

Adversary Immune Leader Election in Ad Hoc Radio Networks^{*}

Mirosław Kutylowski^{1,2} and Wojciech Rutkowski¹
mirekk@im.pwr.wroc.pl, rutkow@im.pwr.wroc.pl

¹ Institute of Mathematics, Wrocław University of Technology
² CC Signet

Abstract. Recently, leader election algorithms have been studied for ad hoc radio networks. Efficient solutions have been obtained with respect to time complexity and energy cost even for the no-collision detection (no-CD) model. However these algorithms are fragile in the sense that an adversary may disturb communication at low energy cost so that the outcome becomes faulty. As a consequence, more than one station may regard itself as the leader. This is a severe fault, since leader election is intended to be a fundamental subroutine used to avoid collisions. Moreover, ad hoc radio networks are often regarded in the context of building secure distributed networks.

It is impossible to make leader election algorithm totally resistant – if the adversary is broadcasting all the time it causes collisions and no message comes through. So we consider the case that an adversary has limited energy resources, as any other participant.

We present an approach that yields a randomized leader election algorithm for a single-hop no-CD radio network. The algorithm has time complexity $O(\log^3 N)$ and energy cost $O(\sqrt{\log N})$. This is worse than the best algorithms constructed so far ($O(\log N)$ time and $O(\log^* N)$ energy cost), but succeeds in presence of an adversary with energy cost $\Theta(\log N)$ with probability $2^{-\Omega(\sqrt{\log N})}$. (The $O(\log^* N)$ energy cost algorithm can be attacked by an adversary with energy cost $O(1)$.)

1 Introduction

Radio Network Model The networks considered in this paper consist of processing units, called *stations*, which communicate through a shared communication channel. Since neither the number of stations nor their ID's are known, they are described as *ad hoc networks*. Since a shared communication channel may be implemented by a radio channel, they are called *radio networks*, or *RN* for short.

The networks of this kind are investigated quite intensively due to several potential applications, such as sensor networks [5]. If deploying a wired network is costly (e.g. a sensor network for collecting and analyzing certain environment data) or impossible (e.g. an ad hoc network of stations deployed in cars for optimization of road traffic), such networks might be the only plausible solution.

In this paper we consider *single-hop* RN's: if a station is sending a message any other station may receive it (so we concern networks working on a small area). If two

^{*} partially supported by KBN grant 7 T11C 032 20

stations are sending simultaneously, then a *collision* occurs. We make here a (standard) pessimistic assumption that in the case of a collision the messages are scrambled beyond recognition. Moreover, we assume that the result is indistinguishable from a random noise – the model is called *no-CD RN*. In accordance with industrial standards, we also assume that a station cannot simultaneously transmit and listen.

Unlike in other distributed systems, RN's work synchronously and computation is divided into *time slots*. (It has been pointed that it is a realistic assumption - at least if GPS signals can be received by the stations.) In a time slot a station can be either transmitting or listening or may perform internal work only. In the first two cases we say that a station is *awake*.

Usually, the stations of a RN are battery operated. So it is crucial to design energy efficient algorithms - it helps to eliminate failures due to battery exhaustion. Since energy consumption of a processor is lower by an order of magnitude than the energy consumption for transmitting and listening, we consider only energy consumption due to communication. Unexpectedly, in the case of small networks the energy for transmitting and listening are comparable [5], so we can disregard these differences until the final fine tuning of the algorithms. Quality of an RN algorithm is determined usually by two complexity measures: time – the number of time slots necessary for the execution of the algorithm; and energy cost – the maximum over all stations of the number of steps, during which a station is awake.

In the literature different scenarios are considered regarding the stations and their knowledge: either the number of stations is known, or the number of stations is known up to some multiplicative constant, or the stations have no knowledge about the number of other active stations. A similar situation is for the station ID's: either the stations have no ID's at all (and they have to be assigned during so called initialization), or in the range $1..n$, where the number of active stations is $\Omega(n)$.

Further information concerning ad hoc networks can be found in a handbook [10].

Leader Election Problem Most algorithms running on RN's in a multiprocessor environment assign some special roles to certain processing stations. The simplest kind of such an initialization is the following *Leader Election Problem*:

A RN network is given, such that each active station has no knowledge which stations are active (except itself). Initialize the network so that *exactly* one station gets status *leader* and the rest of the active stations receive the status *non-leader*.

Additional assumptions may be considered: the stations may know approximately the number of active stations, the stations may have unique ID's in a given range $1..n$. (Note that if the stations know that all ID's in this range are used, the problem is trivial: the station with ID 1 is the leader. The point is that a station is not aware of the other stations: in particular it must consider the case in which all stations with low ID are non-active or do not exist.)

Security issues Practical applications of RN's must take into account various dangers resulting from the communication model. In a wired network an adversary attacking the system might inject packets with manipulated data concerning their origin, but to

some extent it is possible to trace their origin. In wireless networks it is much harder. As long as no special physical equipment is deployed, the mobile intruder is safe. The algorithms built so far for RN's disregard this issue. This is a severe drawback, since for many applications ad hoc networks must offer a certain security level to be useful. This paper addresses this problem for the case of leader election.

Security problems of previous solutions The simplest leader election algorithm is used by Ethernet [9]: with a certain probability each candidate sends a message (we call it a *trial*). If a station sends a message and there is no collision, then it becomes the leader. The step is repeated until a leader is elected. However, this solution demands that every participant must listen or send all the time, so energy cost equals execution time. Another disadvantage is that even if the expected runtime is $O(1)$, getting the leader with high probability is not that easy [11]. The main problem from our point of view is that the adversary may cause collisions all the time – in this way its energy cost will remain the same as the cost of stations trying to elect the leader. Note that, if an adversary cannot send fake messages so that the other stations think it comes from a legitimate member of the group electing the leader, then it cannot attack this protocol to provide *multiple leaders* (at least two stations which regard themselves as the leaders).

Another algorithm of this kind is presented in [8] – it is assumed that the number of active stations is unknown, the stations may detect collisions, and that the algorithm must be uniform, i.e. all stations perform *trials* using the same probabilities depending on computation history (but not on processor ID). It is easy to disguise this algorithm with extra collisions. Namely, the algorithm uses collisions for adjusting probabilities. Fake collisions may cause overestimating the number of stations electing the leader and consequently executing the *trial* with a wrong probabilities so that nobody sends.

If the stations have unique ID's in the range $1..n$ (with some ID numbers unused), then leader election may be performed deterministically by the following simple *Tree Algorithm* [7]: Consider a binary tree with n leaves labeled 1 through n . In this tree a leaf j represents the station with ID j . The internal nodes of the tree have unique labels $n+1, n+2, \dots$ so that the label of a parent is bigger than the labels of its sons. The leader of the subtree rooted at node $n+i$ is found at step i : The leader of the subtree rooted in the left son of node $n+i$ sends a message and the leader of the subtree rooted in the right son of node $n+i$ is listening. (If there is no active station in a subtree, then there is no leader of the subtree and nobody sends or listens, respectively.) If such a message is actually sent, then the leader of the right subtree becomes non-leader. Otherwise it considers itself as a leader of the subtree with root at node i . The leader of the left subtree becomes the leader of the subtree rooted at node i whenever it exists. This is quite tricky: no message from the leader of the right subtree need to be sent.

Tree Algorithm is not immune: For instance, it suffices to make a collision at some point where there are leaders of the right and left subtrees of a given node. Then the leader of the right subtree would think that the left subtree is empty and will regard itself as the leader of both subtrees. The same is assumed by the leader of the left subtree. Starting from this moment both stations will behave exactly the same and make a collision whenever they send. This leads even to an avalanche so that more and more leaders emerge.

Paper [7] presents an algorithm with Tree Algorithm executed as a final stage. In particular, this part can be attacked.

In paper [6] another strategy is applied for the model, where collisions are detectable. In the first phase *trials* are performed with probabilities $2^{-2^{i^2}}$ for $i = 1, 2, \dots$ until there is no collision, say for $i = T$. The stations that have not transmitted are no more considered as candidates for the leader. During the second phase, *trials* are executed with probabilities 2^{-2^i} for $i = T, T-1, \dots$. Again, in the case of a collision all stations that have not transmitted are eliminated. This tricky algorithm can be efficiently attacked by an adversary. An adversary, who knows the approximate number of stations electing the leader simulates collisions so that $2^{2^{T^2}} > n^2$ - only $O(1)$ additional steps are required. Then, all candidates perform the *trial* with probability $2^{-2^{T^2}} < n^{-2}$. So with high probability no candidate for the leader sends. However, the adversary may send the collision signal. In this case all candidates stop trying to become the leader (in the original algorithm it makes sense, they know that the stations that caused the collision remain.) The attack requires $O(\sqrt{\log \log n})$ energy cost, while the original algorithm has energy cost $O(\log \log n)$. The adversary can achieve that no leader is elected.

In [4] three energy efficient leader election are presented. The first of them - log star deterministic algorithm - working for special inputs executes Tree Algorithm with efficient reassignment of tasks to stations. Anyway, it is at least as vulnerable as Tree Algorithm. Since the third algorithm from [4] uses the first algorithm as a final stage, it is insecure, as well.

The general deterministic leader election algorithm from [4] achieves energy cost $(\log n)^{o(1)}$. It is based on Tree Algorithm, but can be attacked in one more way. The algorithm splits the candidates into groups and elects the leader of each group in two phases: in phase 1 each active participant of the group transmits - if there is no collision, then the only active participant knows that it is a leader. Otherwise, we execute recursively the algorithm within the group: an advantage is that the leader elected for this group gets at least one slave - this slave is used afterwards to reduce the energy cost of its master. An adversary causes a collision during phase 1. So the costly phase 2 is executed, but the leader gets no slave.

New result Our primary goal is to design a leader election algorithm that is energy efficient (preferably sublogarithmic) and tolerates an adversary having limited energy resources. The secondary goal is to preserve polylogarithmic execution time. We get the following result:

Theorem 1. *Consider a single no-CD radio network C consisting of $O(n)$ nodes sharing a secret key. Assume that the stations of the network are not initialized with ID's. It is possible to elect a leader of the network with energy cost $O(\sqrt{\log N})$ within time $O(\log^3 N)$, so that the outcome is faulty with probability $O(2^{-\Omega(\sqrt{\log N})})$ in a presence of an adversary station which has energy cost $O(\log N)$.*

2 Basic Techniques

Cryptographic Assumptions We assume that the stations that are electing the leader have a common secret s entirely hidden from the adversary. How to deploy such a secret is a problem considered in the literature as *key predistribution problem*. In the simplest case the stations that should work in some hostile environment are equipped with these secrets at the time of personalization by inserting secret data stored on smart cards. **MKu uwaga: zmienione**

Secret s can be used for protecting the protocol: messages sent at time step t can be XOR-ed with a string $s_t = f(s, t)$, where f is a secure pseudorandom function. This makes the messages transmitted indistinguishable from a random noise. For this reason the adversary cannot detect when and which messages have been transmitted by the stations electing a leader.

Using such stream encipherment is realistic, since the processor may prepare the pseudorandom strings in advance. This fits also quite well to the fact that the processor of the station is active much longer than the station is sending/receiving. The function f need not to be very strong in cryptographic sense – it is only required that it cannot be broken until a leader is elected or leak s . In order to protect a common secret s we may replace it by a value $s_0 = g(s, t_0)$, where g is a secure hash function and t_0 is, for instance, the starting moment of the election procedure.

Thanks to encryption **WRu uwaga: informal - nie wiem czy zmiescimy sie z tlumaczeniem o co chodzi....**, we assume that the knowledge of the adversary is confined to the knowledge of the algorithm executed, its starting point and the approximate number of stations electing the leader. It cannot derive any information from the communication. So the adversary may only hope to cause collisions at right moments and in this way to break down the protocol.

Initial selection It is well known that *trials* are well suited to select a small group of candidates **WRu uwaga: suited - znowu nie ma miejsca zeby cos wyjasniac.... wystarczy odwołanie do (see [4])** for the leader in a way resistant to adversary attacks. Let us recall it in detail:

By a *participant* we shall mean any station which performs the leader election protocol. Let us assume that there are $n = \Theta(N)$ participants and N is known to all stations (including the adversary). First a participant tosses a coin and with probability 0.5 it becomes a *sender*, otherwise it becomes a *receiver*. Then it participates in d rounds (d is a parameter): during a round a participant decides to be active with probability N^{-1} . The round consists of 3 communication steps: during step 1 an active sender broadcasts a random message, simultaneously each active receiver listens, at step 2 the receiver repeats the message received while the sender listens, if it gets back its own message, then it repeats it, while the receiver is listening. If the sender gets its own message, it knows that there was no collision and exactly one participant has responded. At step 3 the receiver may check whether there was no collision. In this case we say that a pair of participants, the sender and receiver, succeeds, and we assign to them the round number as a temporary ID.

In order to keep energy cost $O(1)$ we assume also that a participant may decide to be active for at most one round. Despite that the process is not a sequence of Bernoulli trials, one may prove that the number of successes does not change substantially [1].

Note that initial selection is resistant against an adversary: since we assume that its energy capacity is $O(\log N)$, for a sufficiently large $r = \Theta(\log N)$, the adversary can only reduce probability of success in a trial if r trials are performed.

Time windows Suppose that the stations with a common secret s have to exchange one message and a time window consisting of r consecutive steps is available for this purpose. Then, based on the secret s they may determine the moment within the window in which they actually communicate (by transmitting ciphertexts). Then an adversary can **WRu uwaga: bylo cannot** neither detect a transmission nor guess the moment of transmission. So he can send messages blindly hoping to get a collision. So, if the adversary sends m messages within the window, the probability of a collision at transmission moment equals $\frac{m}{r}$.

There is a trade off between security protection and time complexity of the protocol: using windows of size r changes time complexity by a factor of r .

Random reassignment of ID's After initial selection there are candidates which will elect a leader among themselves. An adversary can disturb choosing candidates for only a small fraction of ID numbers. The problem is that it can focus on some ID number (for instance consecutive ones), if it could bring some advantage for him. There is a simple countermeasure: each temporary ID u is replaced by $\pi_{s,t_0}(u)$, where π is a pseudorandom permutation $\pi_s = h(s, t_0)$, where h is an appropriate cryptographic generator of pseudorandom permutations based on seed s and t_0 .

3 Adversary Resistant Leader Election

The algorithm consists of a preprocessing and $v = \Theta(\sqrt{\log N})$ *group election* phases. Preprocessing chooses a relatively small group of candidates for the leader. Each group election phase has the goal to elect a leader from a different group of candidates chosen by preprocessing assigned to this phase. There is no election of the leader from the group leaders – the first group that succeeds in choosing a group leader “attacks” all subsequent group election phases so that it prevents electing a leader from any subsequent group.

Preprocessing Initial selection from Section 2 is executed for $d = v \cdot k$, where $k = O(\log N)$. If a round j of initial selection succeeds, then the stations chosen are denoted by P_j and P'_j , and called, *stations with ID j* . The station P_j is a *candidate* for the leader in our algorithm, P'_j is required for auxiliary purposes.

In this way we establish $\Omega(d)$ pairs of stations with ID's in the range $1..d$. The pairs with ID's $(i-1) \cdot k, \dots, i \cdot k$, are assigned for group election phase i . Before assigning the ID's permutation trick **WRu uwaga: trick - informal sugestia z komentarzy - technique** (Section 2) is executed. So an adversary attacking during the initial selection cannot determine the ID's that are eliminated by the collisions.

A group election phase quad Let us consider the ID's assigned to this phase. Since every round of initial selection may fail, we talk about *used* and *unused* ID's - in the later case there is no candidate for the group leader with this ID.

Let us arrange all ID's assigned to this phase in a line. Some of these ID's are used, however they may be separated by blocks of unused ID's. At the beginning of a phase each candidate of the group knows that its own ID is used but has no idea about status of the other ID's. The goal of the phase is to chain all used ID's of this group. First each used ID has to find the next used ID according to the natural ordering. Then each candidate will get knowledge about the whole chain. Then it will be straightforward to determine a group leader by applying a certain deterministic function to the chain - this can be done locally by each candidate.

For the sake of simplicity of exposition we assume that the ID's assigned to the phase considered are 1 through k . Building a chain without energy cost limitations is easy: at step j a candidate P_a such that $a < j$ and there is no candidate P_l with ID $l \in (a, j)$, sends a message and P_j listens. If P_j responds at the next step, then P_a knows its successor in the chain and stops its activities for building the chain. If there is no response, then it assumes that ID j is unused and proceeds to step $j + 1$ in which it tries to contact P_{j+1} . However, this approach has certain drawbacks. First, it may happen that there is a long block of unused ID's. In this case the last candidate before the block would use a lot energy to find a subsequent used ID. The second problem is that an adversary may cause collisions - so it might happen that the knowledge of different candidates becomes inconsistent. It is easy to see that in this case the result of the algorithm would be faulty.

Building the chain consists of two sub-procedures. During the first one we build disconnected chains. The second sub-procedure has the goal to merge them. The next two sections present the details.

In the last subsection we describe how to modify these sub-procedures so that a group that succeeds in choosing a group leader can prevent electing a leader in all subsequent groups (note that the changes must be done carefully, these capabilities cannot be granted for an adversary!).

Building chains The procedure of building chains uses k communication slots, each one consisting of four windows (Section 2) of size $\Theta(\log^{3/2} N)$. Each chain corresponds to an interval of ID's inspected by the stations related to the chain. At each moment a chain has an *agent*, which is the station P_a , where a is the last used ID in the interval mentioned. The last ID in the interval (it need not to be a used ID) is the *end of the chain*. During execution of the protocol the chains grow, and at communication slot j , we try to attach ID j into a chain ending at position $j - 1$. There are two potential parties active in this trial: stations P_j, P'_j and stations P_a, P'_a , where P_a is the agent of a chain ending at position $j - 1$. The last chain is called *the current chain*. (For $j = 1$ there is no chain below so there are no agents, but the stations may execute the same code.) For the purpose of communication, information about the current chain can be encoded by a string of length k : it contains a 1 at position j , if j is a used ID, and a 0, if j is an unused ID, or symbol $-$, if position j does not belong to the chain so far. During communication slot j four steps are executed, during which P_j, P'_j, P_a, P'_a listening whenever not transmitting.

step 1: The agent P_a of the current chain transmits the string encoding status of the current chain.

step 2: P'_a repeats the message from the previous step.

step 3: P_j acknowledges that it exists.

step 4: P'_j repeats the message of P_j received in the previous step.

After exchanging these messages the stations have to make decisions concerning their status. If there is no adversary, the following situations may happen. If there are agents of the current chain, ID j is *used* and the communication has not been scrambled, then position j joins the chain and P_j, P'_j take over the role of agents of the current chain. If no proper message is received at the first and the second step, then P_j, P'_j start a new chain. If no message is received at the last two steps, then agents of the current chain assume that the ID j is unused and retain the status of the agents, position j joins the chain as an unused ID. However, there is one exception: if energy usage of P_a, P'_a approaches the bound $b = O(\sqrt{\log N})$, they loose the status of the agents. Since no other stations take over, the current chain terminates in this way. Additionally, P_a, P'_a reach status *last-in-a-chain*.

Now we consider in detail what happens if an adversary scrambles communication (this is a crucial point since no activity of the adversary should yield an inconsistent view of the stations).

Case I: the 3rd message is scrambled. Since P_j does not receive its own message back, P_j and P'_j are aware of the attack. They must take into account that the agents of the current chain may hear only noise and conclude that the ID j is unused. In order to stay consistent with them, P_j and P'_j “commit a suicide” and from this moment do not participate in the protocol. So the situation becomes the same as in the case they have not existed at all.

Case II: the 4th message is scrambled, and the 3rd one is not. Then P_j behaves as in Case I. Only P'_j is unaware of the situation. However, it will never hear its partner P_j , so will never respond with a message (only sending such a message could make some harm.)

In Cases I and II the agents of the current chain either hear only noise or at most one unscrambled message. (The first case occurs also when the ID j is unused.) They know that ID j is unused or the sender with ID j commits a suicide. Therefore the agents retain their status of agents of the current chain.

Case III: the last two messages are not scrambled and at least one message of the first two is scrambled. In this case the agent P_a does not receive its message back. Also the stations P_j, P'_j are aware that either there was no agent or at least one message was scrambled. In this case P_j decides to start a new chain. At the same time P_a decides to terminate the current chain and reaches the status *last-of-the-chain*. Additionally, P_j can also acknowledge to P_i , that it’s transmission was clean of adversary if the attacker decides to attack the second message.

Note that a chain may terminate for two reasons. First, there can be a large block of unused ID’s such that the agents exhaust their energy resources. However, we shall see that this case happens with a small probability. The second case is that an adversary causes starting a new chain (Case III) or enforces suicides (Cases I and II). In the later case the agents of the current chain do not change, consequently this may lead to energy

exhaustion. However, we shall see that this may happen with a quite small probability. In the only remaining case a new chain starts exactly one position after the last position of the previous chain. This makes merging the chains easy.

Merging chains Before we start this part of execution each ID number j is replaced by $((j - 1 + f(s, i)) \bmod k) + 1$, where f is a cryptographic one-way function and s is the common secret of the stations electing the leader. A slight disadvantage is that in this way we split one chain, namely the chain containing information on ID's $k - f(s, i)$ and $k - f(s, i) + 1$. However the advantage is that the adversary cannot attack easily the same places as during building chains phase.

Procedure of merging chains takes $O(\log N)$ communication slots, each consisting of two windows of size $\Theta(\log^{3/2} N)$. For each chain, all its members know the first used ID of the chain. If it is t , then we can also label the whole chain by t .

Consider a chain t . Our goal is to merge it with other chains during communication slots $t - 1$ and t . With high probability the phase of building chains yields a chain ending at position $t - 1$. During merging procedure this chain may be merged with other chains. So assume that immediately before communication slot $t - 1$ a chain l ends at position $t - 1$. At this moment the following invariant holds: *each candidate with ID in chain l knows status of all ID's in this chain, each candidate P_j of chain t knows status of all ID's except the ID's larger than the first used ID j' larger than j* . Then:

slot $t - 1$: the member of chain l with status *last-in-a-chain* transmits a message encoding the status of all ID's in this chain, all candidates from chain t listen,

slot t : all candidates from chains l and t listen, except the station in chain t with status *last-in-a-chain*, which sends a message encoding status of all ID's in chain t .

In order to improve resistance against an adversary these two steps can be repeated a couple of times (this makes the time and energy cost grow). It is easy to see that if the messages come through, then all stations from chains l and t are aware of ID's in both chains. So in fact these chains are merged and the invariant holds.

If an adversary makes a collision at this moment, then the chains l and t do not merge and in subsequent communication slots chain t grows while chain l remains unchanged.

After merging the chains we hopefully have a single chain. Even if not, it suffices that there is a chain having information about at least $k/2$ ID's and corresponding to $c \cdot k$ candidates (where c is some constant) - of course there is at most one such a chain in a group. Finally, the members of this chain elect a group leader in a deterministic way based on secret s and information on used ID's in the chain.

3.1 Disabling the later groups

The idea of the "internal attack" against electing a group leader is to prevent emerging too large chains in the sub-procedure of building chains and preventing merging these chains during the second sub-procedure so that no resulting chain has length at least $k/2$. The attack is performed by a group of $w = \Theta(\sqrt{\log N})$ stations that succeeded to elect a group leader assigned to attack this particular group. Each of them will be involved in communication only a constant number of times. (So $\Omega(\log N)$ candidates

from the group with the group leader are enough and their energy cost grows only by an additive constant.)

The first modification is that each group contains w special positions that are reserved for *alien candidates*, not belonging to the group. The positions of aliens are located in w intervals of length k/w , one alien per interval. Precise location of an alien position within the interval depends on the secret s in a pseudo-random way.

While electing a group leader the alien positions are treated as any other positions in the group. However they may be “served” only by stations from a group that has already elected a group leader. So if no group leader has been elected so far, then “alien ID” work as unused ID and the election works as before. During attack the “alien stations” P_j and P'_j corresponding to alien position j behave as follows: when position j is considered they scramble step 1 (or 2) and perform correctly steps 3 and 4. So the agent of the current chain decides to stop the chain, making place for the chain starting at position j . When position $j+1, j+2, \dots$ are considered the alien stations do not transmit (as it would be the case if they adhere to the protocol). So no chain starting at position j is built, in the best case the next chain starts at position $j+1$.

When the chains of the group are merged, then there is no chance to merge the chains that are separated by at least one position - no further intervention of the aliens is necessary. So the aliens chop the chains into pieces of small size. It is easy to see that at least $w/2 - 1$ consecutive aliens must fail so that a chain of length greater than $k/2$ is built.

Let us discuss what can be done by an adversary to disable the internal attack (this would lead to election of multiple group leaders). The only chance of the adversary is to make collisions at steps 3 and 4, when alien stations P_j and P'_j send messages according to the protocol (consequently, Case I or II and not Case III will apply and the current chain will not terminate at this point). The chances of the adversary are not big, since it must guess correctly the alien positions and guess proper positions in the windows when these messages are transmitted. Finally, disabling separation of the chain at one place is not enough – the adversary must succeed in so many places that a chain of length at least $k/2$ emerges.

4 Proof of correctness

Algorithm behavior without adversary First we check that the algorithm succeeds with a fair probability, if there is no adversary. Except for the “internal attack” the only reason for which a group may fail to elect its leader is that there is a block of unused ID’s of size at least $b = \Omega(\sqrt{\log N})$. (In fact, this is a necessary condition, but not a sufficient one, a long block of unused ID’s close to the first or to the last position would not harm.) Indeed, otherwise each agent P_a encounters a used ID j for $j < a + b$, so before energy limit b of P_a is reached. Then P_j becomes the agent of the same chain and, in particular, the chain does not terminate. We see that if there are no large blocks of unused ID’s a single chain is constructed. It contains all ID’s except the unused ID’s in front of the first used ID (there are less than b of them according to our assumption). Merging chains in this case does not change anything. Finally, there is a chain of size at least $k - b > k/2$ so it elects the group leader.

Let us estimate the probability of creating a large block of unused ID's in a group. As noticed in Section 2, probability of success in a single Ethernet trial is at least μ , where $\mu \approx \frac{1}{e^z}$. There is a technical problem, since the success probabilities in different trials are not independent, but according to [3, Lemma 2] we can bound the probability of getting a block of unused ID's of length m by probability of m successes in a row for independent Bernoulli trials with success probability $1 - \mu$.

There are $k - b$ positions where a block of unused ID's of length at least b may start. Probability of emerging a block at such a point is at most $(1 - \mu)^b$. So we can upper bound the probability that there is at least one block of at least b unused ID's by $(1 - \mu)^b \cdot (k - b)$. Since $b = \Theta(\sqrt{\log N})$, $k = \Theta(\log N)$ this probability is $2^{-\Omega(\sqrt{\log N})}$.

Algorithm behavior in presence of an adversary In order to facilitate estimation of the error probability we assume that the adversary may have energy cost $z = O(\log N)$ during each computation part analyzed below.

Initial selection. The adversary can only lower the probability of success in Ethernet trials. Due to the random reassignment of ID's (as described in Section 2) the adversary does not know which ID it is attacking by making collisions during Ethernet trials. The reassignment permutation is pseudorandom, hence we may assume that a given ID is made unused by the adversary with probability at most $z/k < 1$. So the probability that a given ID becomes unused (whatever the reason is) is upper bounded by $(1 - \mu) + z/k$. If we adjust constants in a right way, then the last expression is a constant less than 1 and we may proceed as in the previous subsection to show that with probability at most $2^{-\Omega(\sqrt{\log N})}$ a block of at least b unused ID's can emerge.

Building chains Now we can assume that there is no block of unused ID's of length at least b . So if the adversary has to prevent electing a group leader it must prevent construction of a large chain. For this purpose, the adversary has to succeed in breaking a chain during building chains procedure (Section 3) and to prevent merging the chains at this position during merging procedure. The next lemma bounds the probability of such a case for any strategy of the adversary.

Lemma 1. *Probability of breaking the chains during the building chain procedure and preventing them to merge in at least one point during a single group election phase is bounded by a constant less than 1.*

Proof. From the point of view of an adversary breaking the chains is a game: first it chooses (at most) z out of k positions (these are positions attacked during building chain procedure). The next move in the game is performed by the network: there is a secret pseudorandom shift of positions. The distance of the shift remains unknown for the adversary. In the next move the adversary chooses again (at most) z positions. The adversary looses, if no position previously chosen is hit again (since there is no place where the chains are not merged again). Even if the adversary succeeds in attacking a place where the network tries to merge the chains, it must make collisions in proper places in two time windows: even if the whole energy z is invested in a single window, then success probability (for the adversary) in this window is $O(1/\sqrt{\log N})$. So the probability of succeeding in two independent windows is $O(1/\log N)$.

WRu uwaga: chce przedyskutowac lemat. moze byc lekko niezrozumialy, gdyz skaczemy pomiedzy roznymi wartosciami ktore liczymy

Let p_i be the probability of hitting by the adversary i positions during the third move of the game. In order to win, the adversary has to prevent merging the chains in at least one of these positions. Probability of such an event is $O(i \cdot \frac{1}{\log N})$. Finally, the total probability that the adversary wins the game is

$$O\left(\sum_{i=1}^z p_i \cdot \left(i \cdot \frac{1}{\log N}\right)\right). \quad (1)$$

Now consider all k shifts for fixed sets of z positions chosen by the adversary during the first and during the third move. Let us count the number of hits. Each position chosen during the third move hits each position chosen during the first move for exactly one shift. So the total number of hits equals z^2 . On the other hand, the number of hits can be counted by the expression $\sum_{i=1}^z (p_i \cdot k) \cdot i$. It follows that

$$\sum_{i=1}^z p_i \cdot i = \frac{z^2}{k} = O(\log N).$$

Hence the probability given by expression (1) is $O(2^{-\Omega(\sqrt{\log N})})$. □

Note that in order to prevent electing a leader the adversary has to succeed in every group. It happens with probability $O\left(\left(\frac{1}{\log N}\right)^v\right) = 2^{-\sqrt{\log N} \log \log N}$, since computations in different groups may be regarded as independent.

Disabling “internal attacks” There is still another chance for an adversary: He can assume that in at least one round the leader was chosen, and attack one of the following group election phases. His goal is to help the leader to be chosen by disabling internal attack of the group that already has a leader. In this way multiple leader emerge as a result of algorithm execution. For this purpose the adversary has to guess at least $\frac{w}{2} - 1$ “alien positions”. For each of these positions, he has to guess the right time slots inside two windows and collide twice with the “aliens”. The probability of such an event is less than:

$$\left(\frac{w}{2} - 1\right) \cdot \left(\left(\frac{1}{\sqrt{\log N}}\right)^2\right)^{\frac{w}{2}-1} = 2^{-\Omega(\sqrt{\log N})}.$$

5 Final Remarks

Our algorithm offers some additional features – it yields a group of $\Omega(\log N)$ active stations which know each other. This can turn out to be useful for replacing a leader that gets faulty or exhausts its energy resources.

Certainly, the solution presented is only a small step towards a secure network self-organization. However, some tasks that emerge there can be solved using the techniques presented.

References

1. T. Jurdziński, M. Kutylowski, J. Zatośniański, *Energy-Efficient Size Approximation for Radio Networks with no Collision Detection*, COCOON’2002, LNCS 2387, Springer-Verlag, 279-289.

2. T. Jurdziński, M. Kutylowski, J. Zatopiański, *Weak Communication in Radio Networks*, Euro-Par'2002, LNCS 2400, Springer-Verlag, 965-972.
3. T. Jurdziński, M. Kutylowski, J. Zatopiański, *Weak Communication in Single-Hop Radio Networks – Adjusting Algorithms to IEEE Standard*, full version of [2], submitted.
4. T. Jurdziński, M. Kutylowski, J. Zatopiański, *Efficient Algorithms for Leader Election in Radio Networks*, ACM PODC'2002, 51-57.
5. D. Estrin, *Sensor Networks Research: in Search of Principles*, invited talk at PODC'2002, www.podc.org/podc2002/estrin.ppt
6. K. Nakano, S. Olariu, *Randomized leader election protocols for ad-hoc networks*, SIROCCO'2000, Carleton Scientific, 253-267.
7. K. Nakano, S. Olariu, *Randomized Leader Election Protocols in Radio Networks with No Collision Detection*. ISAAC'2000, LNCS 1969, Springer-Verlag, 362–373.
8. K. Nakano, S. Olariu, *Uniform Leader Election Protocols for Radio Networks*, ICPP'2001, IEEE.
9. R. M. Metcalfe, D. R. Boggs, *Ethernet: distributed packet switching for local computer networks*, Communication of the ACM 19 (1976), 395-404.
10. I. Stojmenović (Ed.), *Handbook of Wireless Networks and Mobile Computing*, Wiley 2002.
11. D.E. Willard, *Log-logarithmic selection resolution protocols in multiple access channel*, SIAM Journal on Computing 15 (1986) , 468-477.