

# Distributing Identity

*Published in IEEE Robotics and Automation Magazine,  
Vol. 6, No. 1, March 1999, pps 49-56.*

Joshua R. Smith

jrs@media.mit.edu  
Physics and Media Group  
MIT Media Lab  
20 Ames Street  
Cambridge, MA 02139  
USA

**Abstract.** This paper considers distributed protocols for *assigning* IDs. Normally the maker of an RFID system would simply assign each ID sequentially in the factory, or the user would manually burn it in to EPROM. But it may be cheaper and simpler to manufacture units identically, and build the *capability* to become unique into each one. I describe three distributed protocols which make use of hardware random number generation to create unique IDs. In the first, trivial scheme, random IDs of sufficient length are independently chosen by all units. This requires no communication among the units, so it could be used to assign identities in circumstances in which central coordination or communication among the units is impossible. However, because no communication or central coordination is used, the IDs may be quite long. I then present a second protocol which requires a modest amount of communication to assign a set of unique, short IDs. Finally, I present a protocol with more substantial communication requirements that accomplishes the same ID assignment task in much less time.

## 1 Introduction and Motivation

In the next several years, computational, sensory, and communication capabilities will diffuse out of their present home in beige boxes on desktops and into everyday objects such as furniture, clothing, and other “non-technological” objects. As the cost of electronics continues to drop, the number of activated, networked devices will grow rapidly. Each device sharing a particular multiaccess channel will need a unique identifier for that channel. Present communication protocols such as Ethernet specify that the manufacturers must coordinate with one another in order to avoid assigning the same ID twice.[8] This paper explores methods by which the *devices* could coordinate with one another to manage ID assignment dynamically and automatically.

In particular, this paper is an investigation of distributed protocols that utilize physical sources of symmetry breaking to enable a network of initially

identical units to acquire the unique identities required for point-to-point communication over a shared resource such as a bus or common RF band (multi-access channel). I present several protocols, compare their resource use (time, random bits, space, communication), and note a trade-off among these measures of algorithm complexity.

The goal, once again, is that each initially identical unit have a unique identifier by the end of the protocol. Unique identifiers, or at least sources of symmetry breaking, are necessary for point-to-point communication over a shared channel: if two units have duplicate IDs, both will respond to the same set of messages, and when both try to respond simultaneously, there is no way for them to break the deadlock, since the two units begin in the same state and proceed deterministically, in particular pausing and retransmitting at the same time. The “random exponential backoff” used to recover from collisions in CSMA/CD networks (eg Ethernet) is impossible without either a source of thermodynamic randomness, or an identifier such as the so-called MAC address, a unique number associated with each Ethernet adapter, that can seed a pseudo-random number generator.

Thus the protocols must use uncorrelated noise sources, together with communication, to move the system from a perfectly correlated state (all units with ID 0, say) to create a kind of perfect anti-correlation, in which each unit is in a unique state.

An obvious method for generating a unique ID is to hardcode it into the unit’s ROM. In this solution, the symmetry is broken before use, in the course of the manufacturing process. Effectively this means that the manufacturing process for each unit is slightly different. Clearly the process of manufacturing a large number of unique units is more complex than that of manufacturing a large number of identical units, and this complexity is undesirable because it translates to increased cost. For an item as expensive as an Ethernet card (around \$50), the cost of the coordination activity required to ensure that each receives a unique ID is small compared to the cost of the unit. For a \$5 sensor unit, this coordination cost is relatively higher, and for active RFID (\$1) and passive tags (\$.10), the coordination cost begins to dominate the other manufacturing costs.

Dallas Semiconductor in fact sells a “silicon serial number” (for a around \$1 in large quantities) that is essentially a 56-bit unique ID that can be read by a one-wire serial protocol. Each unit made is guaranteed to have a different ID than all others.[7] Effectively this manufacturer is mass producing broken symmetry, centralizing the manufacturing burden of producing unique items into a single production line.

Since the number of “smart” devices with unique IDs will be growing rapidly in the next several years as their cost drops, the idea of building the *capability* to break symmetry into each device, of moving the symmetry breaking step from the production line to the device itself, may become increasingly attractive. An interesting practical question is whether dynamic symmetry breaking schemes such as the ones we will present can be more efficient economically (ie cheaper) than the hard-wired, “mass produced broken symmetry” represented by Dallas’ silicon serial number.

In addition to the possible cost savings, there is another practical motivation for understanding dynamic symmetry breaking protocols: they may enable anonymous communication between units, or between a unit and a public server. Since the identities are determined after the units have been deployed, there is no possibility that the manufacturer or other entity could maintain a list associating IDs with owners. However, in order to evaluate the protocols presented here from the perspective of preserving anonymity, we would have to consider the effect of adversaries who may not obey the protocol, which is beyond the scope of this paper.

## 1.1 Biological Examples

There are several well known biological examples of distributed symmetry breaking protocols. The “jamming avoidance” behavior of the South American weakly electric fish *Eigenmannia* is one such example.[3] Members of this species generate continuous sinusoidal electric fields of constant frequency for both sensing and communication purposes. For a fish to sense its environment effectively, it needs its own electrolocation frequency. When the sensing frequencies of two fish overlap enough to interfere with electrolocation, both fish can sense this collision, and the resulting beat frequency. The fish with the higher frequency raises its frequency still higher, and the lower frequency fish drops its frequency, resolving the conflict. Of course, this may create new frequency conflicts, which are then resolved by the same algorithm. In this way, the frequency spectrum used by the fish can be adaptively re-allocated when a new individual is introduced into a habitat.

Another natural example that has been discussed by Rabin in the computer science literature is the case of mites of genus *Myrmoysus*, which feed on the eardrum of *Phaenidae* moths which in turn are fed on by bats that use sonar. Since the moths use their hearing to avoid the bats, it is in the mites’ interest to eat one of the moths’ eardrums at a time. What protocol do they use to chose an eardrum? Each mite emits the same concentration of a pheremone. The mites move toward the eardrum with higher pheremone concentration. Thus any initial asymmetry in the mite distribution is quickly magnified until all the mites are on one eardrum. In the computer science literature, this is referred to as the Choice Coordination problem. We can take these biological examples as existence proofs of distributed coordination schemes that make use of natural symmetry breaking.

Both of these examples share certain features with the problems we discuss: the initial part of the process involves symmetry breaking, and an increase in entropy, and the final part of the process involves a decrease in entropy, and results in some coordinated behavior (correlated in the case of the mites, and anticorrelated in the case of the fish). In the mite example, the symmetry breaking consists of deciding which of two identical ears to occupy first; the coordination portion occurs when all the mites move to this ear.

## 2 The problem: Symmetry breaking and ID assignment

Like the examples mentioned above, the self-organizing ID assignment problem has two logically distinct subparts. The first part is to make all the units different in some way (to break symmetry), and the second part is to assign the unique IDs to the units, assuming that the symmetry has already been broken. (The two parts may not actually occur as two sequential phases, but they are logically separate.) In the first part, the entropy of the units' memory increases; in the second part it decreases. Self organizing processes are characterized by (local) lowering of entropy; but for this to occur in the context of our deterministic digital devices, a source of entropy, the thermal noise source, must be built in. The electric fish and the mites, by contrast, are “analog,” and thus have ready access to fluctuations.

### 2.1 Related work on symmetry breaking problems

There are several problems that have been studied by computer scientists that have a symmetry breaking component to them. In all these problems, if symmetry is not broken in advance by pre-assigning a unique ID to each processor, then it must be broken in the course of the algorithm by use of independent sources of random bits.

The first well known symmetry breaking problem is the Dining Philosophers problem, posed in 1965 by Dijkstra[2]. In this problem, processes with overlapping resource requirements must be scheduled. The symmetry breaking problem arises when two processors try simultaneously to access the same resource: to break or avoid deadlock, one must yield. Dijkstra proposed a randomized algorithm in which each processor waits for a random interval before trying to use the resource again.

Around 1971 a similar randomized protocol was used to avoid deadlocks after collisions in Alohanet, an early packet radio communication network, and a substantially improved, adaptive version of this scheme was developed in 1973 for Ethernet. [8]

In a 1982 paper, Rabin explored randomized algorithms for the Choice Coordination Problem, in which  $n$  processors must agree on a common choice of one out of  $k$  outcomes.[6] Awerbuch *et al.* present randomized algorithms for the maximal independent set and Dining Philosophers problem for an asynchronous distributed network of processors with arbitrary network topology.[1] Almost all the computer science work on symmetry breaking problems differs from ours in that it considers specific computational problems that have a symmetry breaking component. In this paper, we propose isolating the symmetry breaking problem, and solving it once and for all using an algorithm whose sole function is to assign unique identities. These identities can then be used to break symmetry if desired, and for additional non-computational purposes such as communication.

The one example in the computer science literature of using a symmetry breaking protocol to assign unique identities is a paper of Lipton and Park[4], in which they pose what they call the Processor Identity Problem—fundamentally

the same problem discussed in this paper, but set in the context of a shared memory multiprocessing system. The protocol they describe is of no use in the multiaccess-channel scenario we are considering. Like most of the other work relating to symmetry breaking in computer science, the interactions between the processors (ie the communication model) is very different than ours.

## 2.2 Preliminaries

In this example, we will assume there are  $n$  units with  $T$  bits of storage each. Thus each unit may be in one of  $m = 2^T$  states. The three schemes we will present each assume a different model of communication. The non-interacting algorithm doesn't require any communication infrastructure to assign the IDs. The sequential algorithm assumes a "wired-or" bus, and the parallel algorithm uses what I will call an Amplitude Modulation (AM) bus. On a wired-or bus, if any unit writes a one, all see a one. On the AM bus, all the units operating at a particular frequency can write a modulated carrier (modulated by zero or one, say), and can determine the amplitude of the sum of the (up to)  $n$  signals that were written on the bus at that frequency. So, the AM bus allows many simultaneous channels (different frequencies), and in each channel a sum of the input amplitudes is performed. Thus all the units in a communicating subgroup (cell) can learn the sum of the values they wrote onto the bus, typically the sum of their random bits.

In analyzing the proposed schemes, we will consider the scaling of three quantities as a function of  $n$  and  $p_f$ : the amount of storage, time needed to assign IDs, and the number of random bits used. We will also be mindful of the amount of communication used in each scheme, though it does not appear explicitly in any formulae.

## 2.3 Non-interacting solution and the "birthday" problem

Perhaps the most obvious solution is only to have a symmetry breaking phase: the random bits chosen by a unit in the symmetry breaking phase are its ID. This seems appealing intuitively, since the probability of a particular unit choosing a particular ID is very small,  $2^{-T}$ . However, the probability of that some collision will occur is much higher. The famous birthday paradox asserts that given a set of more than 22 people (with randomly distributed birthdays), the probability that two or more of them will share a birthday is greater than one half. In particular, a simple combinatorial argument shows that the probability of failure  $p_f$  of the non-interacting protocol is

$$p_f = 1 - \frac{m!}{(m-n)!m^n} = 1 - \frac{2^T!}{(2^T-n)!2^{Tn}}$$

There is a well known formula that bounds this probability; see for example Motwani [5] for a derivation.

$$p_f \leq 1 - e^{-n(n-1)/2m} = 1 - e^{-n(n-1)/2^{T+1}}$$

Solving this inequality for  $T$ , we find

$$T \geq \log_2 \left( \frac{-n^2 + n}{\ln(1 - p_f)} \right) - 1$$

Assuming that we are interested in a small probability of failure, we can make the approximation  $\ln(1 - p_f) \approx -p_f$ , so  $T \geq \log_2 \left( \frac{n^2 - n}{p_f} \right) - 1$ . We now drop the additive constant, allowing ourselves one extra bit, and also drop the linear  $n$  term, which is negligible, to find the following useful expression for the number of bits needed in the symmetry breaking ID as a function of the number of units  $n$  and the desired probability of failure  $p_f$ :

$$T \geq \log_2 \frac{n^2}{p_f}$$

Since the algorithm has no other significant storage requirements, the space requirement is also given by this expression for  $T$ . For  $n = 10^6$  and  $p_f = 10^{-15}$ ,  $T = 90$ . Since, with high probability, the symmetry has been broken (each unit has a unique identifier), a compact set of working IDs in  $0, \dots, n - 1$  can now be assigned.

The working IDs can be assigned by a binary search procedure. Assume for the moment that one unit can become “bus master” (below we will present a scheme by which this could happen). The bus master can broadcast a message asking whether there are any units with ID less than  $\frac{1}{2}2^T$  and wait for a response. If there is a response, it cuts its pivot point in half and asks again. We do not need to assume that the bus master can detect communication collisions (multiple responses to its queries). We have chosen  $T$  large enough that the IDs are all distinct. Instead of stopping its search as soon as it has found a range of IDs containing just a single unit, the bus master can continue its binary search until it has discovered the precise “symmetry breaking ID” of each unit. Thus it can find all the occupied IDs.

Finding a single occupied ID takes time  $\log_2 m = \log_2 2^T = T$ , and since this must occur  $n$  times, the total time for this procedure is  $nT$ . Each unit can then be assigned the working ID in the range  $0$  to  $n - 1$ . The time needed to break symmetry is  $T$  and time  $nT$  is needed to assign the short IDs. Thus the total time needed is  $L = (n + 1)T$ . Using the expression for  $T$  derived above, we can write the total time

$$L = (n + 1) \log_2 \frac{n^2}{p_f}$$

The total number of random bits needed is  $R = nT$ ; this can be written

$$R = n \log_2 \frac{n^2}{p_f}$$

We will see in the next section that allowing some communication decreases the ID size requirements.

## 2.4 Sequential symmetry breaking and ID assignment

For this scheme, the communication model is assumed to be a wired-or bus, a term that was defined in section 2.2. The  $n$  units engage in a tournament consisting of a series of  $n$  rounds. Each round ends with one winner, who takes the next unallocated ID, and then doesn't participate in later rounds. Thus the first tournament has  $n$  participants, and the last has just 1. In the terminology of distributed algorithms, this could be described as repeated leader election.

In each round, all the participating units chose a random bit, write it on to the shared bus, and then check the value of the shared bus (I will call this a trial). Because of the wired-or bus, if any unit writes a 1, all will read a 1. So if a unit writes 0 and reads 1, it knows that at least one other unit is sharing the bus and drops out of the round. Eventually, only one unit will remain; when it has read a long enough sequence of bits that are identical to what it wrote, it can be satisfied that it is alone, so it takes the next ID and broadcasts a message for the others to begin the next tournament.

The total space required by the algorithm is simply the number of bits  $T$  needed to represent  $n$  IDs:

$$T = \log n$$

A rough analysis of the running time is as follows. There are two parts to the running time:  $L = L_1 + L_2$ . The first,  $L_1$ , is the expected time to produce a winner; it depends only on the number of units  $n$ . The second,  $L_2$ , is the time it takes the winner to realize that it has won. It depends only on the desired probability of failure  $p_f$ . First we will estimate  $L_1$ . Round  $i$ , which has  $i$  participating units, consists of approximately  $\log_2 i$  trials, since roughly half drop out of the round after each trial. Thus the total number of trials (total time) is approximated by

$$L_1 = \sum_{i=1}^n \log_2 i = \log_2 \prod_{i=1}^n i = \log_2 n! = \frac{\ln n!}{\ln 2} \leq \frac{n \ln n}{\ln 2}$$

To find  $L_2$ , we will evaluate our our “birthday failure” expression from the previous section with  $n = 2$  to find the probability of 2 or more units choosing the same sequence of  $L_2$  random bits. The result is  $L_2 = \log_2 \frac{1}{p_f}$ . Thus

$$L = L_1 + L_2 \leq \frac{n \ln n}{\ln 2} + \log_2 \frac{1}{p_f}$$

The total number of random bits needed is

$$R = \sum_{i=1}^n i \log_2 i$$

For  $n$  sufficiently large, this sum can be approximated by an integral:

$$R \approx \int_{i=1}^n i \log_2 i di = \frac{1}{4} (n^2 (\log_2 n - 1) + 1) \leq n^2 \log_2 2n$$

Allowing some communication—the wired OR bus—decreased the ID size needed. The next protocol significantly decreases the time needed to assign the IDs, though its bandwidth requirements are much higher.

## 2.5 Parallel symmetry breaking and ID assignment

The modest communications needed for the previous scheme allowed the shortest possible IDs to be used. The scheme that will be presented in this section brings to bear more communication resources (bandwidth), and is thereby able to assign a set of short IDs much more quickly.

The idea is to split the  $n$  units into two equal groups after they have chosen  $T$  random bits, and then repeat the process recursively. Each of these iterations assigns one ID bit to each unit. If additional communication bandwidth is available, then the units can divide into smaller and smaller communication cells: each unit's current ID specifies the frequency at which it should communicate. Initially, all the units have the same ID (0), so all share the same communication channel. The additional communications bandwidth allows the IDs to be assigned simultaneously, in parallel.

In this scheme, each unit first chooses a random bit, which splits the group roughly in half; then the deviation from a perfect split is measured using the AM bus, and then further, corrective flips occur, with probabilities chosen so as to reduce the deviation. If a proposed random step ever increases the deviation, it is rejected, so the deviation converges monotonically to zero. For the corrective flip at time  $t$ , we chose the flipping probability  $p_t$  such that the expected value of the sum after the flip is  $\frac{n}{2}$ . We will denote the sum of the random variables  $\xi$ , and the expected value of the sum  $\mu = \frac{n}{2}$ . It will be convenient to define the "excess"  $s = \xi - \mu$ . Without loss of generality, we can always take  $s$  to be positive;  $s$  is negative indicates an excess of 0s, and in this case we can deterministically flip all the units, so that  $s' = |s|$ .

The correction step at time  $t$  is a biased random walk starting from  $\xi_{t-1}$  and moving in the direction of  $\mu$ . The probability  $p_t(\xi_t|\xi_{t-1})$  is a Gaussian with mean  $\xi_{t-1} - p_t\xi_{t-1}$ , because the initial value is  $\xi_{t-1}$ , and  $\xi_{t-1}$  is also the number of units eligible to flip from 1 to 0. At every time step we must choose  $p_t$  so that the mean of  $p_t(\xi_t|\xi_{t-1})$  is  $\mu = \frac{n}{2}$ .

Thus we solve  $\xi_{t-1} - p_t\xi_{t-1} = \frac{n}{2}$  for  $p_t$ , which gives  $p_t = 1 - \frac{n}{2\xi_{t-1}}$ . The mean of the distribution describing the new value of a single unit that is eligible to flip is given by  $\mu_1 = (0)p_t + (1)(1 - p_t) = 1 - p_t$ . The variance of this distribution is given by  $\sigma_1^2 = (0 - \mu_1)^2 p_t + (1 - \mu_1)^2 (1 - p_t)$ , which simplifies to  $\sigma_1^2 = p_t(1 - p_t)$ . Thus if the sum before the correction flip is  $\xi_t$ , so that  $\xi_t$  units are eligible to participate in this flip, the variance of the  $p_t(\xi_t|\xi_{t-1})$  distribution is  $\sigma_t^2 = p_t(1 - p_t)\xi_{t-1}$ .

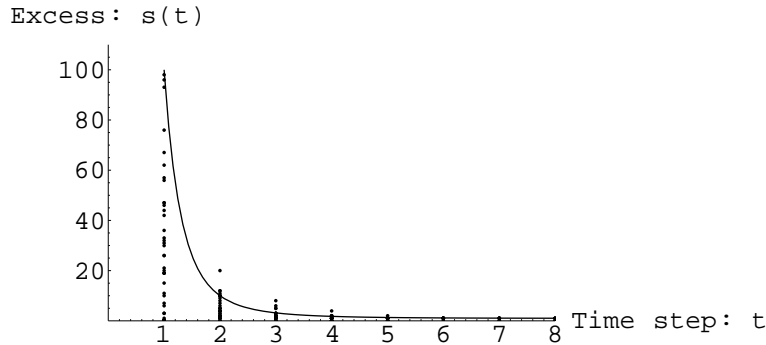
We can study the algorithm's rate of convergence by considering the deterministic dynamics of a "typical value" of the sum  $\xi_t$ . For this typical value, we will use  $\xi_t = \mu + \sigma_t = \mu + (p_t(1 - p_t)\xi_{t-1})^{\frac{1}{2}}$ . We have an explicit expression for  $p_t$  in terms of  $\xi_{t-1}$ , which gives us an explicit difference equation for  $\xi_t$  in terms of  $\xi_{t-1}$  and  $n$ :

$$\xi_t = \frac{n}{2} + \left( \frac{n}{2} - \frac{n^2}{4\xi_{t-1}} \right)^{\frac{1}{2}} \quad (1)$$

This difference equation is non-linear, however, so it cannot be solved for  $\xi$  as a

function of  $t$ . We could model the algorithm's dynamics by brute-force iteration of the equation, but this would not give insight into the scaling properties.

However, it may be verified that  $\xi_t < \mu + \xi_0^{\frac{1}{2^t}}$ , or equivalently,  $s_t < n^{\frac{1}{2^t}}$ . We can use this as an approximate (upper bound) solution of the non-linear difference equation. The approximate solution does give some insight into the algorithm's scaling properties. Figure 1 shows the second expression plotted alongside data from 35 simulated runs of the algorithm with  $n = 10^4$ : the  $s_t$  values are plotted against  $t$ , and the data from all these runs is overlaid. The simulations were performed using Mathematica's default (real) pseudo-random number generator. In each iteration of the algorithm, the units with value 1 flip to 0 with probability  $p_t$ , where  $p_t = 1 - \frac{n}{2\xi_{t-1}}$ , as specified earlier. (Our simulation assumes that the units have access to a real valued random number generator. A more detailed simulation could model the process units would employ if they only had access to binary random number generators.)



**Fig. 1.** Measured and theoretical excess  $s$  as a function of  $t$  for  $n = 10^4$  units. The scattered points represent 35 runs of the algorithm, and the solid line is the analytical model.

Both the non-linear difference equation for  $s$  and the upper bound approximation for  $s_t$  converge to 1 as  $t$  grows, though in the actual algorithm this value eventually converges to zero. The algorithm's dynamics enters a completely new regime when  $s = 1$ ; presumably there is a continuous rather than discrete change to this new regime, and our analysis would ideally model that. We will assume there are two distinct epochs in the algorithm dynamics, one in which  $s > 1$  (the period analyzed above), and  $s = 1$ , the endgame.

We will now find the convergence time of the first phase of the algorithm. We will consider the time required for  $s$  to converge to 2, since our analytical expressions can never actually reach 1. Solving  $s_t = n^{\frac{1}{2^t}} = 2$  for  $t$  yields  $t = \log_2 \log_2 n$ .

Thus  $L_1$ , the time required for the first phase of the algorithm, is given by

$$L_1 = \log_2 \log_2 n$$

**Endgame** Once the excess has been reduced to one, the algorithm's dynamics enters a new regime, since  $s_t$  cannot become any smaller than one without the algorithm terminating. Thus once the algorithm has reached this "endgame," equation 1 is no longer a good model (and the upper bound formula is even worse), so we must analyze it differently. The probability of exactly one of the  $\xi = \frac{n}{2} + 1$  high units flipping in one time step (ie  $\frac{n}{2} + 1$  trials) is given by a Poisson distribution:  $w = \frac{\lambda^1}{1!} e^{-\lambda}$ , where  $\lambda = (\frac{n}{2} + 1)p$ , with  $p = \frac{s}{n/2+s} = \frac{1}{n/2+1} = \frac{2}{n+2}$ . Writing out the expression for  $\lambda$ , one finds that it simplifies to one. Thus the probability of exactly one unit flipping in a single parallel trial is  $\frac{1}{e}$ .

In each iteration of the algorithm, either one unit flips, or some other number flips, so each iterate is a Bernouli trial with  $p = \frac{1}{e}$ . (If more than one unit flips, the units all revert to their previous state; thus there really are only two cases to worry about.) The probability of a certain number of trials  $L_2$  occuring before success is given by a geometric distribution  $p(L_2) = p(1-p)^{L_2-1}$ . The probability of winning at any time  $l$  up to and including  $L_2$  is therefore given by

$$p_s = \sum_{l=1}^{L_2} \frac{1}{e} \left(1 - \frac{1}{e}\right)^{l-1} = 1 - \left(\frac{e-1}{e}\right)^{L_2}$$

We have summed the geometric series to find the final expression. The probability of the algorithm failing to terminate prior to time  $L_2$  is  $p_f = 1 - p_s$ . Solving for  $L_2$ , we find

$$L_2 = \frac{\ln p_f}{\ln \frac{e-1}{e}} = -2.18 \ln p_f$$

In order to make  $p_f$  less than  $10^{-15}$ , we must chose  $L_2 = 75$ . The total running time  $L$  is given by

$$L = L_1 + L_2 = \log_2 \log_2 n + 2.18 \ln \frac{1}{p_f}$$

Under this algorithm, each unit needs  $\log n$  bits of storage to hold its own ID,  $\log n$  more to represent the random variable that it uses in deciding whether to take part in the corrective flip, and another  $\log n$  to represent the  $\xi_t$  value that it reads from the AM bus. Thus the storage  $T$  needed for each unit is

$$T = \log n^3$$

Finally, the total number of random bits  $R$  used is the product of the number of trials  $L$  times  $\log n^2$  (the number of storage bits per unit devoted to random variables; each of these gets randomized once per trial) times  $n$ , the number of units.  $R = (\log_2 \log_2 n + 2.18 \ln \frac{1}{p_f})(\log n^2)n$  which can be rewritten

$$R = (n \log n^2)(\log_2 \log_2 n + 2.18 \ln \frac{1}{p_f})$$

This algorithm assigns a set of short IDs much more quickly than the sequential algorithm. However, its bandwidth requirement is correspondingly higher.

### 3 Conclusion

The tables below summarize and compare the three algorithms. A point not illustrated by the tables is the fact that the three algorithms use different amounts of communication: in the first, symmetry breaking phase of the non-interacting algorithm, the units don't do any distributed communication at all (the second phase of this algorithm, which does involve communication with the elected leader, may be viewed as optional). In the sequential algorithm, the units learn the OR of the other units' random bits. In the parallel algorithm, units communicate simultaneously using multiple frequencies, and the units sharing a frequency learn the sum of the other units' random bits. Clearly the third algorithm requires a channel with more bandwidth (for the multiple frequencies) and better signal-to-noise ratio (to make accurate measurements of the sums). These quantities, bandwidth and signal-to-noise, define the theoretical capacity of the channel, so it is clear that a higher capacity channel is required for the third algorithm. It would not be difficult to estimate the number of bits actually exchanged in the course of the various protocols.

What is notable in examining the tables is that the algorithms that use more communication require less time. The parallel algorithm in particular requires far less running time than the others, and requires a much fatter communication pipe. One can imagine a framework analogous to the Shannon's noisy coding theorem for understanding the fundamental limits of symmetry breaking protocols: it would relate the number of units, the number of random bits required, the time to convergence, the channel capacity of the bus, and probability of failure.

The first step would be to calculate a lower bound on the number of bits that must be exchanged in order for each unit to be (reasonably) certain that no one else is sharing its ID. Next, one might attempt to budget the entropy flow for each of the algorithms: entropy enters the system at the random number generator, then, through the communication mechanism, each unit learns some amount of information about the random variables in the other units. Finally, there is a stage in which unnecessary entropy is rejected and IDs are assigned. It would be interesting to know how closely the algorithms presented here approach these hypothetical lower bounds.

Another interesting question is how efficiently each protocol uses its bus. For example, using the bandwidth and SNR requirements of the parallel algorithm, one could calculate a required (peak) channel capacity, in bits per second. By calculating the total number of bits exchanged and the total running time, one could find the average communication rate, again in bits per second. Does the algorithm use the bus efficiently, utilizing its full capacity most of the time? Or, despite the need for high peak capacity, is the bus idle for much of the algorithm?

We have proposed and analyzed three distributed protocols for using a source of physical symmetry breaking to assign unique identities to a group of  $n$  pro-

processors. The communication requirements, and associated hardware complexity, of the third, parallel protocol are probably too demanding for it to be practical, despite its speed. The second, sequential protocol is faster than the non-interacting protocol, uses less storage, and does not have burdensome hardware requirements. Thus in practice, the sequential protocol is likely to be most useful. There is probably an even more practical, hybrid protocol that combines the minimal hardware requirements of the sequential protocol with some of the speed advantages of the parallel protocol. Its running time would be intermediate between the two, would require somewhat more storage, and would be difficult to study analytically. Such a hybrid protocol is likely to be most useful in practice.

As the number of networked processors explodes in the next few years, distributed protocols that allow the processors to automatically and dynamically manage their own ID assignment may become increasingly appealing. Our investigation of the ID assignment problem suggests that there is interesting theoretical structure lurking beneath this potentially very practical problem.

T (storage/unit)	
non-interacting	$\log_2 \frac{n^2}{p_f}$
sequential	$\log n$
parallel	$\log n^3$

L (time)	
non-interacting	$(n+1) \log_2 \frac{n^2}{p_f}$
sequential	$\frac{n \ln n}{\ln 2} + \log_2 \frac{1}{p_f}$
parallel	$\log_2 \log_2 n + 2.18 \ln \frac{1}{p_f}$

R (random bits)	
non-interacting	$n \log_2 \frac{n^2}{p_f}$
sequential	$n^2 \log 2n$
parallel	$(n \log n^2)(\log_2 \log_2 n + 2.18 \ln \frac{1}{p_f})$

T (storage/unit)	$n = 10^6$	$n = 320$
non-interacting	90	67
sequential	20	9
parallel	60	25

L (time)	$n = 10^6$	$n = 320$
non-interacting	$9.0 \times 10^7$	$2.1 \times 10^4$
sequential	$2.0 \times 10^7$	$2.7 \times 10^3$
parallel	80	78

R (random bits)	$n = 10^6$	$n = 320$
non-interacting	$9.0 \times 10^7$	$2.1 \times 10^4$
sequential	$1.4 \times 10^{13}$	$6.6 \times 10^5$
parallel	$3.2 \times 10^9$	$4.2 \times 10^5$

## Acknowledgments

The original inspiration for this paper came from the Rhythm Tree, a network of 320 drum pads that was part of the Brain Opera, an interactive, multimedia opera conceived and directed by Tod Machover at the Media Lab. Setting the drum pad IDs was a practical problem that had to be solved in the course of producing the opera.

Ara Knaian, who designed the Rhythm Tree hardware, made helpful algorithm suggestions, and included a noise circuit in the Rhythm Tree pad design.

This work was supported by MIT Media Lab's Things That Think consortium.

## References

1. B. Awerbuch, L. Cowen, and M. Smith. Efficient asynchronous distributed symmetry breaking. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, Montreal, Quebec, Canada, 1994.
2. E.W. Dijkstra. Solution of a problem in concurrent programming control. *Communications of the ACM*, 8(9):569, 1965.
3. W. Heiligenberg. *Studies of Brain Function, Vol. 1: Principles of Electrolocation and Jamming Avoidance*. Springer-Verlag, New York, 1977.
4. R.J. Lipton and A. Park. The processor identity problem. *Information Processing Letters*, 36:91–94, 1990.
5. R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, 1995.
6. M.O. Rabin. The choice coordination problem. *Acta Informatica*, 17:121–134, 1982.
7. Dallas Semiconductor. *Automatic Identification Data Book*. Dallas Semiconductor, Dallas, Texas, 1996.
8. A.S. Tanenbaum. *Computer Networks*. Prentice-Hall, Englewood Cliffs, New Jersey, 1989.