Fault-tolerant sequential scan

Paola Flocchini [*§] Andrzej Pelc [†§] Nicola Santoro [‡§]

Abstract

We consider the fault-tolerant version of the *sequential scan* problem. A line of identical cells has to be visited by a scanning head. The head can only distinguish an end of the line from an internal cell but can distinguish neither one end from the other, nor one internal cell from another. When the head starts at an internal cell, its first move is in a direction chosen by the adversary. When the head comes to an internal cell from a neighbor, it has two possible moves: *forward*, which means "go to the other neighbor", and *back* which means "return to the previous neighbor". At this point the adversary can place a *fault* whose effect is the change of the motion direction (going forward instead of back and vice-versa). The head is not aware of the occurrence of a fault.

The execution *cost* of a sequential scan algorithm for a line of length n in the presence of at most k faults is the worst-case number of steps that the head must perform in order to scan the entire line. The worst case is taken over all adversary's decisions. We consider two scenarios: when the length of the line is known to the algorithm and when it is unknown. Our goal is to construct sequential scan algorithms with minimum execution cost. We completely solve this problem for known line size. For any parameters k and n we construct a sequential scan algorithm, analyze its complexity and prove a matching lower bound, thus showing that our algorithm is optimal. The problem of fault-tolerant sequential scan for unknown line size is solved partially. For any parameter k we construct a sequential scan algorithm which explores a line of length n with cost $2kn + o(kn)$, for arbitrary n. For $k = 1$ our algorithm is shown to be optimal. However, we also show an alternative algorithm that has cost at most $O(kn)$ (with a constant larger than 2) for any n and cost $kn + o(kn)$ (which is asymptotically optimal) for infinitely many n. Hence the asymptotic performances of the two algorithms, for unbounded k and n, are incomparable.

[*]SITE, University of Ottawa, Ottawa, ON K1N 6N5, Canada. E-mail: `flocchin@site.uottawa.ca`

[†]Département d'informatique, Université du Québec en Outaouais, Gatineau, Québec J8X 3X7, Canada. E-mail: `pelc@uqo.ca`. Partially supported by the Research Chair in Distributed Computing at the Université du Québec en Outaouais.

[‡]School of Computer Science, Carleton University, Ottawa, Ontario, K1S 5B6, Canada.
E-mail: `santoro@scs.carleton.ca`

[§]Partially supported by NSERC discovery grant.

1 Introduction

Reading all, possibly identical entries in a linear array is a fundamental task arising in many applications. For example, in the write-all problem [9], all zeroes in a table with binary entries have to be replaced by ones; in the case when the table initially contains only zeroes, this task is equivalent to visiting all of its (identical) entries. The problem of finding in an array the first (last) position containing a non-zero entry was studied in [8]. In the case of an array of zeroes with only the first and last entries equal to 1, finding these positions again requires reading an entire linear array of identical entries. The list-ranking problem (cf., e.g., [11]), requiring finding the distance of every element of a linked list from its head, also requires visiting all (possibly identical) elements of a list. While in the above problems the issue was to optimize the execution of the task in parallel, the nature of other applications requires *sequential* scanning of a linear array of identical objects. Such is the case, for example, when a doubly linked list of identical objects is given and both ends of the list have to be found by a sequential algorithm starting from any position of it. Likewise, sequential scanning is required when a scanning head has to read all, possibly identical, entries in cells of a tape. In network exploration, a mobile agent (robot) has to explore a graph by visiting all of its nodes starting from any node. If the graph is a path, exploration is equivalent to sequential scanning of a linear array. In the case of anonymous graphs all entries of the array are identical. Efficient exploration of paths by a mobile agent was studied, e.g., in [4].

Let us consider the task of network exploration by a robot in more detail, and concentrate on the case of the path. A robot starts in an unknown node of the path and may or may not know its length. The robot's task is to visit all nodes of the path. Nodes are anonymous, and hence the robot can only distinguish an endpoint from an internal node: all internal nodes look identical and both endpoints look identical. If the starting node is internal, the robot starts from it in an arbitrary direction (since the robot does not know the distances to both ends of the path, both directions look the same, and thus the choice of the initial direction is made by the adversary). Then, at each internal node, the robot can either move forward (continue in the same direction), or back. Due to possible faults in the controls of the robot, the decision to go forward or back may be sometimes subject to error: if a fault occurs, the robot supposed to go forward goes back and vice-versa. Since all internal nodes look identical, the robot often does not realize that a fault occurred. This fault-prone application of the *sequential scan* problem is one of the motivations of our paper.

We formulate our scenarios and the problem itself in an abstract way to make it suitable for a broader range of applications. Consider a line of *cells* which have to be visited by a mobile entity called a *scanning head*. All cells are identical except the two ends of the line which are called the left and the right *walls* and are denoted by L and R, respectively. All other cells are called *internal*. The head can only distinguish a wall from an internal cell but can neither distinguish one wall from the other, nor one internal cell from another. (The names left and right and symbols L and R are used only for convenience of description.) If the head is at a wall, its only possible move is towards the other wall. When the head starts at an internal cell, its first move is in a direction chosen by the adversary. This reflects the assumption that the head is not aware of its position on the line and has no "sense of direction". When the head comes to an internal cell from a neighbor, it has two possible moves: *forward*, which means "go to the other neighbor", and *back* which means "return to the previous neighbor". At this point (before the actual move) the adversary can place a *fault* whose effect is the change of the motion direction: if the original move was *forward*, a fault causes the head to return to the neighbor from which it came, and if the original move was *back*, a fault results in the move of the head towards the other neighbor. The head is not aware of the occurrence of a fault, unless it expects to get to an internal cell and hits a wall, or vice-versa. When the head is at a wall, a fault has no effect.

The execution *cost* of a fault-tolerant sequential scan (FTSS) algorithm for a line of length n in the presence of at most k faults is the worst-case number of steps that the head must perform in order to scan the entire line. The worst case is taken over all fault configurations, controlled by the adversary and, in the case when the starting cell is internal, over all possible positions of the starting cell and the two possible starting directions. There are two main scenarios: when the length of the line is known to the algorithm and when it is unknown. In each of them we consider the start at a wall and the start at an internal cell. We are interested in minimizing the execution cost of a FTSS algorithm in each case. More precisely, both versions of our problem are formulated as follows.

- **Fault-tolerant sequential scan with known line size:**

  Given positive integers k and n, find a fault-tolerant sequential scan algorithm with minimum execution cost, for a line of length n, in the presence of at most k faults, when the head starts at a wall (resp. at an internal cell).

- **Fault-tolerant sequential scan with unknown line size:**

  Given a positive integer k, find a fault-tolerant sequential scan algorithm with minimum execution cost, for a line of arbitrary length, in the presence of at most k faults, when the head starts at a wall (resp. at an internal cell).

Both for the known and for the unknown length of the line we assume that the upper bound k on the number of faults is known to the algorithm and that the location of faults is worst case. This is a standard approach used in fault-tolerance (cf., e.g., the survey [13] for fault-tolerant models concerning network communication, or the seminal paper [14] for multiprocessor fault diagnosis).

1.1 Our results

We completely solve the problem of fault-tolerant sequential scan for known line size, for any positive parameters k and n. Our main contribution for this version of the problem is the proof of correctness of a natural fault-tolerant sequential scan algorithm which turns out to be optimal. We then prove a matching lower bound that establishes the optimality of the algorithm. For even n the optimal cost is $(k+2)n-1$ when the start is at an internal cell and $(k+1)n$ when the start is at a wall. For odd n it is, respectively, $(k+2)n-k-1$ and $(k+1)n-k$.

The problem of fault-tolerant sequential scan for unknown line size is solved partially. For any number k of faults we construct a fault-tolerant sequential scan algorithm which performs the scan of a line of length n, for arbitrary n, with cost $2(k+1)n-2k-1$ when the start is at an internal cell and with cost $2(k+1)n-2k$ when the start is at a wall. We show that this cost cannot be improved for $k=1$ by establishing matching lower bounds in this case, for infinitely many n. It is natural to ask if these lower bounds generalize to an arbitrary number of faults. In other words, is our algorithm (asymptotically) optimal for arbitrary k and n? We show that this is not the case. For large k and n, the cost of our algorithm is asymptotically $2kn$. More precisely, it is $2kn+o(kn)$, when both k and n are unbounded. However, we also show an alternative algorithm that has cost at most $O(kn)$ (with a multiplicative constant larger than 2) for any n, and cost $kn+o(kn)$ (which is asymptotically optimal) for infinitely many n. Hence the asymptotic performances of the two algorithms, for unbounded k and n, are incomparable. It remains open if there exists a fault-tolerant sequential scan algorithm for unknown line size which has cost $kn+o(kn)$ for all k and n.

To the best of our knowledge the present paper is the first to consider algorithmic aspects of fault-tolerant exploration by a mobile entity in which faults concern moves of the entity, rather than the environment.

1.2 Related work

The previously mentioned problems: write-all [9], finding in an array the first (last) position containing a non-zero entry [8], and list-ranking [11], are examples of tasks involving scanning a linear array or list of possibly identical objects. Unlike in our case, in these papers the emphasis was on efficient parallel execution of the respective tasks.

Sequential scan is closely related to the problem of network exploration by a mobile agent (robot). In the latter problem the agent has to visit all nodes and traverse all edges of an unknown graph. This problem has been studied both for directed [1, 2] and undirected [7, 12] graphs. In particular, in [4] the authors investigated the problem of network exploration using an imperfect map: the agent is provided with an unlabeled isomorphic copy of the undirected graph underlying the network but does not have any sense of direction. In the case of the line this setting is equivalent to sequential scan with known line size (in the fault-free scenario), because having an unlabeled map of the line is equivalent to knowing its length. The quality measure studied in [4] was the *overhead* of an exploration algorithm, defined as the worst case ratio of the time (number of steps) spent by an algorithm having the imperfect map to the optimal time of exploration assuming full knowledge of the graph. It turned out that, even for the line in the fault-free case, finding an exploration algorithm minimizing the overhead is far from trivial. It was proved in [4] that the best possible overhead for the line is $\sqrt{3}$, and an optimal algorithm was constructed.

Our present problem can be viewed as an aspect of fault-tolerant network exploration. One of the well-studied issues in this domain concerns agent security. Protecting mobile agents from malicious hosts was investigated, e.g., in [15, 16, 17]. In [3, 5, 6] the problem of locating a black hole in a network was considered. A black hole is a highly harmful stationary process residing in a node of a network and destroying all mobile agents visiting the node, without leaving any trace. Another problem related to fault-tolerant network exploration was investigated in [10]. A robot, situated in a root of a tree and unaware of the location of faulty edges, has to explore the connected fault-free component of the root, by visiting all its nodes. For a given rooted tree, the overhead of an exploration algorithm was defined as the worst-case ratio (taken over all fault configurations) of its cost to the cost of an optimal algorithm which knows where faults are situated. The goal in [10] was to find exploration algorithms with minimum overhead.

In all the above problems faults concerned the environment, more precisely components of the underlying graph. This should be contrasted with our present approach where faults concern the moves of the exploring agent.

2 Terminology

In the entire paper k denotes an upper bound on the number of faults. It is fixed and known to FTSS algorithms. The length of the line (i.e., the number of its links) is denoted by n and could be known or unknown, depending on the scenario. The line to be scanned will be often viewed as a segment $[a, b]$ with the starting point at 0 and a and b the left and right walls (denoted L and R), respectively. The mobile entity (scanning head) performing the scan is called the *head* for short. We use the predicates *inside* and *at-wall* to mean that the head is at an internal cell, or at a wall, respectively. We say that the line has been *explored* if all of its cells have been visited by the head. In the formulation of our algorithms we use a subroutine *go-straight* which is a sequence of *forward* steps repeated until some condition is met. There are three such conditions: *hit* means that the wall has been hit, *hit*(x) means that the wall has been hit and exactly x *forward* steps were performed, and *nohit*(x) means that the head has performed x *forward* steps without hitting a wall. After the condition is met, the direction of the move of the head is reversed. The sequence of steps during the *go-straight* subroutine is called a *round.*

It should be noted that the head's movement in one direction in a single round happens only when there are no faults in this round. Such a round is called *correct.* With each fault during a round the actual direction of the move of the head changes. A maximal sequence of steps in one direction during a round is called a *stretch.* The length of a stretch depends both on the algorithm and on the fault configuration. Hence a round can be composed of many stretches.

3 Line of known size

3.1 Upper bounds

In this section we present FTSS algorithms for a line of arbitrary known size n and at most k faults. For even n the cost is $(k+2)n-1$ when the start is at an internal cell and $(k+1)n$ when the start is at a wall. For odd n it is, respectively, $(k+2)n-k-1$ and $(k+1)n-k$. We later establish lower bounds showing that these algorithms are optimal.

3.1.1 The even size

The algorithm is composed of $k+2$ rounds if the head starts inside the line, and of $k+1$ rounds if it starts at a wall. During a round the head moves in the same direction until either it hits a wall, or it performs $n-1$ steps (n if starting at a wall) without reaching any wall. At this point it reverses direction. Obviously the second condition means that at least one fault has occurred.

```
Algorithm KnownEven

if inside then count := $k+2$ else count := $k+1$;
repeat count times
    if inside then
        go-straight until (nohit($n-1$) OR hit)
    else /* at-wall */
        go-straight until (nohit($n$) OR hit)
    endif
    reverse direction
end
```

Consider an arbitrary execution of the algorithm. In this execution, let $f_i \geq 0$ be the number of faults occurring in round i; let $Z = \{i : f_i = 0\}$ be the set of correct rounds; let $F = \{i : f_i > 0\}$ be the set of rounds that contain at least one fault; let $E \subseteq F$ be the set of rounds that contain an even positive number of faults; finally let $z = |Z|$, $f = |F|$,

$e = |E|$, and $o = |F \setminus E|$. Let $dir(i)$ denote the direction (from R to L or from L to R) at the start of round i.

By construction, the execution of the algorithm has trivially the following property.

Lemma 3.1 *Consider round i, $1 \leq i \leq count$.*

1. *If round i is correct, the head hits a wall in this round.*
2. *If round i contains an even number of faults then the directions at the beginning of rounds i and $i+1$ are different; i.e., $dir(i+1) \neq dir(i)$.*
3. *If round i contains an odd number of faults then the directions at the beginning of rounds i and $i+1$ are the same; i.e., $dir(i+1) = dir(i)$.*

Lemma 3.2 *If in an execution there are two correct rounds such that all the rounds between them contain an odd number of faults, then the line is explored.*

Proof: Let i and j, $i < j$, be correct rounds such that all the rounds between them contain an odd number of faults. By lemma 3.1(1), the head hits a wall, say R, in round i and, by construction it starts round $i+1$ by moving towards the other wall L. Since in all the rounds $i+1, i+2, \ldots, j-1$ an odd number of faults occurs, then by lemma 3.1(3), $dir(i+1) = dir(i+2) = \ldots = dir(j-1) = dir(j)$; hence, in round j the head will move towards L and, since j is correct, will reach L. $\square$

We now show that the condition of the above lemma always holds if the head starts inside the line.

Lemma 3.3 *Let the head start inside the line. In any execution there are always two correct rounds such that all the rounds between them contain an odd number of faults.*

Proof: Let the head start inside the line; then the number of rounds is $k+2$. We need to prove that there exist $i, j \in Z$ such that for all l, with $i < l < j$, we have $l \notin E$. It is sufficient to prove that the number of correct rounds exceeds by at least two the number of rounds with an even number of faults; i.e., $z \geq e+2$. First notice that a round in E must contain at least two faults, a round in $F \setminus E$ must contain at least one; hence $k \geq o + 2e$. Moreover, since there are $k+2$ rounds, we have $z + e + o = k + 2$; i.e., $z = k - o - e + 2 \geq 2e - e + 2 = e + 2$. $\square$

In the case when the head starts at a wall, there is an additional property.

Lemma 3.4 *Let the head start at a wall. If in an execution all the rounds before the first correct one contain an odd number of faults, then the line is explored.*

Proof: Let the head start from a wall, say R, let j be the first correct round, and let all rounds $i < j$ contain an odd number of faults. By construction the head starts round 1 by moving towards the other wall L. Since in all the rounds $1, 2, \ldots, j-1$ an odd number of faults occurs, then by lemma 3.1(3), $dir(i) = dir(i-1) = L$; hence, in round j the head will move towards L and, since j is correct, will reach L. □

We now show that if the head starts at a wall, at least one of the conditions expressed by Lemmas 3.4 and 3.2 holds.

Lemma 3.5 *Let the head start at a wall. In any execution one of the following conditions holds:*

1. *All the rounds, if any, before the first correct one contain an odd number of faults.*
2. *There are two correct rounds such that all the rounds between them contain an odd number of faults.*

Proof: Let the head start at a wall, say R; in this case the number of rounds is $k+1$. We will prove that if condition (1) does not hold, then condition (2) does. Let $i > 1$ be the first correct round and let $p \geq 1$ preceding rounds contain an even number of faults. After one step in round i, the head is exactly in the situation of an head starting Algorithm KnownEven from the current cell with at most $k - i$ faults. The result then follows from Lemma 3.3.

□

As a consequence of Lemmas 3.2 - 3.5, we get:

Theorem 3.1 *Algorithm* KnownEven *allows the head to correctly explore any line of even and known size, with at most k faults, regardless of the starting point.*

Theorem 3.2 *During the execution of Algorithm* KnownEven*, the head performs at most $(k+2)n - 1$ steps if it starts inside the line, and at most $(k+1)n$ steps if it starts at a wall.*

Proof: Let x be the number of rounds starting at a wall and let y be the number of rounds starting inside the line. Clearly $x + y = count$. Every time a round starts with the head at a wall, the number of steps of that round is at most n; when a round starts with the head inside the line, the number of steps is at most $n - 1$. The total number of steps is at most $S(x) = xn + y(n-1) = xn + (count - x)(n-1)$.
If the head is initially inside the line, then $y \geq 1$ and thus $x < count$; moreover, according to the algorithm, $count = k + 2$. In this case $S(x)$ is maximized when $x = k + 1$; thus, the number of steps when starting inside the line is at most $(k+2)n - 1$.

If the head is initially at a wall, $S(x)$ is maximized when $x = count$. According to the algorithm, when starting at a wall, $count = k + 1$. Thus, the number of steps when starting at a wall is at most $(k + 1)n$.

□

3.1.2 The odd size

When the size of the line is known and is odd, the algorithm can exploit this fact by discovering termination conditions without having to perform a fixed number of rounds. Intuitively, in this case the odd parity of n allows to detect the absence/presence of failures during a round, thus allowing the algorithm to terminate sooner.

Whenever a round starts with the head at a wall, the head moves in the same direction until either it hits a wall, or it performs n steps without reaching any wall. This second condition is particularly important (as we will see later); the head has to remember its occurrence by setting a special flag. On the other hand, if the head finishes this round by hitting a wall in *exactly* n steps, then the algorithm terminates; this is due to the fact that a walk of odd length from wall to wall cannot hit the same wall, and thus must result in the exploration of the entire line. If the algorithm does not terminate, the head reverses its direction before proceeding to the next round.

Whenever a round starts with the head inside the line, the head moves in the same direction until either it hits a wall, or it performs $n - 1$ steps without reaching any wall. If the head finishes this round by hitting the wall in an *even* number of steps, then the head has to check whether the special flag mentioned above is set or not. As we will show later, if the flag is set, the algorithm can terminate. If the algorithm does not terminate, the head reverses its direction before proceeding to the next round.

```
Algorithm KnownOdd

halt:= flag:= 0;
repeat until halt=1
    if at-wall then
            go-straight until (hit OR nohit(n))
            if nohit(n) then flag:=1;
            if hit(n) then halt:= 1;
    else /* inside */
            go-straight until (hit OR nohit(n − 1))
            if ((hit(x) with x even) AND flag=1) then halt:= 1;
    endif
    reverse direction
end repeat
```

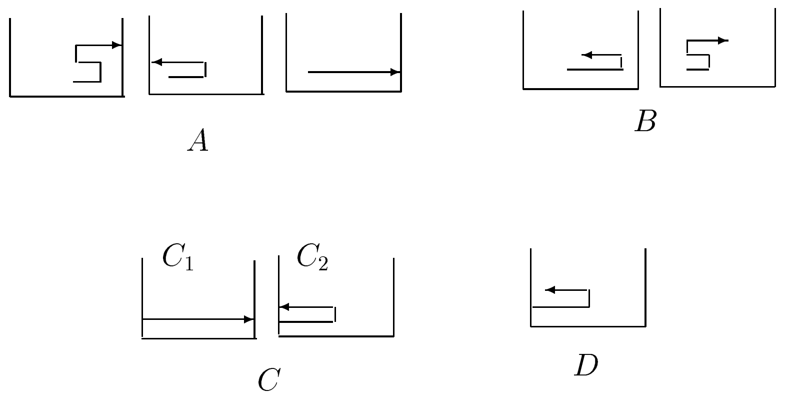

Figure 1: Examples of different types of rounds with various numbers of faults.

Correctness

Consider an arbitrary execution of the algorithm. This execution is composed of a sequence of rounds.

Depending where the head starts and ends a round, we have four possible types of rounds. In the following we enumerate all the possible situations.

A. *Inside/Wall.* A round is of type A when the head starts the round inside the line and ends it at a wall within at most $n-1$ steps. There are two subtypes of such a round:

A_1 : In this case, the predicate $hit(x)$ holds with x even.

A_2 : In this case, the predicate $hit(x)$ holds with x odd.

B. *Inside/Inside.* A round is of type B when the head starts and ends the round inside the line. It is the only round type in which the predicate $nohit(n-1)$ holds. In this round, there has been at least one fault. The round is composed of exactly $n-1$ steps.

C. *Wall/Wall.* A round is of type C when the head starts and ends the round at a wall. There are two subtypes of such a round:

C_1 : In this case, the predicate $hit(n)$ holds; this round is composed of exactly n steps and, as we will show, does not contain any fault.

C_2 : In this case, the predicate $hit(n')$ holds with $n' < n$; this round contains at least one fault. In the worst case there are $n-1$ steps.

D. Wall/Inside. A round is of type D when the head starts the round at a wall and ends inside the line. This is the only round type in which the predicate $nohit(n)$ holds, and there has been at least one fault. The round is composed of exactly n steps.

In Figure 1 are shown examples of: rounds of type A with two, one, and no faults; rounds of type B with one and two faults; rounds of type C with no faults and one fault; a round of type D with one fault.

Lemma 3.6 *After a round of type C_1, the line has been explored.*

Proof: Since n is odd, a round of type C_1 must be correct. In fact, a walk from wall to wall composed of an odd number of steps cannot hit the same wall. □

Lemma 3.7 *After a round of type D, the head is at an odd distance from the wall where it started that round.*

Proof: By definition, a round of type D starts from a wall, say L, and terminates after n steps with the head inside the line. Let this round contain m faults; thus, the movement of this round is composed of a sequence of $m+1$ stretches $s_0, s_1, \dots, s_m$, with $\sum_{i=0}^{m} |s_i| = n$. The distance d of the head from L can be calculated as follows: $d = \sum_{i=0}^{m}(-1)^i |s_i| = \sum_{i=0}^{\lfloor m/2 \rfloor} |s_{2i}| - \sum_{i=0}^{\lfloor \frac{m-1}{2} \rfloor} |s_{2i+1}|$. Let $S_1 = \sum_{i=0}^{\lfloor m/2 \rfloor} |s_{2i}|$and $S_2 = \sum_{i=0}^{\lfloor \frac{m-1}{2} \rfloor} |s_{2i+1}|$. Since $n = S_1 + S_2$ is odd, we know that either S_1 or S_2 is odd; but then also $d = S_1 - S_2$ must be odd. □

Lemma 3.8 *After a round of type B, if the head started at an odd distance from a wall, it will also end at an odd distance from the same wall.*

Proof: Let x be the initial distance of the head from wall L. Let this round contain m faults; thus, the movement of this round is composed of a sequence of $m+1$ stretches $s_0, s_1, \dots, s_m$, with $\sum_{i=0}^{m} |s_i| = n-1$. After this movement, the head is at distance $d = x + \sum_{i=0}^{m}(-1)^i |s_i|$ from L, if it starts the round moving towards R, and at distance $d = x - \sum_{i=0}^{m}(-1)^i |s_i|$, otherwise. As in the previous lemmas, let $S_1 = \sum_{i=0}^{\lfloor m/2 \rfloor} |s_{2i}|$ and $S_2 = \sum_{i=0}^{\lfloor \frac{m-1}{2} \rfloor} |s_{2i+1}|$ (thus, $d = x + S_1 - S_2$ if the head starts the round towards R, and $d = x + S_2 - S_1$ otherwise). Since $n-1$ is even, $S_1 + S_2$ is also even. But then both $S_1 - S_2$ and $S_2 - S_1$ are even. Since x is odd by hypothesis, the distance $x + S_2 - S_1$ (or $x + S_1 - S_2$) to L at the end of the round must also be odd. □

Lemma 3.9 *Consider a round of type A. Let x be the distance of the head from wall L at the beginning of the round, and let y be the number of steps of the round.*

a) If x is odd and y is even, then the head hits R.
b) If both x and y are odd, then the head hits L.
Moreover, in the latter case:
b_1) If the head starts by moving away from wall L, then round A contains at least one fault;
b_2) If the head starts by moving away from wall R and wall L has been already visited, then between the previous hit of the wall and the current there have been at least as many faults as the number of rounds.

Proof: Consider a round of type A. Let this round contain m faults; thus, it is composed of a sequence of $m+1$ stretches $s_0, s_1, \ldots, s_m$, with $\sum_{i=0}^{m} |s_i| = y$. Let $S_1 = \sum_{i=0}^{\lfloor m/2 \rfloor} |s_{2i}|$ and $S_2 = \sum_{i=0}^{\lfloor \frac{m-1}{2} \rfloor} |s_{2i+1}|$.
Case a). Let x be odd and y be even. By contradiction, let the head hit wall L. Since $S_1 + S_2$ is even, also $S_1 - S_2$ and $S_2 - S_1$ are even. The distance of the head from L after this round is either $x + S_1 - S_2$ (if stretch s_0 is towards R) or $x + S_2 - S_1$ (if stretch s_0 is towards L); that is, in both cases, it is odd and hence different from 0; thus, it is not wall L that is hit by the head.
Case b). Let both x and y be odd. By contradiction, let the head end this round at wall R. At the beginning of the round the head is at distance $n-x$ from R, thus $n-x = S_1 - S_2$ if the head is moving towards R, and $n - x = S_2 - S_1$ otherwise. Since $y = S_1 + S_2$ is odd, we know that $S_1 - S_2$ and $S_2 - S_1$ are also odd. However $n - x$ is even, which yields a contradiction.
Subcase b_1). If the head started the round moving away from L, there has been at least one fault during this round since it is terminating in L again.
Subcase b_2). In this case, between the previous and the current hit of wall L there has been one round of type D starting from L, possibly followed by several rounds of type B, and then by the round of type A that we are considering. We want to show that at least one of the above rounds that precede A contains more than one fault. Suppose, by contradiction, that each of them contains a single fault (recall that rounds of type B and D must contain at least one fault). Since after each of these rounds the direction is inverted, all these rounds, as well as round A, start with the head moving away from wall L. This is impossible because by hypothesis the head starts round A by moving away from R. This contradiction implies that at least one round must have more than one fault, in order to allow a change of direction. This implies that between the previous hit of the wall and the current one there have been at least as many faults as the number of rounds. □

Lemma 3.10 *Let the head execute a (possibly empty) sequence of rounds of type B, preceded by a round of type D. If the next round is of type A_1, then at the end of that round the line has been explored.*

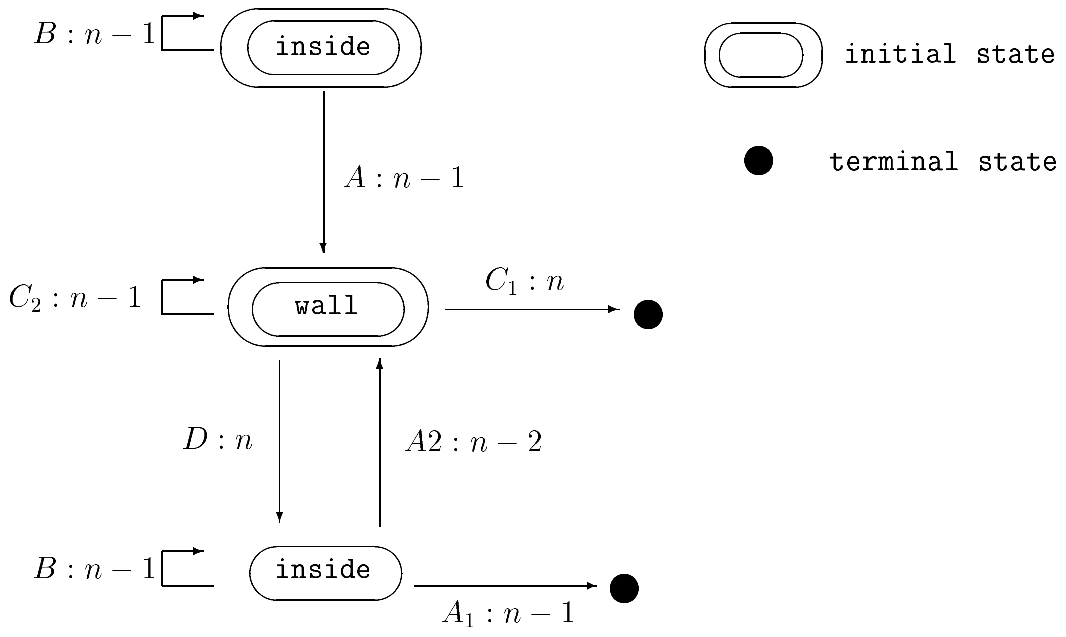

Figure 2: Possible transitions between rounds.

Proof: Let the head start a round of type D at wall L. By Lemma 3.7, after that round, the distance d of the head from L is odd. Let now the head execute a possibly empty sequence of rounds of type B. After this sequence of rounds (by Lemma 3.8 if the sequence is not empty and trivially if it is empty) the head is still at an odd distance from L. Let the next round be a round of type A_1. The proof now follows from Lemma 3.9 case a). □

Theorem 3.3 *Algorithm* KnownOdd *allows the head to correctly explore any line of odd and known size after at most* $k+2$ *rounds, if the head starts inside the line, and after at most* $k+1$ *rounds, if it starts at a wall.*

Proof: First suppose that the head starts inside the line. After the initial round, depending on its type, the head either hits a wall (type A, possibly containing faults) or is still inside the line after $n-1$ steps (type B, containing at least one fault). If still inside the line, the head can continue to remain inside after $n-1$ steps for several rounds (type B, containing at least one fault). Let p be the number of rounds before the head hits a wall for the first time; then at least $p-1$ of these rounds are faulty. Since the number of faults is at most k, after at most k rounds, the head will eventually hit a wall. If the head initially starts at a wall, the description of its behavior starts here.

Once at a wall (say L), two events can occur: either the head hits a wall again or it performs n steps and ends up inside the line.

Suppose that the head hits the wall again. If this happens after n steps, this is a round of type C_1 and, by Lemma 3.6, the line has been explored; notice that in the algorithm, the variable *halt* is set to 1 and the algorithm terminates. Otherwise, this is a round of type C_2, containing at least one fault.

If the round is finished with the head ending up inside the line after n steps, this is a round of type D and at least one fault has occurred; in the algorithm, when this round occurs, the variable *flag* is set to 1. Two possible situations can occur next: either the head continues to stay inside the line after performing $n-1$ further steps, or it hits a wall. In the first case this is a round of type B that contains at least one fault. The head may continue to experience several rounds of this type. In the second case, the head hits a wall. This is a round of type A. If this happens after an even number of steps (i.e., round of type A_1), then by Lemma 3.10, the line has been explored; notice that in the algorithm, in this case the variable *halt* is set to 1 (recall that the flag has been set to 1 in the previous type D round) and the algorithm terminates. Otherwise (if this happens after an odd number of steps), this is a round of type A_2 and the head is back at the starting wall.
The overall situation is summarized in Figure 2.

There are three types of correct rounds that could occur: C_1, A_1 and A_2. If a correct round of type C_1 occurs, the algorithm correctly terminates (the algorithm correctly sets the variable *halt* to 1). If a correct round r of type A_1 occurs, this round must have been preceded by a round of type D, and hence by Lemma 3.7 the distance of the head from L at the beginning of round r is odd, which implies that round r terminates at wall R after an even number of steps (ensuring that the algorithm correctly sets the variable *halt* to 1). We call a correct round of type A_1 or C_1 a *correct terminating round.* On the other hand, a correct round of type A_2 could send the head back to wall L. In such an event, however, we are guaranteed that in the s rounds between the last two hittings of the wall, there have been at least s faults (Lemma 3.9). We call a correct round of type A_2 a *correct non-terminating round.*

To conclude the proof, we need to show that a correct terminating round will occur within at most $k+1$ rounds, if the head started at a wall, and within at most $k+2$ rounds if the head started from inside the line.

Consider the two cases. If the head started at a wall, every round that terminates inside the line contains at least one fault, and every correct non terminating round implies that at least one earlier round between the current and the previous hit contains at least two faults (Lemma 3.9). This implies that within $k+1$ rounds at least one is a correct terminating round. Hence, the algorithm terminates after at most $k+1$ rounds. If the head started inside the line, there are p initial rounds until the head first hits a wall (possibly $p=0$, if the head started at a wall), of which at least $p-1$ contain a fault (if $p>1$). At this point there remain at most $k-(p-1)$ other faults. For the same reason as above, in the next $k-(p-1)+1$ rounds at least one is a correct terminating round. Hence, the algorithm terminates after at most $p+k-(p-1)+1=k+2$ rounds. □

Theorem 3.4 *During the execution of Algorithm* KnownOdd*, the head performs in the worst case* $(k+2)n-k-1$ *steps if it starts inside the line, and* $(k+1)n-k$ *steps otherwise.*

Proof: Any execution of the algorithm corresponds to a path in the graph of Figure 2. On each edge the type and the worst case number of steps of the corresponding round is indicated. Since we are interested in the worst case, we will only consider executions where the maximum number of steps is incurred in each round.

Let us call *cheap* a round composed of $n-2$ steps, *medium* a round composed of $n-1$ steps, and *expensive* a round composed of n steps.

Let $1, 2, \ldots, m$ be the rounds during an arbitrary execution of the algorithm. Let $\mathcal{E} = \{e_1, \ldots e_s\}$ denote the set of expensive rounds, and $\mathcal{C}$ denote the set of cheap rounds. Let y be the number of medium rounds, and z the number of cheap rounds. We first show that, if $s > 1$, then for all i, $1 \leq i \leq s-1$, there exists $c \in \mathcal{C}$, such that c occurs between e_i and e_{i+1}. The only rounds composed of n steps are terminal rounds of type C_1, or rounds of type D. Clearly $e_1, \ldots, e_{s-1}$ must be rounds of type D, thus terminating inside the line. After each round e_i $(1 \leq i \leq s-1)$, before the next expensive round e_{i+1}, a round of type A_2 must necessarily occur (see Fig. 2). In other words, between two expensive rounds there must be a cheap one, which implies that $s - 1 \leq z$.

Let us consider first the case when the head started inside the line. The total number of steps T is at most $sn+y(n-1)+z(n-2) = (s+y+z)(n-1)+s-z$. Since $s-z \leq 1$, we have $T \leq (s+y+z)(n-1)+s-z \leq (s+y+z)(n-1)+1$. If the head started inside the line we have $k+2$ rounds (Theorem 3.3) and then: $T \leq (k+2)(n-1)+1 = (k+2)n-k-1$. If the head started at the wall we have $k+1$ rounds and then: $T \leq (k+1)(n-1)+1 = (k+1)n-k$ $\square$

3.2 Lower bounds

In this section we establish lower bounds on the cost of fault-tolerant sequential scan, showing that the algorithms presented in the previous section are optimal.

Theorem 3.5 *For any FTSS algorithm for a line of known size n with at most k faults, there exists a starting point inside the line and an adversary that forces the head to perform at least* $(k+2)n-1$ *steps, if n is even, and* $(k+2)n-k-1$ *steps, if n is odd.*

Proof: Fix a FTSS algorithm $\mathcal{A}$ on a line of size n. Let position 0 correspond to the left wall. Consider a sequence $\alpha = (s_1, s_2, \ldots, s_k)$ coding the fault-free execution of $\mathcal{A}$ until a wall is hit for the first time, starting from $\lceil \frac{n-1}{2} \rceil$ (w.l.o.g, let the wall hit be L, a similar argument holds when the wall hit is R). The meaning of the sequence α is the following:

Go s_1 steps in one direction;
go s_2 steps in the other direction;
go s_3 steps in the first direction;
go s_4 steps in the other direction;
...

Let y be the rightmost point of α. Thus $\lceil \frac{n-1}{2} \rceil \leq y < n-1$.

Now consider a different scenario for the same algorithm: more precisely, we consider the same execution of algorithm $\mathcal{A}$ (i.e., the same sequence α) with a different starting point $\lceil \frac{n-1}{2} \rceil + n - y - 1$. In this scenario, the line starts $n-y-1$ positions before the end of α and ends one position after its rightmost point (obviously a wall is not hit during the execution of α). At least additional $n-y-1$ steps are required for hitting the left wall, and at least $y+1$ steps are required for hitting the right wall. Thus, the number of steps required to hit a wall for the first time in this scenario is $h \geq y + min\{y+1, n-y-1\}$. Since $y \geq \lceil \frac{n-1}{2} \rceil$, we have that $min\{y+1, n-y-1\} = n-y-1$, and, thus: $h \geq n-1$.

The head has now reached the (left) wall for the first time. Since the line is not yet fully explored, the head has to perform a walk of at least n steps eventually reaching the opposite wall in a fault-free execution of the algorithm. Consider such a walk. Let the adversary place a fault when the head is, for the last time, at distance $\frac{n}{2}$ from the left wall, if n is even, and at distance $\frac{n-1}{2}$, if n is odd. In this way the left wall is hit again after at least n steps, if n is even, and $n-1$ steps, if n is odd. Repeating the same argument k times we can conclude that the head is back at the left wall after performing, since the beginning of the execution, at least $n-1+kn$ steps, if n is even, and $n-1+k(n-1)$ steps, if n is odd. However, the line is not yet fully explored. Hence, the head must still perform at least n steps to reach the right wall, for a total of $(k+2)n-1$ steps, if n is even, and $(k+2)n-k-1$ steps, if n is odd. □

Theorem 3.6 *Let the head start at a wall. For any FTSS algorithm for a line of known size n with at most $k > 0$ faults, there exists an adversary that forces the head to perform at least $(k+1)n$ steps, if n is even, and $(k+1)n-k$ steps, if n is odd.*

Proof: Fix a FTSS algorithm $\mathcal{A}$ on a line of size n. Let the head start at the left wall. In any fault-free execution the head has to perform a walk of at least n steps eventually reaching the opposite wall. Consider such a walk. Using an argument similar to the one of Theorem 3.5, we have that the head must still perform at least n steps to reach the right wall, for a total of $(k+1)n$ steps, if n is even, and $(k+1)n-k$ steps, if n is odd.

□

4 Line of unknown size

4.1 An upper bound

In this section we present a FTSS algorithm working for unknown line size. If there are at most k faults, the head performs no more than $2(k+1)n-2k-1$ steps, if it starts inside the line, and no more than $n(2k+1)-2k$ steps, if it starts at a wall. Hence the asymptotic cost of the algorithm is $2kn+o(kn)$, for unbounded k and n. The algorithm is composed of $k+2$ or $k+1$ rounds depending on whether the head starts inside the line or at a wall. During each round the subroutine *go-straight* is executed until a wall is hit.

```
Algorithm UNKNOWN

if inside then count := $k+2$ else count := $k+1$;
repeat count times
    go-straight until hit
    reverse direction
end
```

Theorem 4.1 *Algorithm* UNKNOWN *allows the head to correctly explore any line, with at most k faults, without knowing its size.*

Proof: By construction, each round ends as soon as the head hits a wall. If the head started at a wall, the line is correctly explored the first time there is a correct round. Since there are $k+1$ rounds and at most k of them are faulty, the line will be correctly explored. Consider now the case when the head starts inside the line. Let the head hit R in the first round. If in the next round the head hits the other wall, the entire line is explored; else, at least one fault must have occurred since, otherwise, according to the algorithm, the head would have hit L. Inductively, if the head has not hit L in the first j rounds, $2 \le j \le k+1$, then at least $j-1$ faults have occurred. Since the total number of faults is k and the number of rounds is $k+2$, it follows that the line will be fully explored.
□

Theorem 4.2 *During the execution of Algorithm* UNKNOWN, *the head performs no more than $2(k+1)n-2k-1$ steps, if it started inside the line, and no more than $n(2k+1)-2k$ steps, if it started at a wall.*

Proof: Consider first the case when the head starts inside the line. The algorithm is composed of $k+2$ rounds. Let f_i denote the number of faults that occur during round i, with $1 \le i \le k+2$. Clearly $\sum_{i=1}^{k+2} f_i = k$ and $f_i \ge 0$.
The first round starts with the head inside the line and contains f_1 faults; it is thus

composed of f_1+1 stretches, the last of which hits the wall. None of the first f_1 stretches hits a wall. Thus, each of them is composed of at most $n-2$ steps. The last stretch starts inside the line and ends at a wall; hence, it is composed of at most $n-1$ steps. In other words, the head performs at most $(n-2)f_1+n-1$ steps in this round.
Any subsequent round i is also composed of f_i+1 stretches; the first and the last stretches have one extremity at a wall and the other inside the line; hence, they are composed of at most $n-1$ steps. None of the other stretches hits a wall and thus each of them is composed of at most $n-2$ steps. As a consequence, during round i, the head performs at most $2(n-1)+(f_i-1)(n-2)$ steps. The total number of steps is thus at most

$$S = S(f_1, f_2, \ldots, f_{k+2}) = (n-2)f_1 + n - 1 + \textstyle\sum_{i=2}^{k}(\ 2(n-1) + (f_i-1)(n-2)\ )$$

where $\sum_{i=1}^{k} f_i = k$ and $f_i \geq 0$. We have: $(n-2)f_1+n-1+\sum_{i=2}^{k+2}(2(n-1)+(f_i-1)(n-2)) = n-1+\sum_{i=2}^{k+2}(2(n-1)) - \sum_{i=2}^{k+2}(n-2) + (n-2)\sum_{i=1}^{k+2}(f_i) = (2(k+1)+1)(n-1)-(k+1)(n-2)+k(n-2) = 2(k+1)(n-1)+1 = 2(k+1)n-2k-1$

Consider now the case when the head starts at a wall. In this case, following the same reasoning, the total number of steps is at most

$$S = S(f_1, f_2, \ldots, f_{k+1}) = \textstyle\sum_{i=1}^{k+1}(\ 2(n-1) + (f_i-1)(n-2)\ )$$

where $\sum_{i=1}^{k-1} f_i = k$ and $f_i \geq 0$. We have: $\sum_{i=1}^{k+1}(2(n-1)+(f_i-1)(n-2)) = \sum_{i=1}^{k+1}(2(n-1)-(n-2)) + (n-2)\sum_{i=1}^{k+1}(f_i) = (k+1)n + (n-2)k = n(2k+1) - 2k$ □

4.2 The lower bound for one fault

In this section we prove that the upper bound from the previous section cannot be improved for $k=1$. We first consider the head starting inside the line.

Fix any FTSS algorithm $\mathcal{A}$ and consider the part of its fault-free execution until a wall is hit for the first time. This part can be coded in one of two possible ways.

- As an infinite sequence of integers $(s_1, t_1, s_2, t_2, \ldots,)$ with the following meaning:

  Go s_1 steps in one direction;
  Go t_1 steps in the other direction;
  Go s_2 steps in the first direction;
  Go t_2 steps in the other direction;
  ...

- As a finite sequence of integers $(s_1, t_1, s_2, t_2, \ldots, s_k)$ or $(s_1, t_1, s_2, t_2, \ldots s_k, t_k)$ with the following meaning:

Go s_1 steps in one direction;
Go t_1 steps in the other direction;
Go s_2 steps in the first direction;
Go t_2 steps in the other direction;
...
Go s_k steps in the first direction (resp. t_k steps in the other direction);
Go until hitting the wall in the other (resp. first) direction.

Call a FTSS algorithm that can be coded in the first (resp. second) way, a type 1 (resp. type 2) algorithm. Parts of the execution that correspond to integers s_i or t_i are called *swings*. The last swing of an algorithm of type 2 is called the infinite swing.

Consider the execution of a FTSS algorithm (of type 1 or type 2) in the infinite line in which the starting point is 0 and the first direction is positive. Hence the swing s_1 ends in point $b_1 = s_1$. Let $a_1 = 0$. Let a_i and b_i, for $i > 1$, be the left and right endpoints of swing s_i. Let a_{k+1} be the left endpoint of the infinite swing of a type 2 algorithm, if the direction of this infinite swing is positive.

Theorem 4.3 *For any FTSS algorithm for a line of unknown size with at most one fault there exist arbitrarily large integers n such that for some starting point inside the line of length n there exists an adversary that forces the head to perform at least $4n-3$ steps.*

Proof: Fix a FTSS algorithm $\mathcal{A}$. Take an arbitrary threshold n_0. We have to show an $n > n_0$ such that $\mathcal{A}$ performs at least $4n-3$ steps for some adversary, on the line of length n. First suppose that $\mathcal{A}$ is of type 1 and let $(s_1, t_1, s_2, t_2, \ldots)$ be the infinite sequence coding its first part. Let a_i and b_i, for $i \geq 1$, be the left and right endpoints of swing s_i. The set of integers $\{a_i, b_i : i > 1\}$ cannot be contained in a finite interval, for otherwise the algorithm would be incorrect. Hence either the sequence $(a_1, a_2, \ldots)$ does not contain the smallest number or the sequence $(b_1, b_2, \ldots)$ does not contain the largest number. Consider three cases.

Case 1. The sequence $(a_1, a_2, \ldots)$ does not contain the smallest number and the sequence $(b_1, b_2, \ldots)$ does not contain the largest number.

We define the following infinite sequences $(a'_1, a'_2, \ldots)$ and $(b'_1, b'_2, \ldots)$ by induction. Let $a'_1 = a_1 = 0$. Let j be the smallest index such that $a_j < a'_1$. Define b'_1 to be the largest integer among $b_1, \ldots, b_{j-1}$. Suppose that $a'_1, \ldots, a'_{k-1}$ and $b'_1, \ldots, b'_{k-1}$ are already defined and let r and s be the smallest indices such that $a'_{k-1} = a_r$ and $b'_{k-1} = b_s$. Suppose that $s \geq r$. Let t be the smallest index such that $b_t > b_s$. Define a'_k to be the smallest integer among $a_{s+1}, a_{s+2}, \ldots, a_t$. Let z be the smallest index among $s+1, s+2, \ldots, t$ such that $a'_k = a_z$. Let m be the smallest index such that $a_m < a'_k$. Define b'_k to be the largest integer among $b_z, b_{z+1}, \ldots, b_{m-1}$. Let x be the smallest index among $z, z+1, \ldots, m-1$, such that $b'_k = b_x$. We have $x \geq z$, hence the inductive construction is completed (cf. Fig. 3).

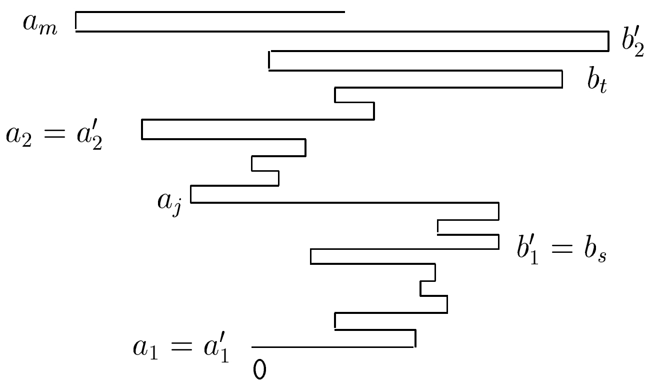

Figure 3: The construction of the sequences (a'_i), (b'_i) in the proof of Theorem 4.3, (case 1).

By construction, the sequences $(a'_1, a'_2, \ldots)$ and $(b'_1, b'_2, \ldots)$ have the following properties.

- the sequence $(a'_1, a'_2, \ldots)$ is strictly decreasing;
- the sequence $(b'_1, b'_2, \ldots)$ is strictly increasing;
- in the fault-free execution of the algorithm in the line segment $I = [a'_v - 1, b'_v + 1]$, the head does not hit a wall between reaching point a'_v and point b'_v for the first time.

Let $v > 2$ be such that $b'_v - a'_v > n_0$. Consider the line segment $I = [a'_v - 1, b'_v + 1]$. Let $n = b'_v - a'_v + 2$ be its length. Before reaching point a'_v for the first time, the head does not hit a wall and performs at least 3 steps. After reaching point a'_v it reaches point b'_v without hitting a wall and subsequently hits the wall for the first time. By construction, this must be the left wall. Hence by the time of first hitting a wall the head performs at least $3 + (n-2) + (n-1) = 2n$ steps (and the line is not yet fully explored). Hence the head must still perform a walk to a distance n from the left wall without hitting a wall during this walk. Now the adversary places the fault in the first point of this walk when the head is at distance $n/2$ (resp. $(n-1)/2$) from the left wall, if n is even (resp. odd). Hence, at least $n-1$ steps after hitting the left wall for the first time, the head hits the left wall again. Since the interval I is still not fully explored, n more steps are needed, for a total of at least $2n + (n-1) + n = 4n - 1 > 4n - 3$ steps, in this case.

Case 2. The sequence $(a_1, a_2, \ldots)$ contains the smallest number and the sequence $(b_1, b_2, \ldots)$ does not contain the largest number.

Subcase 2.1. There exist arbitrarily large integers m such that for all j larger than some index $i(m)$ we have $a_j > m$.

Take such an integer $m >$max(n_0, a_2) and let $i(m)$ be the smallest index such that for all $j > i(m)$ we have $a_j > m$. Let a be the smallest number among $a_1, \ldots a_{i(m)}$ and let b be the largest number among $b_1, \ldots b_{i(m)-1}$. Consider the line segment $I = [a-1, b+1]$ or $I = [a-2, b+1]$, whichever is of even length. Let n be equal to this length. Hence n is an even integer larger than n_0. The rest of the argument is carried out for $I = [a-1, b+1]$. The other case is similar. Denote $x = m - a + 1$ and $y = b + 1 - m$. Hence $n = x + y$.

The swing $s_{i(m)}$ contains the point m. The adversary places the fault during swing $s_{i(m)}$ in this point. Since the sequence $(b_1, b_2, \ldots)$ does not contain the largest number and in view of the placement of the fault, the first time the head hits a wall, it will be the left wall. Since the head started at 0 and before the encounter of the fault it has changed direction at points a and b, the number of steps until hitting the wall is at least $2n - 3$.

Take $c < a$ so small that $x + z \geq 2(n-1)$, where $z = m - c$. At the moment of hitting the left wall in the line segment I in the way described above, the situation from the point of view of the head is identical as if there were no fault but the segment were $J = [c, 2m - a + 1]$ and the wall hit were the right one (at point $2m - a + 1$). In this second scenario a part of the segment J has not been visited yet, and the adversary is left with a fault. Hence in this scenario the head must still hit the opposite wall and hence make a walk at distance at least n without hitting a wall in the meantime. Now the adversary places the fault when the head first gets at distance $n/2$ from the right wall during this walk (recall that n is even). This results in hitting the right wall again after n steps. The same is true in the first scenario where the head will hit the right wall (and in this scenario the segment I is fully explored). However now the situation is again identical in both scenarios and in the second scenario the segment J is not yet fully explored. Hence another walk at distance at least n is needed for the algorithm to be correct in this case. Since the head is in the same situation in both scenarios, it must walk again at distance n from the wall in the first scenario as well (thus performing at least n more steps). This implies that (in the first scenario) it must perform a total of at least $(2n - 3) + 2n = 4n - 3$ steps.

Subcase 2.2. There exists an integer m_0 such that for all $m \geq m_0$ we have $a_i \leq m$ for infinitely many indices i.

Take $m >$max(m_0, n_0). Let c be the smallest number in the sequence $(a_1, a_2, \ldots)$. Let i be such that:
1. $a_i \leq m$;
2. the sum of lengths of swings s_j and t_j for $j < i$ exceeds $2(m - c + 2)$;
3. $b_j > m$ for some $j < i$.

Let b be the largest integer among b_j for $j < i$. Let $k > i$ be such that $a_k \leq m$ and $b_j > b$ for some $j < k$. Let d be the largest integer among $b_1, b_2, \ldots b_{k-1}$. Consider the line segment $I = [c-1, d+1]$ and let n be its length. Since $d > m$, we have $n > n_0$. The sum of lengths of swings s_j and t_j, for $i \leq j < k$, is at least $2(d - m)$. The sum of lengths of swings s_j and t_j, for $j < i$, exceeds $2(m - c + 2)$. Hence the number of steps performed till the end of swing t_{k-1} is at least $2n$ and this is before the first hit of a wall. Hence the number of steps until hitting a wall for the first time is also at least $2n$.

Now an argument analogous to that in Case 1 shows that the head must perform at least $(n-1)+n$ steps after hitting a wall for the first time, for a total of at least $4n-1 > 4n-3$ steps, in this case.

Case 3. The sequence $(b_1, b_2, \ldots)$ contains the smallest number and the sequence $(a_1, a_2, \ldots)$ does not contain the largest number.

The argument is similar as in Case 2, hence we omit it.

This concludes the proof for type 1 algorithms. Now suppose that algorithm $\mathcal{A}$ is of type 2. We present the proof in the case when the infinite swing is in the positive direction. The other case is similar. Fix a positive integer n_0. Let $(s_1, t_1, s_2, t_2, \ldots, s_k, t_k)$ be the sequence coding the first part of algorithm $\mathcal{A}$. Let a_i and b_i, for $i \geq 1$, be the left and right endpoint of swing s_i. Let a be the smallest among integers $a_1, a_2, \ldots, a_k, a_{k+1}$ and let b' be the largest among integers $n_0, b_1, b_2, \ldots, b_k$. If $b' - a$ is even, let $b = b'$, otherwise let $b = b' + 1$. Consider the line segment $I = [a-1, b+1]$. Let n be equal to this length. Hence n is an even integer larger than n_0. The adversary places the first fault during the infinite swing in point b. The first time the head hits the wall, it will be the left wall, after at least $2n-3$ steps. It remains to show that $2n$ more steps are required. The proof is similar as in Subcase 2.1 for type 1 algorithms, hence we omit it. $\square$

We now turn attention to the case when the head starts at a wall. Suppose, without loss of generality, that this is the left wall. Fix any FTSS algorithm $\mathcal{A}$ and consider the part of its execution until a wall is hit for the first time. This part can be again coded in one of the two ways described previously. We keep the same notation and terminology and define the two types of algorithms similarly as before. In particular, the left wall at which the head starts is the point 0. Now the infinite swing of a type 2 algorithm must be in the positive direction. For the case of start at a wall we have the following lower bound which again matches the performance of Algorithm UNKNOWN for $k = 1$.

Theorem 4.4 *For any FTSS algorithm for a line of unknown size with at most one fault there exist arbitrarily large integers n such that if the head starts at a wall of a line of length n then there exists an adversary that forces the head to perform at least $3n-2$ steps.*

Proof: Fix a FTSS algorithm $\mathcal{A}$. Take an arbitrary threshold n_0. We have to show an $n > n_0$ such that $\mathcal{A}$ performs at least $3n-2$ steps for some adversary, on the line of length n. First suppose that $\mathcal{A}$ is of type 1 and let $(s_1, t_1, s_2, t_2, \ldots)$ be the infinite sequence coding its first part. Let a_i and b_i, for $i \geq 1$, be the left and right endpoints of swing s_i. The sequence of integers $\{b_i : i > 1\}$ must be unbounded, for otherwise the algorithm would be incorrect. Consider two cases.

Case 1. There exist arbitrarily large integers m such that for all j larger than some index $i(m)$ we have $a_j > m$.

Take such an integer $m > n_0$ and let $i(m)$ be the smallest index such that for all $j > i(m)$ we have $a_j > m$. Let b be the largest number among $b_1, \ldots b_{i(m)-1}$. Consider the line segment $I = [0, b+1]$. Let $n = b+1$ be its length. The swing $s_{i(m)}$ contains the point m. The adversary places the fault during swing $s_{i(m)}$ in this point. Since the sequence $(b_1, b_2, \ldots)$ does not contain the largest number and in view of the placement of the fault, the first time the head hits a wall, it will be the left wall. The number of steps until hitting the wall is at least $2n-2$. n more steps are necessary to explore the entire line, for a total of $3n-2$ steps.

Case 2. There exists an integer m_0 such that for all $m \geq m_0$ we have $a_i \leq m$ for infinitely many indices i.

Take $m >$max(m_0, n_0). Let i be such that:
1. $a_i \leq m$;
2. the sum of lengths of swings s_j and t_j for $j < i$ exceeds $3m+2$;
3. $b_j > m$ for some $j < i$.

Let b be the largest integer among b_j, for $j < i$. Let $k > i$ be such that $a_k \leq m$ and $b_j > b$, for some $j < k$. Let d be the largest integer among $b_1, b_2, \ldots b_{k-1}$. Consider the line segment $I = [0, d+1]$ and let $n = d+1$ be its length. Since $d > m$, we have $n > n_0$. The sum of lengths of swings s_j and t_j, for $i \leq j < k$, is at least $2(d-m)$. The sum of lengths of swings s_j and t_j, for $j < i$, exceeds $3m+2$. Hence the number of steps performed till the end of swing t_{k-1} is at least $2n+m$ and this is before the first hit of a wall. The head is now at distance at least $n-m$ from the right wall and this wall has not been hit yet. Hence the total number of steps needed to explore the entire line is at least $3n$ is this case.

This concludes the proof for type 1 algorithms. Now suppose that algorithm $\mathcal{A}$ is of type 2. Fix a positive integer n_0. Let $(s_1, t_1, s_2, t_2, \ldots, s_k, t_k)$ be the sequence coding the first part of algorithm $\mathcal{A}$. Let a_i and b_i, for $i \geq 1$, be the left and right endpoint of swing s_i. Let b be the largest among integers $n_0, b_1, b_2, \ldots, b_k$. Consider the line segment $I = [0, b+1]$. Let $n = b+1$ be its length. The adversary places the first fault during the infinite swing in point b. The first time the head hits a wall, it will be the left wall, after at least $2n-2$ steps. n more steps are necessary to explore the entire line, for a total of $3n-2$ steps.

□

4.3 An alternative algorithm

Theorems 4.3 and 4.4 show that Algorithm UNKNOWN cannot be improved for $k = 1$ fault and all (unknown) sizes n of the line. It is natural to ask if the lower bounds from Section 4.2 generalize to an arbitrary number of faults. In other words, is Algorithm

Unknown (asymptotically) optimal for arbitrary k and n? We now show that this is not the case. For large k and n, the cost of Algorithm Unknown is asymptotically $2kn$. More precisely, it is $2kn + o(kn)$, when both k and n are unbounded. The upper bound on this complexity was shown in Theorem 4.2, and the lower bound is easily shown by an adversary that puts a fault one step before the wall in each execution of the repeat loop.

In what follows we present an algorithm working for arbitrary k and arbitrary unknown n, which for infinitely many n has cost $kn + o(kn)$. This is approximately half of the cost of Algorithm Unknown and it is asymptotically optimal, in view of our lower bounds from Theorems 3.5 and 3.6, which hold even for known n.

The idea of the algorithm is the following. First we choose an infinite sequence of numbers $(n_i : i = 1, 2, \ldots)$, for which the algorithm will work efficiently. Many such sequences are possible: it is enough if their terms are odd and grow sufficiently fast. To fix attention and simplify analysis we define them as follows: $n_1 = 3$ and $n_{i+1} = 2^{(3k+2)n_i} + 1$. The algorithm first "guesses" that the length of the line is n_1 and executes procedures probe(n_1). If the guess was correct it detects this fact and stops. Otherwise it executes procedure terminate(n_1). The aim of this procedure is to stop the algorithm after the first guess which exceeds the actual length of the line. If the algorithm did not stop after terminate(n_1), it guesses that the length of the line is n_2 and executes procedure probe(n_2) and possibly procedure terminate(n_2). This continues until the first guess larger or equal than the actual length of the line. Then the algorithm stops. For any length for which a guess was correct, i.e., for any length n_i, the algorithm detects the correctness of the guess and stops after executing procedure probe(n_i), before calling terminate(n_i). At this point the line is explored, if it is indeed of size n_i. We will prove that the number of steps for these lengths of the line is $kn_i + o(kn_i)$. We will also prove that the algorithm is correct for all other lengths, although then it is not as efficient. Nevertheless, for all other lengths n its cost is still $O(kn)$.

The precise description of the two procedures is the following.

```
procedure PROBE($n$)

$was-at-wall := 0$; $halt := 0$
**if** $inside$ **then** $count := k+3$
**else** ($count := k+2$, $was-at-wall := 1$)
**repeat** $count$ times
    **if** $inside$ **then**
        $go\text{-}straight$ **until** ($nohit(n-1)$ OR $hit$)
    **else** /* at-wall */
        $was-at-wall := 1$
        $go\text{-}straight$ **until** ($nohit(n)$ OR $hit$)
    **if** $was-at-wall = 1$ and $at\text{-}wall$ **then**
        $x :=$ the number of steps in last round
        **if** (last round started inside and $x$ even) **then** $halt := 1$
        **if** (last round started at wall and $x$ odd) **then** $halt := 1$
    reverse direction
**end**
```

```
procedure TERMINATE($n$)

**repeat** $3k+2$ times
        $go\text{-}straight$ **until** ($nohit(n)$ OR $hit$)
        reverse direction
**if** there were at least $k+1$ rounds starting and ending at a wall
**then**
        $halt := 1$
**end**
```

Now our algorithm can be succinctly formulated as follows.

```
Algorithm GUESS-AND-PROBE

$i := 0$; $halt := 0$
**while** $halt = 0$ **do**
        $i := i+1$
        PROBE($n_i$)
        **if** $halt = 0$ **then** TERMINATE($n_i$)
**end**
```

Before proceeding to the analysis of our algorithm we explain the meaning of the variables used in our procedures. $was-at-wall$ is a flag that is set to 1 at the first time when the head is at a wall and it is never changed subsequently. $halt$ can be set to 1 in both procedures and its role is to stop the algorithm as soon as it is certain that the entire line has been explored. It is set to 1 in procedure PROBE when the head was previously at a wall, then it hits the wall again and the last round either started inside and had an even number of steps or started at a wall and had an odd number of steps. *halt* can be also set to 1 in procedure TERMINATE when there were at least $k+1$ rounds starting and ending at a wall.

We first show that the algorithm never stops prematurely, regardless of the length of the line.

Lemma 4.1 *For any length of the line, when Algorithm* GUESS-AND-PROBE *stops then the entire line is explored.*

Proof: The algorithm stops after the first call of procedure PROBE(n_i) or procedure TERMINATE(n_i) in which the variable *halt* is set to 1. Consider two cases.

Case 1. *halt* is first set to 1 in procedure PROBE(n_i).

This happens when the head was previously at a wall then it hits the wall again and the last round either started inside and had an even number of steps or started at a wall and had an odd number of steps. We first show that at this point the entire line is explored. Consider the execution of the procedure since the previous hit of a wall. Without loss of generality assume that it was wall L. Now the head is again at a wall.

First assume that the last round started inside and had an even number of steps. The sequence of rounds between the start from wall L and the present hit was the following: a sequence of rounds ending inside the line followed by the last round hitting a wall. The cumulative number of steps in the sequence of rounds ending inside the line is odd: the first round has n_i steps, the following ones have $n_i - 1$ steps, and n_i is odd. Hence the distance of the head from wall L after each of these rounds is odd as well. Since the last round has an even number of steps, the distance of the head from L after this round is also odd, hence it cannot be 0. It follows that now the head cannot be at wall L. Hence it is at wall R and the exploration is completed.

Next assume that the last round started at a wall and had an odd number of steps. At the end of this round the head must be at an odd distance from the wall where it started. Since it is now at a wall, this cannot be the wall at which it started. Hence it must be the other wall and the line is explored.

Case 2. *halt* is first set to 1 in procedure TERMINATE(n_i).

This means that there were at least $k+1$ rounds starting and ending at a wall. At most k of them could contain a fault, hence at least one of them is correct. During such a round, the head must go from one wall to the other and hence must explore the entire line. □

We now analyze the algorithm in the case when the length of the line is n_i, i.e., when one of the guesses is correct.

Lemma 4.2 *If the length of the line is n_i, for some $i > 0$, then Algorithm* GUESS-AND-PROBE *stops after executing procedure* PROBE*(n_i) and the line is explored.*

Proof: Lemma 4.1 implies that when Algorithm GUESS-AND-PROBE stops, the line is completely explored. It remains to prove that this will happen after executing procedure PROBE(n_i). Define a *phase* to be a sequence of rounds between two consecutive hits of a wall. Hence a phase is composed of a sequence of rounds ending inside the line and a last round that hits a wall.

Claim. In every phase of r rounds in which *halt* is not set to 1 there are at least r faults.

First notice that if $r = 1$ then there is one round in the phase which starts and ends at a wall. If this round has less than r faults, i.e., if it is correct then it has exactly n_i steps and *halt* is set to 1, because n_i is odd. Hence the claim holds for $r = 1$. Assume that $r > 1$. In order to prove the claim observe that every round that terminates inside the line must contain at least one fault. Hence if the claim is false then each of the first $r-1$ rounds of the phase must contain exactly 1 fault and the last round must be correct. Suppose (without loss of generality) that the phase starts at wall L. Then the direction at the beginning of each round must be from L to R. However (as observed in the proof of Lemma 4.1) at the beginning of the last round of the phase the head is at an odd distance from L, hence at an even distance from R. Therefore the number of steps in the last phase is even and hence *halt* is set to 1, contrary to the assumption. This contradiction proves the claim.

Now consider two cases.

Case 1. The head starts at a wall.

If *halt* is not set to 1 after the first k rounds then the adversary must have used all the faults, in view of the proof of the claim. Hence the $(k+1)$th round must be correct and hence the head must hit a wall. Now the $(k+2)$th round must be also correct and hence the head will hit the other wall after exactly n_i steps, causing the variable *halt* to be set to 1 and the algorithm to stop.

Case 2. The head starts inside the line.

Suppose that there are t rounds before hitting a wall for the first time. Each of the first $t-1$ of them terminates inside the line and hence must contain at least one fault. Suppose that *halt* is not set to 1 after the first $k+1$ rounds. Then the adversary must have used all the faults, in view of the claim. Similarly as in Case 1, the $(k+2)$th round must be correct and hence the head must hit a wall. Now the $(k+3)$th round must be also correct and hence the head will hit the other wall after exactly n_i steps, causing the variable *halt* to be set to 1 and the algorithm to stop.

It follows that if the line has length n_i then Algorithm GUESS-AND-PROBE always stops after executing procedure PROBE(n_i) and that the line is then explored. □

Our next lemma establishes the complexity of Algorithm GUESS-AND-PROBE for lines of any length n_i.

Lemma 4.3 *If the length of the line is n_i, for some $i > 0$, then Algorithm* GUESS-AND-PROBE *uses $kn_i + o(kn_i)$ steps.*

Proof: If the length of the line is n_i, the algorithm executes procedure PROBE(n_j) for $j \leq i$ and procedure TERMINATE(n_j) for $j < i$. Procedure PROBE(n_j) has at most $k+3$ rounds of length at most n_j, hence it uses at most $(k+3)n_j$ steps. Procedure TERMINATE(n_j) has $3k+2$ rounds of length at most n_j, hence it uses at most $(3k+2)n_j$ steps. Since $n_i = 2^{(3k+2)n_{i-1}} + 1$, all calls for $j < i$ use a total of $O(\log(kn_i))$ steps. It follows that the entire algorithm uses at most $(k+3)n_i + O(\log(kn_i)) = kn_i + o(kn_i)$ steps. □

It remains to show that Algorithm GUESS-AND-PROBE is always correct, although possibly less efficient than for lengths n_i. In particular we have to show that the algorithm always stops.

Lemma 4.4 *Algorithm* GUESS-AND-PROBE *correctly explores a line of any length n and uses $O(kn)$ steps.*

Proof: Fix any length n of the line. Let m be the smallest n_i such that $m \geq n$ and let $j = i-1$. We first show that the algorithm stops (at the latest) after executing procedure TERMINATE(m). Since $m \geq n$, every round of procedure TERMINATE(m) that ends inside the line must contain at least one fault. Hence there are at most $2k+1$ rounds in procedure TERMINATE(m) that do not start and end at a wall. It follows that there are at least $k+1$ rounds that start and end at a wall, and consequently Algorithm GUESS-AND-PROBE stops after executing procedure TERMINATE(m), unless it stopped before.

We now estimate the number of steps used until the end of procedure TERMINATE(m). All calls of procedures PROBE(n_t) and TERMINATE(n_t), for $t < j$, take $O(\log(kn))$ steps. All rounds in procedures PROBE(n_j) and TERMINATE(n_j) are of length at most $n_j < n$ and there are $O(k)$ of them, hence procedures PROBE(n_j) and TERMINATE(n_j) use $O(kn)$ steps. It remains to consider procedures PROBE(m) and TERMINATE(m). Each correct round in these procedures uses at most n steps and each fault can increase a round by at most n steps. Since there are $O(k)$ rounds in both these procedures, it follows that the total number od steps in both of them is $O(kn)$. Hence the entire cost of Algorithm GUESS-AND-PROBE is $O(kn)$. □

We have proved the following result.

Theorem 4.5 *Algorithm* GUESS-AND-PROBE *correctly explores a line of any length n, with at most k faults. For every n it uses $O(kn)$ steps and for infinitely many n it uses $kn + o(kn)$ steps, which is asymptotically optimal.*

5 Conclusion

We considered fault-tolerant aspects of the fundamental problem of sequential scan, where a line of identical objects has to be explored in spite of adversarial faults affecting moves of the exploring mobile entity. We established optimal cost of fault-tolerant sequential scan for a line of known size and partially solved the problem for unknown size. It remains open if there exists a sequential scan algorithm for a line of unknown size n and at most k faults, which has cost $kn + o(kn)$, for all k and n. Our conjecture is no, i.e., we think that the leading factor 2 in Theorem 4.2 cannot be removed.

Viewed from the point of view of applications to network exploration, our study opens the area of fault-tolerant exploration by a mobile entity in which faults concern moves of the entity, rather than the environment. In particular, it would be interesting to investigate optimal fault-tolerant graph exploration algorithms for labeled graphs. Either nodes or ports of the underlying graph can be labeled and the mobile entity (agent) can perceive these labels. This capability would add a lot of power to exploration algorithms, as the agent could memorize its"trace" and compare it to the currently read label, thus potentially becoming aware of a fault earlier than in an anonymous scenario. Even for the line, the ability to perceive and memorize labels would probably yield significant changes in performance, compared to our present model.

Acknowledgments. Thanks are due to the anonymous referees whose detailed remarks enabled us to remove some errors and improve the readability of the paper.

References

[1] S. Albers and M. R. Henzinger, Exploring unknown environments, SIAM Journal on Computing 29 (2000), 1164-1188.

[2] M.A. Bender, A. Fernandez, D. Ron, A. Sahai and S. Vadhan, The power of a pebble: Exploring and mapping directed graphs, Proc. 30th Ann. Symp. on Theory of Computing (STOC 1998), 269-278.

[3] J. Czyzowicz, D. Kowalski, E. Markou, A. Pelc, Searching for a black hole in tree networks, Proc. 8th International Conference on Principles of Distributed Systems (OPODIS 2004), 35-45.

[4] A. Dessmark and A. Pelc, Optimal graph exploration without good maps, Theoretical Computer Science 326 (2004), 343-362.

[5] S. Dobrev, P. Flocchini, G. Prencipe, N. Santoro, Mobile agents searching for a black hole in an anonymous ring, Proc. of 15th International Symposium on Distributed Computing, (DISC 2001), 166-179.

[6] S. Dobrev, P. Flocchini, G. Prencipe, N. Santoro, Searching for a black hole in arbitrary networks: Optimal Mobile Agents Protocols, Proc. 21st ACM Symposium on Principles of Distributed Computing (PODC 2002), 153-161.

[7] C.A. Duncan, S.G. Kobourov and V.S.A. Kumar, Optimal constrained graph exploration, Proc. 12th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA 2001), 807-814.

[8] F.E. Fich, P. Ragde, A. Wigderson, Relations between concurrent-write models of parallel computation, SIAM Journal on Computing 17 (1988), 606 - 627.

[9] P.C. Kanellakis, and A.A. Shvartsman, Efficient parallel algorithms can be made robust, Distributed Computing, 5 (1992) 201 - 217.

[10] E. Markou, A. Pelc, Efficient exploration of faulty trees, Proc. 15th Australasian Workshop on Combinatorial Algorithms (AWOCA'2004), 52-63. Also: Theory of Computing Systems, to appear.

[11] G.L. Miller, and J.H. Reif, Parallel tree contraction and its application, Proc. 26th symp. on Foundations of Computer Science (FOCS 1985), 478-489.

[12] P. Panaite and A. Pelc, Exploring unknown undirected graphs, Journal of Algorithms 33 (1999), 281-295.

[13] A. Pelc, Fault-tolerant broadcasting and gossiping in communication networks, Networks 28 (1996), 143-156.

[14] F. Preparata, G. Metze and R. Chien, On the connection assignment problem of diagnosable systems, IEEE Transactions on Electron. Computers 16 (1967), 848-854.

[15] T. Sander, C.F. Tschudin, Protecting mobile agents against malicious hosts, Proc. Conf. on Mobile Agent Security (1998), LNCS 1419, 44-60.

[16] K. Schelderup, J. Ines, Mobile agent security – issues and directions, Proc. 6th Int. Conf. on Intelligence and Services in Networks, LNCS 1597 (1999), 155-167.

[17] J. Vitek, G. Castagna, Mobile computations and hostile hosts, in: Mobile Objects, D. Tsichritzis, Ed., University of Geneva, 1999, 241-261.