On the complexity of global computation in the presence of link failures: The general case

Y. Afek

Computer Science Department
Tel-Aviv University
Israel 69978

D. Hendler

Computer Science Department
Tel-Aviv University
Israel 69978

Abstract

This paper presents $\Omega(m \log n)$ and $\Omega(mn)$ messages lower bounds on the problem of computing a global sensitive function in bidirectional networks with link failures (i.e., dynamically changing topology), where n and m are the total number of nodes and links in the network. The $\Omega(m \log n)$ lower bound is under the assumption that n is a-priori known to the nodes, while the second bound is for the case in which such knowledge is not available.

A global sensitive function of n variables is a function that may not be computed without the knowledge of the values of all the n variables (e.g. maximum, sum, etc). Thus, computing such a function at one node of a distributed network requires this node to communicate with every other node in the network.

Though lower bounds higher than $\Omega(m)$ messages are known for this problem in the context of link failures, none holds for dense bidirectional networks. Moreover, we are not aware of any other non-trivial lower bound higher than $\Omega(m)$ for dense bidirectional networks.

1 Introduction

While the design of distributed algorithms for static and dynamic networks has rapidly developed in recent years, the study of lower bounds on the complexity of such protocols was lagging behind. Moreover, most of the existing lower bounds for distributed computing are for computations on a ring network [1, 2, 3, 4, 5, 6]. True, in static networks it is often the case that the ring lower bound is close or is easily extended to a lower bound on general networks (e.g. leader election lower bounds). In dynamic networks however, the situation is different and a big gap usually exists there between the lower bound on a ring and the upper bound on a general network. A good example of this gap, is the problem of computing a global sensitive function in a bidirectional dynamic network. The best known upper bound for this problem on a general network is $O(nm)$ messages, and the best lower bound for bidirectional networks, demonstrated on a ring, is $\Omega(n \log n)$ messages, resulting in a general lower bound of $\Omega(m + n \log n)$ messages [7], where n and m are the total number of nodes and links in the network, respectively. In the special case of a strongly connected unidirectional network, Goldreich and Sneh have demonstrated an almost tight lower bound of $\Omega(\frac{nm}{PolyLog(n)})$ for computing a global sensitive function.

In this paper we present an $\Omega(m \log n)$ messages lower bound on the problem of computing a global sensitive function in asynchronous bidirectional networks in which links may fail-stop. A global sensitive function of n variables [7, 8] is a function that may not be computed without the knowledge of the values of all the n input variables (e.g. maximum, sum, etc). This paper considers computing such functions on n input variables, one in each node of the network. Computing such a function at one node requires this node to communicate with every other node in the network. Clearly, if the set of links that may fail-stop disconnects the network then no algorithm can compute a global sensitive function. Henceforth we assume that the set of links that may fail-stop does not disconnect the network, but no other assumption is made on this set.

An $O(mn)$ messages computation of global sensitive functions is easily derived by initiating a flood of the input variable from each node that joins the computation (assuming that n is known to the processors). This is the asymptotically most efficient protocol that is known to this problem, even if messages size is not constrained. If n is unknown then a solution with $O(mn)$ messages - for both the unidirectional and the bidirectional problems - is derived by using the connectivity with termination

case		*This work*	*previous work*
Bidirect	n known	$m \log n$	$m + p \cdot \log p$
	n unknown	$m \cdot n$	$m + p^2$
Unidirect	n known	-	$\frac{mn}{PolyLogn}$
	n unknown	$m \cdot n$	$\frac{mn}{PolyLogn}$

* Where p is the length of the longest Hamiltonian circuit.

Figure 1: Summary of our asymptotic lower bound results and the corresponding results of Goldreich, Sneh and Shrira.

detection mechanism described in [9] (Algorithm CT2 there).

In [10] Goldreich and Shrira present two lower bounds on the problem of computing a global sensitive function in a ring network with fail-stopped links, which they call the *consultation problem*: A tight lower bound of $\Omega(n \cdot \log n)$ messages when n - the number of processors - is known to the processors, and a tight lower bound of $\Omega(n^2)$ messages when the processors have no information about n. In [11] Goldreich and Sneh present an almost tight lower bound of $\Omega(m \cdot n / PolyLog(n))$ messages for *unidirectional* general networks, or networks where edge-failures may be unidirectional. The best lower bound on global sensitive computations in a general bidirectional network until this work is $\Omega(m + p \cdot \log p)$ messages if n is known to the processors and $\Omega(m + p^2)$ messages if the processors have no knowledge of n, where p is the number of processors in the longest Hamiltonian circuit of the network [7]. In this paper we improve these results (see table in Figure 1) and achieve a lower bound of $\Omega(m \log n)$ for the case where n is known and a tight lower bound of $\Omega(m \cdot n)$ messages for the case that the processors have no a-priori information about n. To the best of our knowledge, these are the first non-trivial lower bounds for any problem on dense bidirectional networks in this model.

2 The model

We consider communication networks that are given by an undirected graph $G(V, E)$, where V the set of nodes represent processors and E the set of edges represent *bidi-*

rectional communication-links. Communication over links obeys the FIFO discipline. In addition the networks we consider are completely *asynchronous*, i.e., there is no a-priory bound on the time a sent message may be delayed over a link.

A link has *fail-stopped* during an execution if there is a point in the execution after which it delivers no message (i.e., no messages are received by both its end points, regardless of what is sent over it). The transmission delay of messages sent after this point on the fail-stopped link is considered to be infinite (∞), this model is termed ∞-delay in [12]. We further assume that *link failures are undetectable*, hence there is no way a processor can tell whether a message is not received over a link because the link has fail-stopped or because the link is slow.

Global computation, or - as called by Goldreich Shrira and Sneh - *consultation*, is the task of computing an n variate global sensitive function over a set of inputs, one at each node of the network. A function is global sensitive [10, 11], if there is an n-tuple $v_1, \ldots, v_n$ in its domain, whose value cannot be determined by any subset of $n-1$ elements. Examples of such functions are: *sum*, *maximum*, *and*, *search*, etc.

Additionally, in the lower bound proofs we assume that the computation is initiated by a single processor. Since this assumption strengthens the protocol it does not weaken the lower bound results.

Global sensitive functions may not be computed if the communication network is disconnected, therefor we consider only networks in which during any execution, the subnetwork induced by the set of non-faulty links is a connected spanning subgraph of the network (i.e., the set of fail-stopped links never disconnects the network). Such networks are called *safe*.

All the results in this paper are on the *message complexity* of the problem under consideration. The *message complexity* of a protocol is the worst case number of messages sent by the protocol, where the maximum is taken over all possible runs for a given network. The *message complexity* of a problem, for a given network, is the minimum of the above over all protocols that solve the problem.

3 Known network size

In this section we prove the following theorem:

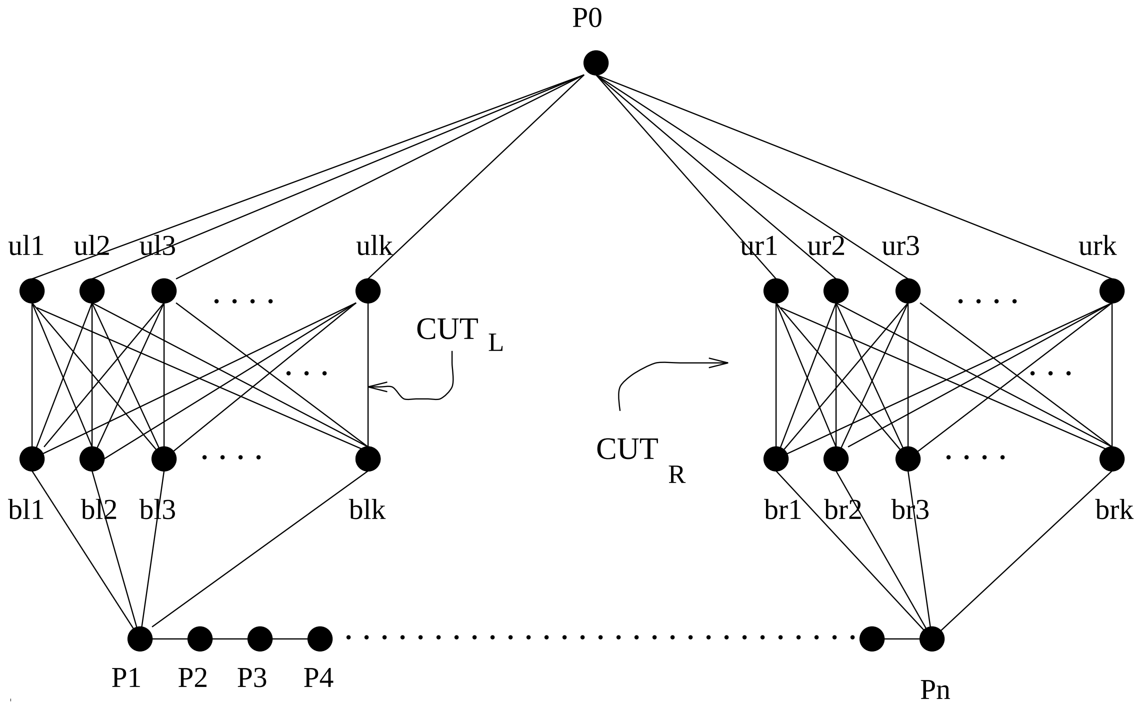

Figure 2: The network $G(n,m)$, $m = k^2$.

Theorem 1 *For every n and $m, m = O(n^2)$, there is a safe network with $O(n)$ processors and $O(m)$ links on which the worst-case message complexity of any global computation is $\Omega(m \cdot \log n)$ messages.*

Our lower bounds extend the ideas appearing in the lower bounds of [10]. The proof there is rather complicated and is not easily extended to a general network. We managed to find a simpler argument for the ring lower bounds, which we were then able to generalize to denser networks.

To prove the theorem for specific n and m, we construct a communication network $G(n,m)$ - with $O(n)$ processors and $O(m)$ links - on which we demonstrate our lower bound. This lower bound is achieved even for protocols where the processors are assumed to know the network topology (hence they know n).

The strategy of the lower bound proof employs an adversary having control on the delay of any message sent by the global computation protocol to construct a scenario in which any global computation on $G(n,m)$ sends $\Omega(m \log n)$ messages before it terminates. The scenario is constructed in $\log n$ phases in each of which the protocol sends $\Omega(m)$ messages.

In the following we describe $G(n,m)$ and then the strategy of the adversary by which the scenario is constructed.

As is shown in Figure 2, $G(n,m)$ has $(n + 4 \cdot \lceil\sqrt{m}\ \rceil + 1)$ processors and $(2 \cdot m + 4 \cdot \lceil\sqrt{m}\ \rceil + n - 1)$ links. It is composed of a path of processors - $p_1, \ldots, p_n$ - separated from the initiating processor p_0 by 2 bipartite graphs, the first of which (denoted B_l) connecting p_0 and p_1 , and the second (denoted B_r) connecting p_0 and p_n. Each such bipartite graph contains 2 sets of $\lceil\sqrt{m}\ \rceil$ vertices each, which are interconnected by m links. The nodes of B_l (B_r) are composed of the groups tl_i (tr_i) and bl_i (br_i), $i = 1 \ldots \lceil\sqrt{m}\ \rceil$, where tl (tr) stands for top-left (top-right) and bl (br) stands for bottom-left (bottom-right). In addition, p_0 is connected to processors tl_i, tr_i, $i = 1 \ldots \lceil\sqrt{m}\ \rceil$, p_1 is connected to processors bl_i, $i = 1 \ldots \lceil\sqrt{m}\ \rceil$ and p_n is connected to processors br_i, $i = 1 \ldots \lceil\sqrt{m}\ \rceil$.

We refer to processors $p_i, i = 1, \ldots, n$ as *Path processors*.

We partition the links into three classes, according to the way the adversary schedules messages sent over them:

- *Path-links*: These are the links along the path from p_1 to p_n. Processor $p_i, i = 1 \ldots n-1$ is connected with processor p_{i+1} by link e_i.
- CUT_l and CUT_r links: These are the links internal to the left and right bipartite subgraphs respectively, separating p_0 from the path processors.
- *Other*-links: Messages sent by the protocol on any other network-link are delivered by the adversary immediately with 0-delay.

The adversary proceeds in phases. Every phase has an associated pair of 2 blocked path links, on which messages may not pass during that phase:

Definition 1 *A blocked pair is a pair of 2 path-links* $\{e_l, e_r\}$ *, on which all messages are delayed throughout a phase. The links* e_l, e_r *are termed the blocked links of the phase. Messages delayed on a blocked link will be received only if and when that link is unblocked by the adversary at a subsequent phase. We denote the blocked pair associated with phase* i *by* BP_i.

Messages sent during phase i on path-links not in BP_i are delivered with 0 delay.

A correct global computation may not terminate before all of the network processors have joined the computation, because otherwise their local values can not be taken into account. In particular, all of the path-processors should join the computation. For a processor other than p_0 (the initiator) to join the computation, that processor must receive a message of the protocol. In the special case of the path of processors, the set of nodes which have not yet joined the computation constitutes a continuous segment of processors. Such a segment is associated with every phase of the adversary, as follows:

Definition 2 *The unvisited segment associated with phase i of the adversary - denoted by UV_i - is a range of processor-indices: $[l, r], 1 \leq l \leq r \leq n$, such that at the beginning of the i'th phase, processors $p_l, \ldots, p_r$ are the only path-processors that have not yet joined the computation. We denote by $LEN(UV_i) = r - l + 1$ the number of processors in UV_i.*

Every phase of the adversary terminates when the network reaches a *stable state*:

Definition 3 *A stable state is a state of the network in which the only messages of the global computation in transit are on links of the current phase's blocked pair, and each of the 2 blocked links has at least one delayed message on it.*

Phase i, $i \geq 1$ starts from a stable state, where the only delayed messages (if any) are on BP_{i-1}. We define BP_0 - the blocked pair before the protocol starts - to be the empty set.

When p_0 receives an external signal to start the computation, the adversary starts to delay messages on path-links $e_{\lfloor n/4 \rfloor}$ and $e_{\lfloor 3n/4 \rfloor}$ (i.e. $BP_1 = \{e_{\lfloor n/4 \rfloor}, e_{\lfloor 3n/4 \rfloor}\}$). When phase i reaches a stable state, assuming that this is not the last phase, the adversary extracts one of the two links in BP_i and replaces it with the middle link of the unvisited segment UV_{i+1}. Later we show how to select the extracted link in a way that ensures the required complexity.

In every phase, the adversary applies the following rule for scheduling messages sent over CUT links:

Definition 4 *Delay rule for CUT_l, CUT_r links: When a phase starts, all the links of CUT_l and CUT_r are blocked by the adversary until there are messages in transit on all the links of either CUT_l or CUT_r. When one of the 2 edge-cuts is saturated, then the edges of both CUT_l and CUT_r are unblocked until the phase terminates.*

The following lemma proves that every phase reaches a stable state and that the blocked pair of a phase determines the borders of the unvisited segment of the next phase.

Lemma 3.1 *Let $BP_i = \{e_l, e_r\}$ be the blocked pair in the beginning of phase i, then: (1) Phase i reaches a stable state and (2) $UV_{i+1} = [l+1, r]$.*

Proof: The proof goes by induction over the phases. It relies on the invariance that $BP_i = \{e_l, e_r\}$ is a link-cut that prevents path-processors $p_{l+1} \dots p_r$ from joining the computation during phase i. Let us prove for the first phase - phase 1. Phase 1 starts at p_0, i.e. $UV_1 = [1..n]$. Remember that for phase 1 we arbitrarily set BP_1 to be $\{e_{\lfloor n/4 \rfloor}, e_{\lfloor 3n/4 \rfloor}\}$ and therefore processors $p_{\lfloor n/4 \rfloor+1} \dots p_{\lfloor 3n/4 \rfloor}$ may not join the computation during phase 1. Assume that phase 1 does not reach a stable state, then there are 2 cases: (a) The protocol sends an infinite number of messages in phase 1 without ever reaching nodes $v_{\lfloor n/4 \rfloor+1} \dots v_{\lfloor 3n/4 \rfloor}$ and the lower bound is proved. (b) Phase 1 ends in a state in which there are no more messages in transit over links not in BP_1, and there are no delayed messages on at least one of the 2 links of BP_1 (i.e. Phase 1 ends in a state in which only on one link of BP_1 there are messages in transit). In this case the adversary may fail-stop all the links that have delayed messages on them. The network remains connected since the adversary has fail-stopped one path-link at most (the link of BP_1 that had messages in transit over it, if any) which could not have disconnected $G(n, m)$. As for links of CUT_l or CUT_r, from the delay rule for these links, when one of the 2 cuts is saturated all its links are unblocked, and therefore these links also may not disconnect the network. Consequently the network remains connected without any delayed messages, and $p_{\lfloor n/4 \rfloor+1}, \dots, p_{\lfloor 3n/4 \rfloor}$ did not join the computation - a contradiction to the protocol's correctness.

To prove part (2) of the Lemma for phase 1, note that processors $p_{\lfloor 3n/4 \rfloor+1}, \dots, p_n$ and $p_1, \dots, p_{\lfloor n/4 \rfloor}$ *have* received messages from the protocol since there are delayed messages on $e_{\lfloor n/4 \rfloor}$ and $e_{\lfloor 3n/4 \rfloor}$ which must have arrived through these nodes.

As for phase $i, i > 1$, from the induction hypothesis, phase $i-1$ reaches a stable state, in which the only delayed messages are on the links of $BP_{i-1} = \{l_{i-1}, r_{i-1}\}$ and

$UV_i = [l_{i-1}+1, r_{i-1}]$. Since $BP_i = \{l_i, r_i\}$ is constructed from BP_{i-1} by replacing either l_{i-1} or r_{i-1} by the middle link of UV_i, it follows that processors $p_{l_i+1}, \ldots, p_{r_i}$ which did not join the computation during the previous $i-1$ phases cannot join it also in phase i. The rest of the proof for phase i is identical to that of phase 1.

∎

It follows that there are $\log n$ phases:

Corollary 3.2 . *There are $\lfloor \log n \rfloor$ adversary phases.*

Proof: Let $BP_i = \{l_i, r_i\}, BP_{i+1} = \{l_{i+1}, r_{i+1}\}$ be the blocked-pairs of phases $i, i+1$ respectively. From the way the adversary constructs BP_{i+1} from BP_i: $\lfloor LEN([l_i, r_i])/2 \rfloor + 1 \geq LEN([l_{i+1}, r_{i+1}]) \geq \lfloor LEN([l_i, r_i])/2 \rfloor - 1$. Combining this with part (2) of Lemma 3.1 we get: $\lfloor LEN(UV_i)/2 \rfloor + 1 \geq LEN(UV_{i+1}) \geq \lfloor LEN(UV_i)/2 \rfloor - 1$. Since $UV_1 = [1, n]$ the result follows.

∎

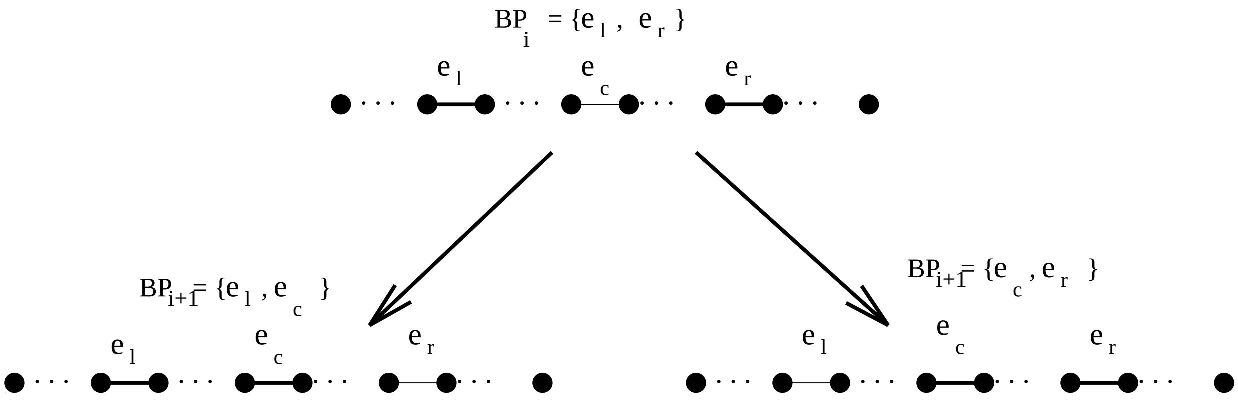

Figure 3: The 2 possibilities of the adversary for constructing BP_{i+1}: At least one of them will require communication between p_0 and the path-processors.

Until now we demonstrated that the adversary proceeds in $\lfloor \log n \rfloor$ phases, each of which terminating in a stable state. In the following we prove that the protocol is forced to send at least m messages in every phase.

When phase i terminates and a stable state is reached, the adversary has to select which of the two links in $BP_i = \{e_l, e_r\}$ to replace by the middle link of $[l+1, r]$ (see

Figure 3). Let us now specify the criterion by which the adversary selects which link to unblock. The link is selected so that before a stable state is reached again (i.e. before phase $i+1$ terminates) a message is received by p_0. The following Lemma proves, that at least one of the two links meets this criterion. [1]

Lemma 3.3 *Let $BP_i = \{e_l, e_r\}$, $LEN[l+1, r] > 1$ (i.e. phase i is not the last phase) and $c = \lfloor (l+r+1)/2 \rfloor$, then by setting BP_{i+1} to be either $\{e_l, e_c\}$ or $\{e_c, e_r\}$ a message will be received at p_0 before phase $i+1$ terminates.*

Proof: Let us assume the contrary. Let us denote by R_l (R_r) the sequence of events which results from setting $BP_{i+1} = \{e_l, e_c\}$ ($\{e_c, e_r\}$), starting from the stable state at the end of phase i, until a stable state is reached again. According to our assumption, the only processors in which events may occur in this sequence of events are processors $p_1, \ldots, p_c$ and the processors of B_l ($p_{c+1} \ldots, p_n$ and processors of B_r). This is true, because processors $p_1 \ldots p_c$ and the processors of B_l are separated from processors $p_{c+1} \ldots p_n$ and the processors of B_r by link e_c and by processor p_0.

Consequently R_l and R_r are independent sequences of events not affecting one another, and therefore we may schedule their events in any mixed order and the resulting network state will be the same. Hence, the adversary may block e_c, unblock *both* e_l and e_r and a stable state will be reached without any event occurring at p_0.

When that stable state is reached, the adversary may fail-stop e_c and any links of CUT_l and CUT_r that may have delayed messages on them without causing $G(n,m)$ to become disconnected. Since there are no other links with messages in transit, it follows that the protocol has terminated without all processors reporting to p_0, a contradiction to its correctness. ∎

Corollary 3.4 . *During every phase of the adversary, the protocol sends at least m messages.*

---

[1] This is the main point where our proof simplifies the proof of [10]. In [10], an event occurring in the initiating node is termed a *trigger*. Roughly, the proof there tries to schedule messages so that a trigger will occur with a minimum decrease in the length of the unvisited segment. The proof there also tries to capture each and every trigger. In order to do so a rather delicate analysis is required. However, to prove the lower bound it suffices to prove that *one* trigger at least occurs between each halving of the unvisited segment. Therefore we set the links that will be blocked in a phase and prove that one trigger occurs during it, rather than try to capture the next 'minimum' trigger.

Proof: From the delay rule for CUT_l & CUT_r links, no communication is possible from any path-processor to p_0 during a phase, unless either CUT_l or CUT_r is saturated during that phase. According to Lemma 3.3 such a communication must take place before the phase terminates and according to Lemma 3.1 the phase indeed terminates. Since $|CUT_l| = |CUT_r| = m$, and since the phase starts from a stable state, where there are NO messages in transit on links of CUT_l and CUT_r it follows that the protocol sends messages over all the links of either CUT_l or CUT_r during the phase. ∎

Now the proof of the theorem may be concluded:

Proof: According to Corollary 3.4, every phase of the protocol sends at least m messages and from Corollary 3.2 there are $\lfloor \log n \rfloor$ phases. ∎

Corollary 3.5 . *The worst-case message complexity of any global computation on a complete network is* $\Omega(n^2 \cdot \log n)$

Proof:

$G(n, m)$ may be mapped into a $(5 \cdot n + 1)$-node complete network. ∎

4 Unknown network size

In this section we prove the lower bound on global computations in the case that n is unknown a-priori; However, as [10] point out, we have to assume that every processor has a-priori knowledge about the identities of its neighbors, otherwise termination detection is not possible.

We extend [10] in a simple manner to achieve the following result:

Theorem 2 *Let A be a global computation protocol operating without a-priory knowledge of the number of processors, then for every n and $m, m = O(n^2)$, there is a safe network with $O(n)$ processors and $O(m)$ links on which A may be forced to send $\Omega(m \cdot n)$ messages.*

Proof: We use an adversary similar to that used in section 3 and as before, we consider a computation of A on $G(n, m)$. This adversary also blocks 2 path-links in every phase. However, when a phase terminates, it replaces one of the 2 links of BP_i not with the middle link, but with its neighboring link. Therefore, we need the following Lemma:

Lemma 4.1 *Let $BP_i = \{e_l, e_r\}$ be the blocked-links set of phase i. Let us also assume that phase i is not the last phase (i,e, $LEN[l+1, r] > 1$), then by setting BP_{i+1} to be either $\{e_{l+1}, e_r\}$ or $\{e_l, e_{r-1}\}$ a message is received at p_0 before phase $i+1$ terminates.*

Proof: Since A doesn't have a-priory knowledge about the number of processors in the network and since at the end of phase i processors $p_{l+1} \ldots p_r$ have not yet joined the computation, as far as A knows the network on which it is running may be either one of the networks $G(k, m), k \geq (n - r + l + 2)$.

Let us now assume to the contrary that in both cases - by setting BP_{i+1} to be $\{e_{l+1}, e_r\}$ or by setting it to be $\{e_l, e_{r-1}\}$ - the phase terminates without any message being received at p_0. Then, by exactly the same argumentation we used in Lemma 3.3, we may unblock both e_l and e_r and set $BP_{i+1} = \{e_{l+1}, e_{r-1}\}$ and the network will reach a stable state without p_0 receiving any message. However, this implies that A would have failed on $G(n - r + l + 2, m)$, because on that network processors p_{l+1} and p_r are neighbors and therefore e_{l+1} and e_{r-1} are actually the same link and the adversary may fail-stop it and prove the protocol incorrect. ∎

By Lemma 4.1, the adversary may cause $\Omega(n)$ phases. By Corollary 3.4, during every such phase A sends at least m messages and the theorem is proven. ∎

This lower bound is tight, since - as [10] point out - every processor may broadcast a message including its value, its id and the list of neighbor-ids as soon as it joins the computation. Thereafter every processor is able to detect when it had received all the network values (as in algorithm CT2 in [9]). This process requires $O(n \cdot m)$ messages.

Note that Theorems 1 and 2 are the two extremes in regard to the "amount" of a-priori knowledge available to the global computation protocol concerning n - the number of the network processors. Theorem 1 assumes an exact a-priori knowledge of n (and consequently the lower bound there is lower) whereas Theorem 2 assumes no such a-priori knowledge (and consequently the lower bound there is tight). However, there may also be intermediate cases - falling in a continuum between these two extremes - in which the protocol has only partial knowledge concerning n. Here we show that the same result as in Theorem 2 applies also in the case that the processors a priori know that n is at least L and at most kL for some constant k, $k > 1$.

Theorem 3 *Let A be a global computation protocol for arbitrary networks in which $L \leq n \leq kL$, for some constant k greater than one. Then the worst-case message complexity of A is $\Omega(m \cdot n)$.*

Proof outline: The conditions of Theorem 1 hold until the initiator knows about the first L processors whereas the conditions of Theorem 2 hold for the remaining $(k-1)L$ processors. Consequently A may be forced to send
$\Omega(m \cdot (\log L + L(k-1)))$ messages on $G(n, m)$. ∎

5 Conclusions and open problems

Global computation is a significant general set of distributed problems. This work strengthens the conjecture that in general dynamic networks, these problems have an $\Omega(m \cdot n)$ worst-case message-complexity. This conjecture, which has been proved for unidirectional dynamic networks (albeit with a Polylogarithmic slack) is now proven also for bidirectional dynamic networks, where the network size is not a-priori known; Still, the problem remains unsolved for the most interesting case:

What is the worst-case message complexity of global computation in a general dynamic network, where the network size is known a-priori?

We have considered for some time the *bit-complexity of the global computation* problem, i.e. the worst case bit-complexity of global computations, as opposed to the *message-complexity global computation* problem with which this work concerned itself. However, it seems that the difficulty of global computation is (almost?) entirely in its message-complexity, since the upper bound of bit-complexity global computation is $O(m \cdot n \cdot \log n)$ adding only a factor of $\log n$ to the $m \cdot n$ upper-bound of the message-complexity global computation.

Note, that global computation is not the only problem in general dynamic networks that has an upper bound of $\Omega(m \cdot n)$ messages and a much smaller lower bound. The *end-to-end* problem has the same upper bound [13] and only a trivial lower bound of $\Omega(m)$ messages. We believe that the tasks of finding lower bounds to these two problems face similar difficulties, but we were not able to find a reduction from the global computation lower bounds to corresponding lower bounds for the end-to-end problem. It is easy to see that end-to-end may be accomplished by a global-computation protocol (and so global computation is harder) but we found no reduction in the opposite direction. Therefore we pose the following question:

Is there a reduction from the global computation lower bounds to corresponding lower bounds for the end-to-end problem?

It seems that there is an inherent difficulty for processors in dynamic networks to distribute and coordinate efficiently a global computation. This difficulty probably stems from the fact that from the time a message is sent until it is received, the network topology may change completely without the processors being aware of the change. This difficulty manifests itself in the $m \cdot n$ upper bounds, which are a result of algorithms that basically send a flood of messages over the whole network (costing m messages) for each and every node without any significant inter node coordination. We believe that clearing this difficulty is fundamental to a better understanding of dynamic networks.

Acknowledgments

We thank Oded Goldreich for pointing out to us that the $\Omega(m \cdot n)$ lower bound holds even if the protocol has a linear a-priori bound on the network size. We also thank Yishay Mansour, Moty Ricklin and Mike Saks for helpful discussions.

References

[1] G. N. Frederickson and N. A. Lynch. Electing a leader in a synchronous ring. *Journal of The ACM*, 34(1):98–115, January 1987.

[2] J. Pachl, E. Korach, and D. Rotem. A technique for proving lower bounds for distributed maximum-finding algorithms. In *Proc. of the 14th Ann. ACM Symp. on Theory of Computing*, pages 378–382, 1982.

[3] G. N. Frederickson and N. A. Lynch. A general lower bound for electing a leader in a ring. Technical report, Purdue University, March 1985. CSD-TR-512.

[4] O. Goldreich and L. Shrira. The effect of link failures on computations in asynchronous rings. In *Proc. of the ACM Symp. on Principles of Distributed Computing*, August 1986.

[5] N. Linial. Distributive graph algorithms - global solutions from local data. In *Proc. of the 28th IEEE Ann. Symp. on Foundation of Computer Science*, pages 331–335, October 1987.

[6] I. Cidon and Y. Shavitt. Message terminate algorithms for rings of unknown size. Technical report, Technion, August 1991.

[7] O. Goldreich and L. Shrira. Consultation in the presence of faults: Two lower bounds. Technical report, Technion, Februar 1985.

[8] Y. Afek, G. M. Landau, B. Schieber, and M. Yung. The power of multimedia: Combining point-to-point and multiaccess networks. *Information and Computation*, 84(1):97–118, January 1990.

[9] A. Segall. Distributed network protocols. *IEEE Trans. on Information Theory*, IT-29(1):23–35, January 1983.

[10] O. Goldreich and L. Shrira. On the complexity of computation in the presence of link failures: the case of a ring. *Distributed Computing*, 5, 1991.

[11] O. Goldreich and D. Sneh. On the complexity of global computation in the presence of link failures: the case of uni-directional faults. In *Proc. 11th ACM Symp. on Principles of Distributed Computing*, pages 103–111, August 1992.

[12] Y. Afek and E. Gafni. End-to-end communication in unreliabel networks. In *Proc. of the 7th ACM Symp. on Principles of Distributed Computing*, pages 131–148, August 1988.

[13] Y. Afek, , and E. Gafni. Bootstrap network resynchronization: An efficient technique for end-to-end communication. In *Proc. of the Tenth Ann. ACM Symp. on Principles of Distributed Computing (PODC)*, August 1991.

Danny Hendler was born in Kiryat-Haim near Haifa, Israel, on April 17th 1961. Received his B.Sc. and M.Sc. in Computer Science from Tel-Aviv University, Israel, in 1986 and 1993, respectively. In the past 8 years he works as a free lance software-consultant, specializing mainly in communication, telephony and voice-mail applications.

Yehuda Afek received a B.Sc. in Electrical Engineering from the Technion and an M.Sc. and Ph.D. in computer Science from the University of California, Los-Angeles. In 1985 he joined the Distributed Systems research Department in AT&T Bell Laboratories as a Member of Technical Staff. In 1988 he joined the Computer Science Department in Tel-Aviv University where he now holds a permanent position. From 1989 to 1994 he was also a consultant for AT&T Bell Laboratories. His interests include communication protocols, distributed computing and asynchronous shared memory systems.