Communicating Mobile Nano-Machines and Their Computational Power

Wiedermann, Jiří
2008

Institute of Computer Science
Academy of Sciences of the Czech Republic

Communicating Mobile Nano-Machines and Their Computational Power

Jiří Wiedermann and Lukáš Petrů

Technical report No. 1024

May 2008

Pod Vodárenskou věží 2, 182 07 Prague 8, phone: +4202 6605 3520, fax: +420 286 585 789,
e-mail:jiri.wiedermann@cs.cas.cz

Institute of Computer Science
Academy of Sciences of the Czech Republic

Communicating Mobile Nano-Machines and Their Computational Power[1]

Jiří Wiedermann[2] and Lukáš Petrů[3]

Technical report No. 1024

May 2008

Abstract:

A computational model of molecularly communicating mobile nanomachines is defined. Nanomachines are modelled by timed probabilistic automata augmented by a severely restricted communication mechanism. We show that for molecular communication among such machines an asynchronous stochastic protocol originally designed for wireless communication in so-called amorphous computers with static computational units can also be used. We design an algorithm that using randomness and timing delays selects with a high probability a leader from among sets of anonymous candidates. This enables a simulation of counter automata proving that networks of mobile nanomachines possess universal computing power.

Keywords:
molecular communication; nanomachines; timed probabilistic automata; universal computing

[1]This research was carried out within the institutional research plan AV0Z10300504 and partially supported by the GA ČR grant No. 1ET100300419 and GD201/05/H014.

[2]Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod Vodárenskou věží 2, 182 07 Prague 8, Czech Republic

[3]Faculty of Mathematics and Physics, Charles University, Malostranské náměstí 25, 118 00 Prague 1, Czech Republic, lukas.petru@st.cuni.cz

1 Introduction

Nanomachines are molecular cell-sized artificial devices or engineered organisms produced by self-assembly or self-replication, capable of performing simple tasks such as actuation and sensing (cf. [6]). Construction of various nanomachines seems to be entirely within reach of current nano- and bio-technologies (cf. [4]). Once constructed and endowed with a certain sensor and actuating abilities communication among nanomachines becomes an important problem. Design of the corresponding molecular mechanism presents a great challenge for nano-technology, bio-technology, and computer science. In molecular biology there is an explosively growing field dealing with molecular communication. Here, the respective research is interested almost exclusively in the (bio)chemical aspects of existing communication mechanisms in biological systems. It seems that almost no attention has been paid to the algorithmic aspects of the respective communication process. Communication at a nanoscale substantially differs from communication scenarios and frameworks known from classical distributed systems. Key features of traditional versus molecular communication have been neatly summarized in [6]. In molecular communication, the communication carrier is a molecule; chemical signals are extremely slowly propagated by diffusion in an aqueous environment with low energy consumption. In general, it is not necessary that the signal will reach all targets: a majority will do. This is to be compared with traditional communication via electromagnetic waves where electronic or optical signals are propagated at light speed with high energy costs in an airborne medium and message delivery to all targets is required.

In what follows we will concentrate onto a scenario in which nanomachines form an autonomous system operating in a closed liquid environment without external control (a similar scenario is also considered in [6]). The system consists of a finite number of nanomachines freely "floating" in their environment that interact via molecular communication creating thus a kind of ad-hoc network.

From an algorithmic viewpoint several questions immediately arise within such a system. What protocol is used for molecular communication? What happens if there are more nanomachines communicating concurrently? Would it be necessary to synchronize them so that one would act as a sender and the other as a receiver? How do we know that a target machine has received a signal? If we allow a finite number of different signals (types of molecules) by which the machines can communicate, what happens if a machine having several receptors detects different signals at different receptors at the same time? What happens when a (would be) receiving machine is engaged in sending a signal? What are the computational limits of the underlying system?

It is the purpose of the present paper to answer such questions. In order to do so three things are needed: (i) a more detailed algorithmic (computational) model of a nanomachine, (ii) a communication protocol, i.e., an algorithm controlling the communication behavior of each nanomachine, and (iii) a simulation procedure showing the relation of networks of communicating nanomachines to a standard (classical) model of computations whose computational power is known.

Our paper is related to the recent research in amorphous computing and population protocols, cf. [10] and [3]. Whereas [10] dealt only with a model of "static" amorphous computer with its processors fixed in the plane, the present paper considers the case of "mobile" amorphous computing. Our results also address the open problem from [3] concerning the power of one-way communication in a model with anonymous agents. We believe that our results also contribute to the problem of how far can one go in "simplifying" the models of distributed universal computation. Our model of one-way communicating timed probabilistic mobile automata seems to represent a kind of the local minimum along these lines.

The structure of the paper is as follows. In Section 2 the nanomachine computational model is introduced and it is shown that communication protocols designed in [10] can be also used for the case of molecular communication. In Section 3 a simulation of a counter automaton by nanomachines is shown. Conclusions are described in the closing Section 4. Here the super-Turing computing power of infinite families of sets of communicating timed probabilistic automata is indicated.

2 Computational Model

Nanomachine computational model. From computational point of view we will see each nanomachine as a timed probabilistic finite-state automaton. In essence this is a finite state automaton augmented with quantitative information regarding the likelihood that transitions occur and the time at which they do so. Timing is controlled by a finite set of real valued clocks. The clocks can be reset to 0 (independently of each other) with the transitions of the automaton, and keep track of the time elapsed since the last reset. The transitions of the automaton put certain constraints on the clocks values: a transition may be taken only if the current values of the clocks satisfy the associated constraints (cf. [2], [7]).

A biomolecular realization of a probabilistic automaton has been described in [1]. As far as timing mechanisms of biological automata are concerned, in biology there is a vast body of research dealing with biological oscillators and clocks controlling various biological processes in living bodies and it is quite plausible that such mechanisms could also be considered in nanosystems.

Our version of probabilistic timed automaton will in fact be a probabilistic timed transducer (Mealy automaton) having a finite number of input ports (receptors) and output ports (emitters). The signal molecules will be represented by elements of automaton's finite working alphabet. Now we state the condition under which the nanomachines can communicate. These condition are designed so as to make the underlying molecular mechanism as simple as possible.

1. Each automaton is able to work in two modes: in the receiving mode, reading (in parallel) the symbols (molecules) from its input ports, and in the sending mode, writing the same symbols to its output ports. A read operation is successful if and only if all symbols at all input ports are identical; otherwise the read operation fails. Except of these ports, a nanomachine can also have receptors detecting other than signaling molecule stimuli, and other actuators doing some job corresponding to the purpose to which the nanomachine has been designed.

2. In the sending mode, an automaton releases signal molecules of the same kind through all its output ports; these molecules diffuse in the environment and eventually can reach the input ports of an automaton in a receiving mode; if this is the case and the target automaton detects at all its ports only signal molecules of one kind, the signal is accepted by the receiving automaton.

3. After a certain time, signal molecules disintegrate into other molecules which are not interpreted by the automata as signal molecules. In their life time, signal molecules can travel, by diffusion, in average a certain maximal distance called *communication radius*.

4. The automata work asynchronously — there is no global clock in the system. The actions of each automaton are governed by automaton's local clock. The local clocks in the automata are not synchronized, however, they all "tick" (roughly) at the same rate since they all are realized by the same biochemical oscillators. A slight variation in the clock rate does not harm our purposes.

5. The automata have no identifiers, i.e., for communication purposes all senders and receivers "look the same".

6. The automata are equipped by a finite set of timers (clocks) that are assigned to certain transitions. These timers are started with the start of the automaton at hand and are updated at each automaton's step. Upon expiration of time to which the timer has been initially set the transition controlled by the timer at hand is allowed to be realized. A timer can be reset again to zero by a special transition. Values to which the timer are set may depend on the total number of communicating automata.

Note that the previous conditions are quite restrictive. Condition 1 means that an automaton cannot simultaneously be in a sending and receiving mode; a signalling molecule reaching an input port of an automaton in a sending mode will not be detected by that automaton. Condition 2 essentially says that if a broadcast from one automaton is "jammed" by a broadcast of an other automaton broadcasting different signal, then the receiving automaton does not accept any signal.

Condition 3 ensures that signal molecules cannot "roam" for ever in the environment and that current signals eventually prevail over the old ones in the communication radii of the automata. Condition 4 says that we cannot count on automata switching their sending and receiving modes synchronously. Also note that our automata can move in their environment, either passively, due to the external forces (e.g., in a bloodstream), or actively, like some bacteria. This by itself, but the more so in conjunction with condition 5, prevents whatever kind of "acknowledgements" of received messages.

Definition 1 *A multiset of communicating identical nanomachines is called a* nanonet*.*

Now the task is to design a communication protocol that, under the previous conditions, within a nanonet will enable, with a high probability, signal delivery from a sending automaton to all automata within a communication radius of a sending automaton.

Basic Protocol. There is a similarity between the computational model of "static" amorphous computer from [10] and the present model. Both models are asynchronous and able to exploit randomness. What makes the difference is the mobility of the basic computational units in the present model and their fixed number of states, independent of the number of communicating machines compensated, to some extend, by clocks (timers) by which automata are equipped. The value of these timers can depend on the number of automata. Yet from the viewpoint of the design of basic communication primitives the similarities prevail and small modifications allow using the definitions and results from [10] concerning the basic communication protocols also in the new context. Therefore we only present the respective results from [10] "translated" to the context of nanomachines. For the proofs of the corresponding theorems the reader is referred to the original paper [10].

Consider the so-called *communication graph* G of a given nanonet whose nodes are nanomachines (automata) and edges connect those automata which are within the communication radius of each other. The size of G will be measured in the number n of its nodes. Obviously, the topology of G depends on time since in general our automata move. In order to enable communication among all (or at least: a majority of) available automata in a nanonet most of the time G must have certain desirable properties. What we need are connected graphs with small diameter and a reasonable node degree. The most important property of G is its connectivity: connectivity is a necessary condition in order to be able to harness all processors. Graph diameter $D = D(n)$ bounds the length of the longest communication path. Finally, the node degree Q (i.e., the maximal neighborhood size of a node) determines the collision probability of signals.

Definition 2 (Well-behaved nanonet) *A nanonet $\mathcal{N}$ whose underlying computational graph of size n stays connected and whose diameter and maximal neighborhood size stay bounded by $D = D(n)$ and a constant Q, respectively, all the time, is called a* well–behaved (n,D,Q)-nanonet, *or simply a well-behaved nanonet, if its parameters are clear from the context.*

The requirements that connectivity, bounded diameter and bounded neighborhood of a graph with dynamic topology remain preserved all the time are quite strong and one can hardly imagine that in "practice" they will be fulfilled, indeed. Nevertheless, we need the invariance of these properties in order to be able to analyze the correctness and efficiency of the communication primitives we will deal with in a sequel. Later, after presenting these primitives we will argue that they are sufficiently "robust" in order to operate correctly and with a similar efficiency also in instances of nanonets that occasionally, for short time, deviate from their well-behaved properties.

Assuming that all nodes of a nanonet could participate in a communication an algorithm for sending a signal from a node to an other node within the first's node communication radius is needed.

Protocol Send: A node is to send a signal s with a given probability $\varepsilon > 0$ of failure. We want the protocol to work correctly under the assumption that all nodes work concurrently, asynchronously, using the same protocol and hence possibly interfering one with each other's broadcast.

The idea is for each node to broadcast sporadically, minimizing thus a communication collision probability in one node's neighborhood. This is realized as follows. Let T be time to transfer a signal between any two neighbors at the communication radius distance. Each node has a timer measuring *timeslots* (intervals) of length $2T$. During its own timeslot, each node is allowed either to listen till

the end of its timeslot, or to send a signal at the very beginning of its timeslot and then listen till the end of this timeslot. At the start of each timeslot a node sends s with probability $p = 1/(Q+1)$ and this is repeated for $k = O(Q\log(1/\varepsilon))$ subsequent timeslots. The probability of a node sending s is controlled by the transition probability of the respective probabilistic automaton.

Theorem 1 (Sending a signal) *[10] Let $\mathcal{N}$ be a well–behaved (n, D, Q)-nanonet, let $1 > \varepsilon > 0$ be an priori given allowable probability of failure. Assume that all nodes send their signals asynchronously according to* Protocol Send. *Let X be a node sending a signal s and Y be any of X's neighbors. Then s will be received by Y in time $O(Q\log(1/\varepsilon))$ with probability at least $1-\varepsilon$.*

Note that in principle the protocol can work for any value of $1 > \varepsilon > 0$. However, in such a case we also need that the sending automaton will be able to send s $k = k(\varepsilon)$ times in a row. It is here where we need to invoke a timer possessed by our timed probabilistic automaton: this timer is "set" to the interval $\approx kT$ and is used for controlling the automaton's communication actions.

Broadcasting protocol In order to send a signal from a node to any other node of a nanonet which is not in the communication radius of the sending node we use the idea of flooding the network by that signal, i.e, broadcasting the signal to all nodes of the network.

Algorithm Broadcast. The main idea of flooding technique is to use every node reached by a given signal as a "retranslation station" that distributes the signal further through the network. After retransmitting a signal each node stops retransmitting any signals equal to the last retransmitted signal — it locks itself with respect to that signal. However, any signal different from the last retransmitted signal is again retransmitted and afterwards the node locks itself again with respect to that last signal, etc.

The following algorithm will be used to deliver a signal s from a node X in the network to all remaining nodes which are not locked with respect to s. To broadcast s to the network, X sends s using *Protocol Send* with probability ε/n of failure. Upon receiving s, any node sends s using *Protocol Send* with failure probability ε/n and afterwards it locks itself with respect to s. The locking mechanism is implemented straightforwardly: each node remembers the last sent signal and ignores it if it has received it again—it "locks itself" with respect to that signal.

Theorem 2 (Broadcasting) *[10] Let $\mathcal{N}$ be a well-behaved (n, D, Q) nanonet. Then, for any ε : $0 < \varepsilon < 1$,* Algorithm Broadcast *delivers s to each node of $\mathcal{N}$ that has not been locked with respect to s in time $O(DQ\log(n/\varepsilon))$ and with probability $1-\varepsilon$. Afterwards, all nodes in $\mathcal{N}$ will be in a locked state with respect to s.*

Note that this time, in order to achieve the failure probability ε of the broadcasting algorithm *Protocol Send* must be performed with error probability ε/n. This calls for repeating each send operation in a node $k = O(Q\log(n/\varepsilon))$ times—i.e., this time k should grow not only with a decreasing ε, but also with increasing size of the nanonet. Similarly as in the case of *Protocol Send,* the respective timer is realized by means of the timing mechanism of the underlying automata. The necessity of having a timer dependent on n makes our nanomachines using the previous broadcasting algorithm a non-uniform computational model. Note, however, that the non-uniformity does not concern the number of an automaton's states — this number does not depend on n. What depends on n is the size of parameters to which clocks (timers) must be initially preset.

Now it is time to return to the problem of non well-behaved nanonets. From the description of the communication protocol it is seen that occasional short-time connectivity "interruptions" could not harm the communication as long as constant k controlling the repeated sending trials is set high enough to overcome the interruption periods. A sufficiently high value of k will also help to overcome occasional local increase of maximal neighborhood size which can block efficient communication. Finally, a generous estimate of the nanonet diameter (which is always bounded by n) that is in fact determined by the shape of the closed environment in which the nanomachines move will help to overcome the time variations of instantaneous nanonet diameter.

3 Distributed Computing Through Nanomachines

Computational scenario. Let us assume that each nanomachine gets its "own" input from the domain $\{0, 1\}$ through its sensors; this input can be used in the subsequent "collective" data processing. If outputs from the nanomachines are also restricted to the domain $\{0, 1\}$ then the respective nanonet can be seen as a device computing functions with their inputs and outputs represented in unary. We show that a nanonet of size $O(n)$ can simulate a counter machine with a constant number of counters each of size $O(n)$, with high probability. An essential operation for representing counters in a nanonet will be operation of leader selection.

Leader selection. In what follows we will frequently need to "address" certain subsets of nanomachines using *Algorithm Broadcast*. The subsets of machines at hand will be distinguished from other machines by the internal states of these machines. A suitable form of distinguishing subsets of machines is to consider composite states, i.e., of pairs of states from cartesian product $Q_1 \times Q_2$ of finite state sets. Then the machines whose first state component $q \in Q_1$ is fixed while the second state component varies over Q_2 define a subset of machines called the *multiset of machines determined by state* q. In cases when we are not especially interested in the state specifying the set of determined machines we will simply speak about the set of determined machines.

Consider now the following situation: in a nanonet there is one distinguished nanomachine called the *base*. The input data reside in a set of determined machines, one input bit per machine. The base is expected to "organize" the data processing in the nanonet. Doing so it will repeatedly use the following pattern: it will send appropriate signals — instructions throughout the net at appropriate times and collect the reactions to these signals from the individual nanomachines. For sending signals from the base to the rest of the nanonet and from specific nanonet nodes back to the base *Algorithm Broadcast* will be used. As seen from Theorem 2, flooding the net with a signal from the base (and indeed, from any nanomachine) takes time $cDQ \log(n/\varepsilon)$ for some constant $c > 0$. That is, only after elapsing this time we can expect with probability $1 - \varepsilon$ that the farthest nanomachine will obtain a signal sent by any other nanomachine(s). The same time is needed in order a response (if any) from some machines to arrive back to the base. Thus, after sending a signal the base (or any other machine) has to wait for $cDQ \log(n/\varepsilon)$ time units until the next signal can be sent, or $2cDQ \log(n/\varepsilon)$ until a reaction for a signal can be obtained. The necessary timing mechanisms can again be handled, as in similar cases before, via timers. Moreover, when performing a sequence of such operations then in order to keep the prescribed failure probability low the probability of error of each flood operation must be lowered in the inverse proportionality with the length of the sequence (since the failure probabilities accumulate). Working in this way the nanomachines in the net eventually reach the state when the desired result gets represented again in some determined machines and the processing of a given input can terminate.

A *counter* representing a natural number m is a (multi)set C consisting of exactly m determined machines. For the simulation of a counter machine we are after three operations over counters are of importance: testing a counter for zero, and increasing or decreasing a counter by one.

The former operation is easy. To test counter C for zero the base floods the net by a signal asking each nanomachine to respond (i.e., to send the signal to all members in the net) if and only if it belongs to C. In this way the base can learn that C represents a positive value.

Adding a one to a counter C is a two phase operation. In the first phase, a single nanomachine—so-called *leader*—from among those not in C must be selected and put into a unique state distinguishing it from all nanomachines in the net. In the second phase, the state of this distinguished machine is set to the state in which all machines of C find themselves at that time. By this the cardinality of the current set C is increased by one. In the case of counter decrement we proceed similarly, except that the leader is sought within C and after finding it this member is "deleted" from C by making its state different from the rest of C.

Hence, the essence of both "arithmetical" operations boils down to leader selection in a determined set. Without lost of generalization we will now assume that our nanomachines make use of composite states from cartesian product $Q_1 \times Q_2$. In a semi-informal way in Fig. 3.1 we describe a randomized algorithm that from a set $C[q, 1] \neq \emptyset$ of all nanomachines from $\mathcal{N}$ that are in state $[q, 1]$, selects with a high probability a single nanomachine that remains in state $[q, 1]$, whereas all other nanomachines

randomized algorithm LEADER(C):M;
comment $C \neq \emptyset$ is the set of all nanomachines from $\mathcal{N}$ in state $[q,1]$, there is no nanomachine in $\mathcal{N}$ in state $[q,0]$; LEADER puts all nanomachines except one from C into state $[q,0]$ leaving a single nanomachine $M \in C$ in state $[q,1]$;
const: ℓ;

Rename $C[q,1]$ to $C[q,2]$;

Phase 1: Random splitting of $C[q,2]$*:*
iterate:
for each machine in $C[q,2]$ **do begin**
  choose randomly $x \in \{0,1\}$; **if** $x = 0$ **then** enter state $[q,2]$ **else** enter state $[q,3]$;
end;
if $C[q,2] \neq \emptyset$ **and** $C[q,3] \neq \emptyset$ **then begin** rename $C[q,3]$ to $C[q,1]$; **go to** iterate **end**
if $C[q,3] \neq \emptyset$ **then** rename $C[q,3]$ to $C[q,2]$;

Phase 2: Checking uniqueness:
test:i:=1;
while $i \leq \ell$ **do begin**
  for each machine in $C[q,2]$ **do begin**
    choose randomly $x \in \{0,1\}$;**if** $x = 0$ **then** enter state $[q,2]$ **else** enter state $[q,3]$;
  end;
  if $C[q,2] \neq \emptyset$ **and** $C[q,3] \neq \emptyset$ **then begin** rename $C[q,3]$ to $C[q,1]$; **go to** iterate **end;**
  if $C[q,3] \neq \emptyset$ **then** rename $C[q,3]$ to $C[q,2]$;
  $i := i + 1$
end;
rename $C[q,2]$ to $C[q,0]$;
return $C[q,0]$;

Figure 3.1: Determining a leader

are put into state $[q,0]$. The algorithm works correctly under the assumption that previously there were no nanomachines in $\mathcal{N}$ in state $[q,0]$.

Algorithm LEADER works as follows. In Phase 1 the algorithm "renames" set $C[q,1]$ to $C[q,2]$. Then, depending on the outcome of a coin tossing LEADER randomly splits the input set $C[q,2]$ into two disjoint sets: set $C[q,2]$ and set $C[q,3]$, respectively. If both sets are non-empty we "return" the machines originally in state $[q,1]$ back to this set by renaming $C[q,3]$ to $C[q,1]$ and iterate the previous splitting process with $C[q,2]$. The renaming of machines is necessary in order the iteration process could distinguish between machines eliminated in the previous iteration from those proceeding to the next round. The iteration terminates once the splitting operation does not succeed in splitting the input set into two non-empty subsets. At this place the remaining non-empty set is "suspected" to be a singleton set with a constant probability.

Let us estimate the expected number of iterations needed to reach this state. Suppose that initially C contains $n \geq 1$ nanomachines. Let $T(n)$ denotes the expected number of "splitting rounds" of algorithm LEADER until it generates a leader (a set denoted as $C[q,2]$ at the end of Phase 1 in algorithm LEADER) from among n candidates. Then, clearly, $T(1) = 0$ and for $n \geq 1$, $T(n) \leq 1 + 2^{-n} \sum_{i=0}^{n} \binom{n}{i} T(i)$ since after making the first splitting round we have eliminated $n - i$ leader contestants with probability $2^{-n}\binom{n}{i}$ for any $i : 0 \leq i \leq n$. One can verify the solution of the previous recurrence relation in form $T(n) = O(\log n)$.

This leads us to a conjecture that LEADER needs about $O(\log n)$ rounds in order to select a singleton set with constant probability. Indeed, this is the case as shown in the following theorem from [5]. There the idea of repeatedly removing a randomly selected subset of candidates from leader candidacy was used in a different situation — namely as the basic protocol (in place of our *Protocol Send*) for eliminating nodes from a node's neighborhood. The goal was to eventually achieve a non-

conflicting message delivery in a communication model similar to ours except that synchronicity of processors was assumed. Nevertheless, the complexity analysis of the underlying procedure also applies in our case:

Theorem 3 *[5] Let $p_k(n)$ be the probability that a singleton set will be obtained after applying exactly $k \geq 0$ steps of random splitting in algorithm LEADER (Phase 1), starting with a set of cardinality n. Then, for $k \geq 2\lceil \log n \rceil$, $p_k(n) > 1/2$.*

Thus, after performing Phase 1 we have our leader with probability greater than $1/2$. However, for our purposes—operating the counters—this result is not yet fully satisfactory since we need a leader selection algorithm with an error probability that can be made arbitrarily small.

The essence of our problem is this: in cases when in Phase 1 the independent coin toss in each machine has ended with the same outcome, our set $C[q, 2]$ contains more than one member at the end of Phase 1. In this case the original set has not been split but the splitting iteration had terminated. This is why the algorithm proceeds to Phase 2 where the cardinality of $C[q, 2]$ is further tested.

The test in Phase 2 tries repeatedly, ℓ times, to split set $C[q, 2]$ by a randomized process whose goal is to break symmetry in case when the cardinality of the tested set is greater then 1. We show that if this test fails in ℓ trials, then the probability that $C[q, 2]$ is not a singleton is less than $2^{-\ell}$. Assume that initially the cardinality of $C[q, 2]$ was $r \geq 2$. Then the probability that $C[q, 2]$ will be not split in ℓ independent trials equals the probability that tossing ℓ times r coins we get in each toss either all heads or all tails. The probability of such a result in a single toss is 2^{1-r} and in ℓ tosses $2^{(1-r)\ell} \leq 2^{-\ell}$.

If from among ℓ splitting trials one is successful then the algorithm returns to Phase 1.

Thus, summarizing, after passing ℓ splitting trials Phase 2 of LEADER returns a singleton set with failure probability at most $2^{-\ell}$.

The last thing we have to ensure that with growing n the error probability of leader selection will tend sufficiently fast to zero. To this end we can select constant ℓ to be equal to constant k from *Algorithm Broadcast*: $\ell = O(Q \log(n/\varepsilon)$. This choice would lead to a unique leader selection with probability $2^{-O(Q \log(n/\varepsilon))} = (n/\varepsilon)^{-O(Q)} = (n/\varepsilon)^{-c}$ for some constant $c > 0$.

Theorem 4 *In a well behaved (n,D,Q)-nanonet for any $\varepsilon : 0 < \varepsilon < 1$, algorithm LEADER determines a leader in expected time $O(DQ \log n \log(n/\varepsilon))$ with probability of error $max\{1, O(\varepsilon \log n + (\varepsilon/n)^c)\}$.*

Universal computation. Having an algorithm for performing the basic operations over counters it is a straightforward matter to simulate a counter machine (also known as Minsky machine—cf. [8]) by a set of nanomachines. As far as the complexity characterization of the simulation is concerned note that the computation of a counter machine with a constant number of counters of length $O(n)$ and unary input of length n can be of at most polynomial time complexity $p(n)$ (since a counter machine can enter only polynomially many different configurations). From Theorem 4 it follows that choosing failure probability $\varepsilon > 0$ in *Protocol Send* and performing $p(n)$ operations over the counters the failure probability of the simulation algorithm can be as high as $c_2 p(n)(\varepsilon \log n + (\varepsilon/n)^{c_1})$ for some constants c_1 and c_2. The expected simulation time will be $O(DQp(n) \log n \log(n/\varepsilon))$. Thus, in order to keep the overall failure probability of simulation low one has to fix the value of ε appropriately.

Theorem 5 *Let $\mathcal{M}$ be a counter machine with input of size n operating with a constant number of registers of size $O(n)$ in time $p(n)$. Then for any $\varepsilon : 0 < \varepsilon < 1$ there exist constants $c_1 > 0$ and $c_2 > 0$ and a nanonet $\mathcal{N}$ consisting of $O(n)$ nanomachines using* Protocol Send *with failure probability ε such that $\mathcal{N}$ simulates $\mathcal{M}$ for inputs of size at most n in expected time $O(DQp(n) \log n \log(n/\varepsilon))$ with probability of failure bounded by $\max\{1, c_2 p(n)(\varepsilon \log n + (\varepsilon/n)^{c_1})\}$.*

Note that counter machines are universal computing devices, and hence the last theorem claims that a nanonet can perform universal computations with a high probability. Most probably, in practice nobody would consider performing a universal computation by nanonets. Nevertheless, our result shows that in principle a nanonet can perform whatever computational task arising in a practical application.

4 Conclusion

It has been stated that network protocols increasingly rely on the use of randomness and timing delay [7]. This has also been the case of communicating nanomachines where randomness and timing had played the crucial role in the design of the basic communication protocol. The underlying computational model—probabilistic timed finite state automata—have increasingly often been used in modelling and analyzing of non-uniform complex systems which make use of intensive communication among their components. Such systems frequently arise in computer networks, electronic devices and biological organisms. In these applications, the probabilistic timed automata are mainly used for formally capturing the computational behavior of the underlying systems and eventually enabling a formal (even automatic) verification of systems' correctness. So far it seems that in the computability or computational complexity theory systems of probabilistic timed communicating automata have not been considered as a universal computational model. The current paper shows that it is worth to do so: the resulting model is among the simplest ones as far as the simplicity of its components and its potential for its realization by various means, most notably by nanotechnologies, is concerned. The computational universality of communicating nanomachines offer yet an other example of a non-uniform computational model that is captured by the so-called *Extended Turing Machine Paradigm* coined by van Leeuwen and Wiedermann [9], [11], claiming that *"A computational process is any process whose evolution over time can be captured by evolving automata or, equivalently, by interactive Turing machines with advice."* Obviously, for each input of size n a multiset of timed probabilistic automata can be simulated by a Turing machine with advice which for each n returns the clocks of the simulated automata; the value of these clocks depends only on the input length. In this context it is amusing to observe that the clock give our set of communicating timed automata super-Turing computing power. Namely, for each n there exists (but cannot be computed) a clock that stops only after the time consumed by the longest computation, taken over all Turing machines of size n and all inputs of size at most n, is exhausted. An infinite family of sets of communicating timed probabilistic automata equipped by such clock (one set and one clock for each n) could then solve the halting problem.

Bibliography

[1] Adar R., Benenson Y., Linshiz G., Rozner A, Tishby N. and Shapiro E.: Stochastic computing with biomolecular automata. Proc. Natl. Acad. Sci. USA, 101, pp. 9960-65, 2004

[2] Alur, R., Dill, D.L.: A Theory of Timed Automata Theoretical Computer Science 126:183-235, 1994

[3] Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M. J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. Distributed Computing 18(4): 235-253 (2006)

[4] Bath, J., Turberfield, A.J.: DNA nanomachines, Nature nanotechnology, Vol. 2, May 2007, pp. 275 - 284

[5] Bar-Yehuda, R., Goldreich, O., Itai, A.: On the Time-Complexity of Broadcast in Multi-hop Radio Networks: An Exponential Gap Between Determinism and Randomization. J. Comput. Syst. Sci. 45(1): 104-126 (1992)

[6] S. Hiyama, Y. Moritani, T. Suda, R. Egashira, A. Enomoto, M. Moore and T. Nakano: Molecular Communication, In: Proc. of the 2005 NSTI Nanotechnology Conference, 2005.

[7] Kwiatkowska, M., Norman, G., Parker, D., Sproston, J.:. Performance Analysis of Probabilistic Timed Automata using Digital Clocks. In: Formal Methods in System Design, 29, Springer, pp. 37-78 August 2006.

[8] Minsky, M.: Computation: Finite and Infinite Machines. Prentice-Hall. 1967

[9] J. van Leeuwen, J. Wiedermann, The Turing machine paradigm in contemporary computing, in: B. Enquist and W. Schmidt (Eds), *Mathematics Unlimited - 2001 and Beyond*, Springer-Verlag, Berlin, 2001, pp. 1139-1155.

[10] Wiedermann, J., Petrů, L.: On the Universal Computing Power of Amorphous Computing Systems. To apper in Theory of Computing Systems, Springer, New York, 2008

[11] Wiedermann, J., van Leeuwen, J.: How We Think of Computing Today (Invited Talk). In: Proc. of the Fourth Conference on Computability in Europe, CiE 2008, Athens. To apper in LNCS, Springer, 2008