pi_{k,n}$ has exactly one successor in $\pi_{k,n}$. However there might be two successors of ρ in $\pi_{k,n}$. Clearly $\bar{0}$ is a always a successor. If $n>2^k$ then $\pi_{k,n}$ contains β_k as substring and ρ has b as a successor in $\pi_{k,n}$, where b is the unique successor of ρ in β_k. Now $b \neq \bar{0}$ if and only if and $n \neq 0 \bmod 2^k$.

Lemma 11: Assume all n bits of an input string θ are legal w.r.t. $\pi_{k,n}$ and let ρ consists of the last k bits of $\pi_{k,n}$. If $n = 0 \bmod 2^k$ then θ is a cyclic shift of $(\beta_k)^{n/2^k}$. If $n \neq 0 \bmod 2^k$ then θ contains $\rho\bar{0}$ at least once as a cyclical substring and $\rho\bar{0}$ occurs exactly once if and only if θ is a cyclical shift of $\pi_{k,n}$.

Proof: If $n = 0 \bmod 2^k$ then $\pi_{k,n} = (\beta_k)^{n/2^k}$. Thus $\pi_{k,n}$ and β_k have the same set of cyclical substrings of length k and each such substring has the same unique successor in $\pi_{k,n}$ and in β_k. Since all bits of θ are legal w.r.t. $\pi_{k,n}$ we conclude that if $n = 0 \bmod 2^k$ then θ must be a cyclic shift of $(\beta_k)^{n/2^k}$.

If $n \neq 0 \bmod 2^k$ then by the legality assumption the successors of all cyclical substrings of length k of θ are uniquely determined except for ρ which has two successors: $\bar{0}$ and the successors of ρ in β_k. Thus after each occurrence of ρ the current copy of β_k is completed or it is cut off at ρ and a new copy of β_k is begun. The definition of ρ implies that the subword of β_k beginning with $\bar{0}$ and ending with ρ has length $n \bmod 2^k$ and equals $\pi_{k,n \bmod 2^k}$. Thus θ must be a cyclical shift of a string of length n of the form $(\beta_k + \pi_{k,n \bmod 2^k})^*$. Now the second half for the claim follows easily. □

If the ring size n is not divisible by $k = 1 + \log^* n$ then $STAR(n)$ simply calls $NON-DIV(k,n)$. However if $n = 0 \bmod (1 + \log^* n)$ then the algorithm recognizes a complicated pattern $\theta^{(n)}$ which we will now describe. Let $n' = \dfrac{n}{1+\log^* n}$. $\theta^{(n)}$ is a string in the language $F = (\#\{\bar{0},0,1\}^{\log^* n})^{n'}$ over the four letter alphabet $\{0,1,\bar{0},\#\}$ containing $l(n)$ "interleaved" de Bruijn Sequences, where $l(n)$ is defined as follows: let $k_0 = 1$ and $k_{i+1} = 2^{k_i}$; $l(n)$ is the minimum i such that k_i does not divide n'. Note that $\log^* n$ is the minimum i such that $k_i \geq n$, and hence $l(n) \leq \log^* n$.

For all strings θ in F, let $\theta[i]$ (for $1 \leq i \leq \log^* n$) be the concatenation of the n' bits which are the i-th letters to the right of the # letters. For example, $\theta[1]$ consists of the bits to the right of the # letters, and $\theta[\log^* n]$ consists of the bits to the left of the # letters. $\theta^{(n)}$ is the string in F with the following properties:

(1) $\theta^{(n)}[i] = \pi_{k_{i-1},n'}$, for $1 \leq i \leq l(n)$, and

(2) $\theta^{(n)}[i]$ contains only 0's, for $l(n)+1 \leq i \leq \log^* n$.

As a rough outline, Algorithm $STAR(n)$ first checks in Step S0 whether the input θ is in F. Note that if this is true then there are exactly n' processors with input #. These processors check in loop i of Step S1 whether $\theta[i]$ is a cyclic shift of $\pi_{k_{i-1},n'}$. To do this for $\theta[l(n)]$ Step S2 is needed as well.

Algorithm *STAR* (n)

S0. IF $n \neq 0 \bmod (\log^* n + 1)$ THEN call $NON-DIV(k,n)$, for $k = \log^* n + 1$, to recognize cyclic shifts of $0^{n \bmod k}(0^{k-1}1)^{n/k}$.

ELSE

BEGIN

Send your input to your right neighbor, forward $\log^* n$ inputs, and wait for $1+\log^* n$ inputs.

IF the number of # signs you have received is not exactly one THEN send a *zero−message* to the right and terminate with output zero.

%% If no processor initiates a *zero−message* in the previous statement then there are n' # signs in the input, all of which are exactly $1+\log^* n$ apart. Each of processor with input # denotes the $\log^* n$ bit string between itself and the previous processor with input # as $b_1 \cdots b_{\log^* n}$. %%

IF your input is # and $b_{l(n)+1} b_{l(n)+2} \cdots b_{\log^* n}$ contains a letter other than 0 or if $b_i = \bar{0}$, for $1 < i \leq l(n)$, and $b_{i-1} \neq \bar{0}$ THEN send a *zero−message* to the right and terminate with output zero.

END

%% For the remaining part of the algorithm whenever any processor receives a *zero−message* (*one−message*) it forwards it and terminates with output zero (respectively one). Furthermore we only address processors with input #. All other processors always only forward the messages they receive and terminate as described above.%%

S1. For $i := 1$ to $l(n)$ DO

BEGIN

IF $i = 1$ THEN all processors are *initiators*.

ELSE you are an *initiator* if your bit b_{i-1} equals $\bar{0}$.

%% The *initiaters* are $k_{i-1}(1+\log^* n)$ apart.%%

IF you are an *initiator*

THEN start an *input collection message* by sending your bit b_i to the right.

IF you are an *initiator* THEN do the following once and ELSE do it twice:

Append your bit b_i to the *input collection message* received and forward it.

%% The second *input collection message* received by each *initiator* processor has length $2k_{i-1}$. The concatenation of the second halves of all these messages constitute $\theta^{(n)}[i]$. %%

IF you are an *initiator* and any of the bits in the second half of the last message is not legal w.r.t. $\pi_{k_{i-1},n'}$ THEN initiate a zero-message.

%% Note that $2^{k_{i-1}} = k_i$ divides n' for all i except $i = l(n)$. %%

END

S2. IF your last message contains $\rho\bar{0}$ as a substring, where ρ consists of last $k_{l(n)-1}$ bits of $\pi_{k_{l(n)-1},n'}$

THEN send a *size−counter* with count one to the right.

IF you receive a *size−counter* and did not start a *size−counter*

THEN increment it and forward it.
ELSE IF the received *size−counter* equals n'
THEN send a *one−message* to the right.
ELSE Send a *zero−message*.
END

Lemma 12: For all ring sizes $n = 0 \bmod (log^{*}n + 1)$, Algorithm $STAR(n)$ recognizes cyclic shifts of $\theta^{(n)}$ in $O(n\log^{*}n)$ messages.

Proof: Case 1: A *zero−message* was generated in Step S0. That is either (i) the input string θ is not in $F = (\#\{\bar{0},0,1\}^{\log^{*}n})^{n'}$, or (ii) for some processor with input # one of the bits $b_{l(n)+1}, \cdots, b_{\log^{*}n}$ is not 0, or (iii) for some such processor $b_i = \bar{0}$ and $b_{i-1} \neq \bar{0}$ for some $1 \leq i \leq l(n)$. In any of the above three cases the input string is not a cyclical shift of $\theta^{(n)}$ and the processor which generated a *zero−message* will not forward any *size−counter*. Hence no *size−counter* will traverse the whole ring, and no processor will initiate a *one−message*. We conclude that in this case all the processors will eventually output zero.

Case 2: Case 1 does not hold and for some $1 \leq i \leq l(n)$, $\theta[i]$ contains a bit which is not legal with respect to $\pi_{k_{i-1},n'}$. Assume i is minimum. We first claim that all initiators of loop i are $k_{i-1}(1+\log^{*}n)$ apart. This clearly holds for $i=1$. If $i>1$ then in loop $i-1$ all bits of $\theta[i-1]$ were legal w.r.t. $\pi_{k_{i-2},n'}$. Recall that $n'=0 \; mod \; k_{i-1}$ and that $2^{k_{i-2}} = k_{i-1}$. Thus Lemma 11 implies that $\theta[i-1]$ is a cyclic shift of $(\beta_{k_{i-2}})^{n/k_{i-1}}$ and all occurrences of $\bar{0}$ in $\theta[i-1]$ are k_{i-1} apart. This guarantees that the claim holds for $i>1$ as well.

Since all initiators are properly spaced the second input collection message the initiators receive are exactly $2k_{i-1}$ long and the second halfs of these messages form $\theta[i]$. Each initiator can check whether the bits in the second half are legal w.r.t. $\pi_{k_{i-1},n'}$ since it knows the preceding substrings of length k_{i-1}. By the above choice of i one of the bits must be illegal and thus some initiator will start a *zero−message*. Thus in Case 2 all processors correctly output zero.

Case 3: Case 1 does not hold and for $1 \leq i \leq l(n)$, all bits of $\theta[i]$ are legal w.r.t. $\pi_{k_{i-1},n'}$. Since all bits of $\theta[l(n)]$ are legal w.r.t. $\pi_{k_{l(n)-1},n'}$ and $n' \neq 0 \; mod \; k_{l(n)}$, Lemma 11 guarantees that there is at least one occurrence of $\rho\bar{0}$ in $\theta[l(n)]$, where ρ is the last $k_{l(n)-1}$ bits of $\pi_{k_{l(n)-1},n'}$. Thus there will be at least one initiator with $\rho\bar{0}$ as a substring of its *second input collector* message and these initiators will start *size−counters*. As for *NON−DIV* if more than one counter is initiated then this eventually causes a *zero−message* and all processors terminate with output zero. Only if exactly one counter travels around the whole ring, a *one−message* is initiated and all processors output one. The correctness of $STAR(n)$ in Case 3 follows again from Lemma 11 which assures us that exactly one counter is initiated in Step S3 if and only if $\theta[l(n)]$ is a cyclic shift of $\pi_{k_{l(n)-1},n'}$.

It is easy to see that Algorithm *STAR* sends $O(n\log^{*}n)$ messages in Step S0. In Step S1 each loop costs $O(n)$ messages and there are $l(n) = O(log^{*}n)$ iterations. In Step S2 only $O(n)$

message are sent. □

Note that $\theta^{(n)}$ might use up to four letters. To recognize similar strings $\theta'^{(n)}$ over a binary input alphabet we encode the i-th letter ($1 \le i \le 4$) by $1^i 0^{5-i}$. If $n \neq 0 \bmod 5$ then $\theta'^{(n)} = 0^{n \bmod 5}(0^4 1)^{n/5}$ and otherwise $\theta'^{(n)}$ equals $\theta^{n/5}$ using the five bit encoding for all letters. It is easy to see that for all ring sizes $\theta'^{(n)}$ can also be recognized in $O(n \log^* n)$ messages.

Theorem 3: There is a non-constant function for binary input alphabet defined for all ring sizes n with message complexity $O(n \log^* n)$ on the unidirectional ring.

Acknowledgment: Thanks to Hans Bodlaender for pointing out Lemma 10 to us, to Gerhard Buntrock for many useful discussions, and to an anonymous referee for numerous and insightful comments, which helped us to clarify and improve the presentation of this paper.

REFERENCES

[AAHK89] K. Abrahamson, A. Adler, L. Higham and D. Kirkpatrick, "Randomized Function Evaluation on a Ring," *Distributed Computing* Vol. 3, 1989, pp. 107 - 117.

[AHU83] A. V. Aho, J. E. Hopcroft, J. D. Ullman, "Data Structures and Algorithms," Addison-Wesley, 1983.

[ASW88] C. Attiya, M. Snir, and M. K. Warmuth, "Computing on an Anonymous Ring," *J. of the ACM*, Vol. 35, No. 4, Oct. 1988, pp. 845-875.

[B46] N. G. de Bruijn, "A Combinatorial Problem," *Proc. of Koninklijke Nederlands Akademie van Wetenschappen,* Vol. 49, Part 2, 1946, pp. 758-764.

[B80] J. E. Burns, "A Formal Model for Message Passing Systems," Technical Report No. 91, Computer Science Department, Indiana University, Bloomington, IN 1980.

[BB89] P. W. Beame and H.L. Bodlaender, "Distributed Computing on Transitive Networks: The Torus", proc. of the 6th STACS, pp. 294-303 (1989).

[BMW91] H. L. Bodlaender, S. Moran and M. K. Warmuth, "The Distributed Bit Complexity of the Ring: >From the Anonymous to the Non-anonymous Case", Accepted for publication in *Information and Computation*.

[DKR82] D. Dolev, M. Klawe and M. Rodeh. "An $O(n \log n)$ Unidirectional Algorithm for Extrema Finding in a Circle," *J. of Algorithms,* Vol. 3, No. 3, 1982, pp. 245-260.

[DG87] P. Duris and Z. Galil, "Two Lower Bounds in Asynchronous Distributed Computation," *Proc. of IEEE Conference on Foundations in Computer*

Science, 1987, pp. 326-330.

[E79] S. Even, "Graph Algorithms," Computer Science Press, 1979.

[FLM85] M. J. Fischer, N. A. Lynch, and M. Merritt, "Easy Impossibility Proofs for Distributed Consensus Problems," *Proc. of the Fourth Annual ACM Symposium on Principles of Distributed Computation,* Minaki, Ontario, Canada, August 1985, pp. 59-70.

[H68] J. Hartmanis, "Computational Complexity of One-Tape Turing Machine Computations," *J. of the ACM,* Vol. 15, No. 2, 1968, pp. 325-339.

[MW86] S. Moran and M. K. Warmuth, "Gap Theorems for Distributed Computation," *Proceeding of Fifth Annual ACM Symposium on Principles of Distributed Computing,* pp. 131-140, Calgary, Alberta, Canada, 1986.

[MZ87] Y. Mansour and S. Zaks, "On the Bit Complexity of Distributed Computations in a Ring With a Leader," *Information and Computation,* Vol. 75, No. 2, 1987, pp. 162-177.

[P82] G. L. Peterson, "An $O(n \log n)$ Unidirectional Algorithm for the Circular Extrema Problem," *ACM Transactions on Programming Languages and Systems,* Vol. 4, 1982, pp. 758-762.

[PKR84] J. Pachl, E. Korach and D. Rotem, "A new technique for proving lower bounds for distributed maximum-finding algorithms," *J. of the ACM,* Vol. 31, No. 4, Oct 1984, pp. 905-918.

[T64] B. A. Trachtenbrot, "Turing computations with logarithmic delay" (In Russian), *Algebra i Logica,* Vol. 3, 1964, pp. 33-48, English Translation in University of California Computing Center, Technical Report, No. 5, Berkeley, California, 1966.