Randomized Wait-Free Naming

Alessandro Panconesi
Email: ale@cwi.nl
CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Marina Papatriantafilou
Email: ptrianta@cwi.nl
CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
& Computer Technology Institute, Patras, Greece
& Dept. of CE and Informatics, University of Patras, 26500 Patras, Greece

Philippas Tsigas
Email: tsigas@cwi.nl
CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
& Computer Technology Institute, Patras, Greece
& Dept. of CE and Informatics, University of Patras, 26500 Patras, Greece

Paul Vitányi
Email: paulv@cwi.nl
CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
& Faculteit Wiskunde en Informatica, Universiteit van Amsterdam, The Netherlands

Abstract

We present new distributed randomized naming protocols improving previous results in renaming and unique processor identity protocols. They are wait-free (which implies maximal fault-tolerance) and allow stronger adversaries. They also have low complexity. We give the first wait-free protocol achieving optimal key space range. (This is impossible for deterministic wait-free methods, so we use randomization.) We also introduce a novel wait-free object, a test-and-set object which upon invocation succeeds with probability less than 1, and we give a low complexity implementation of such objects.

AMS Subject Classification (1991): 68M10, 68Q22, 68Q25
CR Subject Classification (1991): B.3.2, D.4.1, D.4.5, G.3
Keywords & Phrases: Asynchrony, Distributed Computing, Fault-tolerance, Naming, Processor Identities, Randomized Algorithms, Renaming, Shared Memory Systems, TestAndSet Objects
Note: Partially supported by NWO through NFI Project ALADDIN under Contract number NF 62-376; the first author was supported by an ERCIM Fellowship; the second author was also partially supported by a NUFFIC Fellowship; the second and third author were also partially supported by the European Union through ALCOM ESPRIT Project Nr. 7141; the fourth author was also partially supported by the European Union through NeuroCOLT ESPRIT Working Group Nr. 8556. A preliminary version appeared in *Proc. Fifth Annual International Symposium on Algorithms and Computation*, Beijing, Lecture Notes in Computer Science, Volume 834, Springer-Verlag, Heidelberg, 1994, pp 83-91.

1. Introduction

Consider a set of n processes communicating through shared variables. We want these processes to execute a protocol such that when each process terminates it has obtained a *key* (or name or identity). We require different processes to obtain different keys, and we would like the range of the key space to be as small as possible (preferably $\{1, \ldots, n\}$). In our setting the processes are asynchronous with possible halting failures. (For synchronous processes without failures simple ranking suffices.) We call a protocol that satisfies the above a *naming* protocol.

We present new distributed randomized naming protocols improving previous results in renaming and unique processor identity protocols. They are wait-free (which implies maximal fault-tolerance) and allow stronger adversaries. They also have low complexity. We give the first wait-free protocol achieving optimal key space range. (This is impossible for deterministic wait-free methods, so we use randomization.) We achieve optimal key-space range which is impossible for deterministic wait-free methods. The discussion of the precise results is deferred to the next section where they are compared with related work.

The interest in constructing such protocols stems from the fact that in a real concurrent system or computer network the existence of distinct keys is useful or mandatory to resolve problems with exclusive access, resource allocation, network operation, leader election or choice coordination. Since processes may crash and new processes may be created, keys of processes may be duplicated and the key range may expand. Some network operation protocols crash on duplicate keys and perform more efficiently for small key ranges [16, 11]. A naming protocol rebalances the key space.

In one instance of the naming protocol the processes initially have distinct keys from an arbitrary key range. This is known as a *renaming* protocol. Theoretical interest is partly due to the fact that asynchronous processes have to agree on irrevocable decisions in the presence of faults. The existence of an affirmative deterministic solution ([3]) came as some surprise since under the same circumstances the problem of non-trivial consensus cannot be solved ([8], [9], [14]).

In fault-tolerant allocation of identical resources one can view a key as a permit for a resource. Managing their circulation among competing processes can be viewed as a repetitive variant of the renaming problem ([3]).

1.1 Related Work

We consider naming in the *standard model of asynchronous shared memory systems* ([13]). The processes communicate via a set of single-writer, multi-reader atomic registers. Each process owns a subset of these registers. Only the owner of a register can write it (by performing a write operation) while all the other processes can read it (by performing a read operation). Each process in one atomic step either writes or reads one of the registers or flips a local coin not visible to the other processes. A desirable property for protocols in such a system is *wait-freedom*. For our problem this means that each process terminates its protocol after a finite number of steps regardless of other processes' halting failures ($n-1$ *resiliency*) or relative speeds (no fast process will slow down to the speed of a slow one).

We model asynchrony by a scheduling demon which initially indicates which process makes the first atomic step, and after the execution of each atomic step indicates what process does the next atomic step or randomized choice. A resulting sequence of atomic steps is called a *history*. A protocol is correct if each history of the protocol is correct. The scheduling demon is often viewed as an *adversary* which plays against the processes. The adversary can be either (i) *oblivious* if it defines the order in which processes are scheduled oblivious to the system states in the execution, or (ii) *adaptive* if at each step it selects a process to take an action based on the state of the system. An adaptive adversary can be either *weak* or *strong*, depending on whether in a system state it has knowledge only of the shared memory state or also of the internal states of the processes.

The first solutions to our problem were for deterministic (not randomized) protocols, starting from a system state in which each processor has already a unique key but from an arbitrarily large (and unknown) key range. For n processes using asynchronous message passing (which we have not discussed) Attiya et. al. in [3] obtained a time complexity of the solution exponential in n and a range of the keys of $n + t$ in the presence of at most $t < n/2$ stopping failures. (The time complexity measures the worst case number of steps that a process needs to take until it reaches a decision.) Later, in [5] Bar-Noy and Dolev transform the solutions of [3] into two wait-free solutions for the shared memory model, which achieve execution times $(n + 1)4^n$ and $n^2 + 1$ and key space ranges of $2n - 1$ and $(n^2 + n)/2$ respectively. In [6] a wait-free solution is presented with $O(n^3)$ time complexity and $2n - 1$ key range. (These wait-free protocols make use of multi-valued atomic wait-free registers as primitives of the constructions.)

In [10] it is shown that deterministically one cannot do better than a key range of $2n - 1$ if up to $n - 1$ faults have to be tolerated. This raises the question of what randomization can do.

In what is known as the *Processor Identity Problem*, a set of initially identical processes have to find unique names (symmetry breaking). Here deterministic solutions are impossible. The best known solutions, as well as some impossibility results, can be found in the elegant [12]. They use a model of computation which is totally symmetric: if shared memory is used then all the processes should have write access to a common pool of shared registers instead of some processors owning a strict subset of those registers. In [12] it is shown that the problem cannot be solved unless n is known to all the processes, and that there is no finite-state solution if the adversary is adaptive. Moreover, an unbounded memory solution is presented and analysed considering a weak adaptive adversary, while two bounded solutions are given for the case when an oblivious adversary is assumed. The expected time complexity of the solutions is $O(\log n)$ and the key range is n. However, the solutions are not wait-free.

1.2 New Results

Our solutions are randomized and implemented in terms of *wait-free* test-and-set primitives. (Rather, test-and-set-once primitives since in the execution of the naming protocol we test-and-set each such object only once, and never reset.) The time complexity figures (number of accesses to constituent variables) hold with high probability with respect to an adaptive adversary.

In [4] a randomized wait-free construction for test-and-set is implemented in multireader

(1-writer) shared variables. Each such test-and-set object uses $O(n^2)$ multireader (1-writer) shared variables and an access by a process takes $O(n)$ accesses to the constituent shared variables.

We introduce and implement a novel primitive: a wait-free *weak* test-and-set object. In contrast to the construction in [4] which may be called a *strong* test-and-set, a weak test-and-set once invoked yields a winner with success probability α $(0 < \alpha < 1)$ and does not yield a winner (every contending process obtains value 0: the test-and-set fails) with probability $1-\alpha$. We can set α arbitrarily close to 1. The weak test-and-set object we construct uses $O(n \log^2 n)$ multireader (1-writer) single bit variables (or $O(n)$ multireader (1-writer) $2 \log \log n$ bit variables or 1 snapshot object [2, 1]). An access of the weak test-and-set by a process takes $O(n \log^2 n)$ accesses to the constituent multireader variables or $O(\log^2 n)$ accesses to the snapshot object. It is convenient to speak about an *α-test-and-set* object where we mean a weak test-and-set object for $\alpha < 1$ and strong one for $\alpha = 1$.

The SQUEEZE protocol uses $O(\log^2 n)$ accesses to α-test-and-set objects and r a parameter of the protocol and achieves a key range of n/α (= number of α-test-end-set objects used)[1]. For $\alpha = 1$ the protocol achieves a key range of exactly n at the cost of using $O(n \log^2 n)$ accesses to multireader (1-writer) variables, and it uses $O(n^3)$ multireader (1-writer) variables altogether.

The SEGMENT protocol uses $O(\log n)$ accesses to α-test-and-set objects and achieves a key range of $(1+c)n$ satisfying $\alpha \geq (1+2\epsilon)/(1+c)$ where $\epsilon > 0$ can be chosen fixed but arbitrarily small (it reflects the probability of staying within the time bound). (The number of α-test-end-set objects used is equal to the number of keys.)) [2]

Note that these results do not contradict the impossibility result of [12]. In our model the processes communicate through (probabilistic) test-and-set objects which are common to all processes. Such objects are stronger primitives than just shared variables. Viewed as black boxes they are symmetric with respect to all processors. Looking at their randomized implementation in terms of shared variables, such variables are nonsymmetric in that they are owned by different processors. It is worth pointing out that our protocols always terminate, and that the result is always correct (the new identities are distinct).

1.3 Test-and-Set

A *test-and-set* is a concurrent data object $TS(\cdot)$ shared between n processes $1, \ldots, n$. We only need a restricted version, where each such object only executes 1 test-and-set in its life time. The way we use it, $TS(i)$ is a function procedure which can be called with input

[1] Plugging in the above figures means that for $\alpha < 1$ the protocol uses $O(n \log^4 n)$ accesses to multireader (1-writer) variables and either $O(n^2 \log^2 n)$ multireader (1-writer) 1-bit variables or $O(n^2)$ multireader (1-writer) $2 \log \log n$ bit variables, or $O(\log^4 n)$ accesses to snapshot objects and $O(n)$ snapshot objects with n fields each, and n/α key range.

[2] We interpret the complexity in terms of multireader variables (from which the randomized test-and-set objects are constructed.) For $\alpha = 1$ we do not achieve a key range of exactly n but we do use $O(n \log n)$ accesses to multireader (1-writer) variables and $O(n^3)$ multireader (1-writer) variables. In the case of $\alpha < 1$ the method achieves a $(1+\delta)n$ key range ($\delta > 0$ but can be made very small by choosing α close to 1) and uses only $O(n^2 \log^2 n)$ multireader (1-writer) 1-bit variables or $O(n)$ multireader (1-writer) $2 \log \log n$ bit variables and $O(n \log^3 n)$ accesses to multireader variables, or $O(n)$ snapshot objects with n fields and $O(\log^3 n)$ accesses to snapshot objects.

parameter $i \neq 0$ and returns a value 0 or i. If the value is 0 then the call is said to be a *failure*, if the returned value is i then the call is said to be *successful*. Each of the n processes has a local variable key. For convenience, let us denote the variable key of the jth process by key_j. Initially, all key_j 's equal 0 ($1 \leq j \leq n$). There are also shared variables X_i ($1 \leq i \leq m$). Initially, the value of $X_i = i$. At any time exactly one of $X_i, key_1, \ldots, key_n$ has value i, all others have value $\neq i$. If a process j with $key_j = 0$ atomically executes a test-and-set operation then the effect is

$$TS(i) = \text{read } key_j := X_i; \text{ write } X_i := 0; \text{ return } key_j.$$

Note that for our purpose we do not need to reset the test-and-set object.

1.4 Randomization, Adversaries and Wait-Freedom

We allow the strongest adaptive adversary to schedule the distributed processes. It can do everything except predict the outcome of future coin tosses by the processes. Under this adversarial scheduling, our protocols below have low running time with high probability. Typically we show that a protocol has running time $O(\log^2 n)$ with probability at least $1 - e^{-n^\delta}$, $\delta > 0$, and at most $n + O(\log^2 n)$ otherwise. Hence, the expected running time is at most

$$e^{-n^\delta} n + (1 - e^{-n^\delta}) \log^2 n \leq \log^2 n + o(1).$$

We will use an estimate of the size of the tails of the binomial distribution. For s successes out of n independent trials with probability p of success, the classic Chernoff bounds give the following estimate (each tail separately can be bounded by half of the right-hand side),

$$P(|s - np| \geq \epsilon n) < 2e^{-\epsilon^2 n/4p(1-p)}. \tag{1.1}$$

2. A Probabilistic Test-and-Set Object

The object is *probabilistic* in the sense that a test-and-set succeeds with probability at least α with $0 < \alpha < 1$, for $\alpha = 2s/(1+s)$ where s ($0 < s < 1$) is a parameter of the protocol against the strongest adaptive adversary.

We explain the working of the new object with a balls and bins scenario. At the start there is a row, also called a *level*, of n balls. Each ball, independently of any other ball, flips a coin and with probability s *steps forward*, moving to the next level, or *backs-off* (it stops for ever) with probability $1 - s$. The value of s is a parameter set by the algorithm. At the next level the remaining balls, the *survivors*, flip their coins again, stepping forward with probability s or backing-off with probability $1 - s$, and so on. At each level ℓ there are 3 possible (mutually exclusive) outcomes: *i)* exactly one ball b steps forward and all others back-off, in this case b is declared the *winner* and the process ends; *ii)* every ball backs-off, in this case the process ends with no winner; *iii)* more than one ball steps forward, in which case the process continues (until one of the two aforementioned events occurs). This idea is implemented in the protocol of Figure 1 and the probabilistic behavior is treated in the Appendix.

procedure TS(i):

integer constant $m, k \in \{1..2n\}$; {Declarations}

rational constant $s \in \{l2^{-k} : 0 < l < 2^{-k}\}$

integer variable $i, q, level \in \{0..m\}$;

invoking processor $p \in \{1..n\}$;

begin

$TS() := 0$; {Initialize}

for $q \in \{1..n\}$ **do** read $(q, 1)$ **od**; {Check contention at gate}

if $(q, 1) = 0$ for all q **then** $level := 0$ **else goto** step L2;

L1 $level := level + 1$; write $(p, level) := 1$; {Raise level competition}

for $q \in \{1..n\}$ **do** read $(q, level)$ **od**; {Check contention at level}

L3 **if** $(q, level) = 0$ for all $q \neq p$ **then**

if $level = k$ **then** $TS() := i$; **return**(i) **else goto** step L1

else

if $level = k$ **then goto** step L2

else goto step (L1,L2) with probability $(s, 1 - s)$

L2 **for** $l := level, \ldots, 1$ **do** write $(p, l) := 0$ **od**; **return**(0); {Backoff}

end

Figure 1: Protocol Probabilistic Test-and-Set

The object is implemented as follows. There are n processes which communicate through atomic binary variables each of which is denoted as $(processor, level)$. The first coordinate indicates a process identity (w.l.o.g. a number in $\{1..n\}$), and the second coordinate indicates a confidence level which is an integer in $\{1..k\}$. The variables constitute an array $(1..n, 1..k)$, each element of which is a multireader bit. Each process p can write all shared variables $(p, \cdot)$ and can read all shared variables. Each process is given the number k of levels, and a randomized subroutine to make a binary choice with success probability s and failure probability $1 - s$ $(0 < s < 1)$. The local variables are $level, q, l$ and are integer valued. The input parameter s is a probability of the form $l2^{-j}$ $(1 \leq l \leq 2^j)$ for some fixed integer j and is generated by a subroutine flipping a fair coin j times in a row and checking whether the outcome is j zeros. The input parameter i is an integer *key*. The value $TS()$ equals 0 at the conclusion of a failed test-and-set, and it equals i at the conclusion of a success test-and-set. The protocol of process p is as described in Figure 1

2.1 Correctness

Safety.

For each Probabilistic test-and-set object, at most one process p ever executes $TS() := i$ with $i \neq 0$. After a process p has written (p, k) and before p has erased (p, k) no process

q which reads (p, k) in that interval executes $TS() := i$. Once an $TS() := i$ assignment is executed at level $level = k$ the bits (p, j) $(1 \leq j \leq k)$ keep value 1 forever. By atomicity of write/read actions, either write (p, k) by p precedes read (p, k) by q or vice versa. Moreover, read (q, k) by process p with outcome 0 must precede assignment $TS() := i$. Hence, if the latter assignment takes place then write (q, k) by process q follows after read (q, k) by process p, which in its turn happens after write (p, k). Since read (p, k) by process q happens after write (q, k), it follows that read (p, k) by process q happens after write $(p, k) := 1$ by process p. Therefore, process q in read (p, k) obtains value 1 and exits through step L2.

Liveness.
The algorithm is wait-free (contains no loops). Assume that every process which has no key yet is enabled, and that every process which is enabled eventually makes a step. Since every process always makes progress, every process starting its protocol either exits as winner ($TS() := i$) or through backing off in step L2.

Success Probability
We estimate a lower bound on the probability that a round ends successfully, that is by a process q executing $TS() := i$. We first analyze a simplified situation, and then use this to analyze the full protocol under the strongest possible adaptive adversary. Such an adversary knows everything of the current and past state of the system, but not the outcome of future random choices in the protocols.

Analysis of Simplified Situation: Recall the discussion of the balls and bins scenario interpretation with an indeterminate number of levels in Section 2. We want to estimate the probability that the game ends with a winner. Clearly, if there is a way for a ball to determine if it is a winner, this process can be used to assign a token to a unique ball among a set of competitors. Since there is a non-zero probability that the process ends with no winner, we can think of it as a probabilistic *test-and-set*. Below, we show that the probability of failure can be pushed arbitrarily close to 0, its precise value depending on the choice of s and on how long we are willing to simulate the process without aborting it.

Given an initial row of n balls we denote the probability of the game ending with a winner by $f(n)$. We define $f(1) = 1$ and $f(0) = 0$.

CLAIM 1 Let $s \neq 1$. Then $f(2) = 2s/(1+s)$.

PROOF. The equation $f(2) = s^2 f(2) + 2s(1-s)$ implies the claim. □

CLAIM 2 For all $n \geq 2$, $f(n) \geq f(2)$

PROOF. We prove the claim by induction. The base case, $f(2) \geq f(2)$, is trivial. For the inductive step, assume that $f(k) \geq f(2)$ for all $2 \leq k < n$. Let X be a random variable denoting the number of surviving balls after the first step. Then,

$$\begin{aligned} f(n) &= \sum_{k=0}^{n} f(k) P(X = k) \\ &= P(X = 1) + f(n) P(X = n) + \sum_{k=2}^{n-1} f(k) P(X = k) \end{aligned}$$

Recalling that $P(X=k)=\binom{n}{k}s^k(1-s)^{n-k}$, it follows from the induction hypothesis that

$$
\begin{aligned}
f(n) &\geq \left(P(X=1)+\sum_{k=2}^{n-1} f(k)P(X=k)\right)\bigg/ 1-P(X=n) \\
&\geq \left(P(X=1)+f(2)\sum_{k=2}^{n-1} P(X=k)\right)\bigg/ 1-P(X=n) \\
&= (P(X=1)+f(2)\,(1-P(X=0)-P(X=1)-P(X=n)))/\,(1-P(X=n)) \\
&\geq f(2)
\end{aligned}
$$

□

THEOREM 1 *Let $n \geq 2$ be the number of balls in the game. The probability that there is a winner within k levels is at least $f(2)-ns^k$.*

PROOF. Let W_k be the event that there is a winner within k levels. Then

$$
\begin{aligned}
P(W_k) &= P(\text{there is a winner}) - P(\text{there is a winner after } k \text{ levels}) \\
&\geq f(n) - P(\text{some ball makes at least } k \text{ levels}) \\
&\geq f(2) - ns^k.
\end{aligned}
$$

□

An important corollary of this theorem is that the probability of having a winner can be made arbitrarily close to 1 in finite time by appropriate choices of the parameters s and k. In particular, for each fixed but *arbitrarily small* s the value of k can be as small as $\log n$.

Analysis of the Protocol with Adaptive Adversary We now prove that even assuming an adaptive adversary, with high probability the game quickly ends with a winner. Considering the protocol of Figure 1, note that at each level at most *one* ball may advance to the next level with probability 1. The adversary can exploit this feature, but not too much.

THEOREM 2 *A probabilistic* test & set *with parameter s and n competitors succeeds in time $O(\log^2 n)$ with probability at least*

$$\alpha = (1-o(1))\frac{2s}{1+s}.$$

It is implemented in $O(n\log^2 n)$ multireader (1-writer) atomic 1-bit variables or $O(n)$ multireader (1-writer) atomic $2\log\log n$-bit variables and $O(n\log^2 n)$ accesses to the multireader (1-writer) variables. Alternatively, it can be implemented using 1 snapshot object with n fields and $O(\log^2 n)$ accesses per test-and-set operation.

PROOF. (Sketch.) The crucial observation is that, for each level ℓ, at most one ball steps ahead with probability 1; all other balls flip coins. Suppose we just observe the process evolving for a total of k levels. Call a level *red* if at that level there is one (unique) ball stepping forward with probability 1. Now, either there are $\sqrt{k}$ red levels or there is a stretch of $\sqrt{k}$ levels without red levels. In the latter case, the probability of having a winner is at least $f(2) - ns^{\sqrt{k}}$. In the former, notice that one ball stepped to level $\sqrt{k}+1$ with probability 1. So, we have at least one ball at that level. What is the probability that there is *another* ball? This probability is at most $ns^{\sqrt{k}}$ because for another ball to be there there must be a sequence of at least $\sqrt{k}$ successful coin flips (successful meaning a ball steps forward). So, the probability of having a winner in the former case is at least $1 - ns^{\sqrt{k}}$. By choosing k appropriately, say $k = c\log^2 n$, for some constant $c > 0$ only depending on s, the claim follows. □

Finally, we can replace each row of 1-bit multireader (1-writer) variables in protocol of Figure 3 by a single $2\log\log n$-bit multireader (1-writer) variable which is used as a counter which counts up to $k = c\log^2 n$, and modify the protocol in the obvious way. We can further simplify the construction by putting each such counter in a field of a single snapshot object as defined in [2, 1].

3. Randomized Solutions

Our protocols are based on a balls-and-bins scenario, a widely used paradigm in the analysis of probabilistic processes [7]. In our scenario the key-space is made of m bins, and each process is given a ball to be thrown uniformly at random into the bins, independently of the other processes. In what follows, let $m = (1+c)n$, with $0 \le c \le 1$. There are several ways in which this scenario can be used to assign keys.

For motivation, suppose we decide for the following policy: if a ball falls uniquely in a bin, that key is assigned to the throwing process, otherwise the ball is returned to the process, which tries again. Below we show that with high probability the number of repetitions will be exponentially large in n. So the naive strategy doesn't work, and we have to do better.

3.1 Naive Approach Doesn't Work

For motivation, suppose we decide for the following policy: if a ball falls uniquely in a bin, that key is assigned to the throwing process, otherwise the ball is returned to the process, which tries again.

This policy is attractive because it is simple to implement; we only need to determine if a ball is alone in a bin and if a bin key has already been assigned. But unfortunately, for $c < 1$, with high probability the number of repetitions will be exponentially large in n. So the naive strategy doesn't work, and we have to do better.

To see the problem consider the following strategy for the adversary. Without loss of generality, we can assume that whenever a bin contains two or more balls the adversary can *recycle* all of them, that is, take all of them out to be thrown again. (This follows from the fact that our type of adversary can stop a ball as soon as its destination bin is randomly selected and before it has actually entered the bin.) Let $\lambda = 1/(1+c)$ so that $n = \lambda m$, and fix some $\epsilon < \lambda^2$. Initially, all n balls are thrown. With high probability, there will be many bins with two or more balls (roughly an $1 - e^{-\lambda}(1-\lambda)$ fraction). The adversary then selects a fraction of these bins in order to have a total of ϵn recyclable balls. These balls are momentarily *frozen* (that is, stopped). The process continues without any intereference from the adversary until all the remaining, unfrozen balls end up alone into different bins. At this point, there are $(1-\epsilon)n = (1-\epsilon)\lambda m$ bins containing a unique ball. Call these the *red* bins. Now the ϵn frozen balls are unfrozen and thrown again. Each ball ends up into a red bin with probability $(1-\epsilon)\lambda$. Moreover, it ends up there *alone* (meaning no other formerly unfrozen ball enters the same bin) with probability roughly $(1-\epsilon)\lambda e^{-\epsilon}$.[3] Hence, the adversary has now (roughly) $2(1-\epsilon)\lambda e^{-\epsilon} n$ recyclable balls. If $2e^{-\epsilon}(1-\epsilon)\lambda > \epsilon$, a condition satisfied for small enough ϵ, the adversary can again freeze ϵn balls and the whole process can be repeated! Simple calculations using the Chernoff bounds show that, with high probability, the number of repetitions is exponentially large (in n).

In the next subsections, we introduce two different strategies such that every ball is thrown at most poly-logarthmically many times, with high probability.

3.2 Squeeze Protocol

Let $0 < \alpha \leq 1$ be the success probability (reliability) of our test-and-set's. We use a key space of $m = \lfloor n/\beta \rfloor$ keys $\{1, \ldots, m\}$, where $\beta = (1-\delta)\alpha \approx \alpha$ (δ is an arbitrarily small constant which intuitively takes care of fluctuations below the expected values). The protocol is as follows. Let r $(0 < r < 1)$ be a parameter to be fixed later and assume the integer rounding. Initially, every ball of $b_0 = n$ balls selects (is thrown into) a random bin of an initial segment of rm bins. Call this the first r-segment. By choosing r appropriately we can ensure that, with high probability, every bin in the first r-segment is accessed (every test-and-set in the r-segment is invoked). Therefore, after the first throw, with high probability, approximately $\alpha rm \approx rn$ processes have a key. Repeat the procedure with the remaining $b_1 \approx (1-r)n$ balls on the second r-segment which is determined as follows. Apart from the first r-segment there are $m_1 = (1-r)m$ remaining bins. The first rm_1 of them form the second r-segment. Again, with high probability $\alpha rm_1 \approx r(1-r)n$ bins will be taken, meaning that many additional processes obtain a key. Continuing this way, in the kth round each of the remaining $b_k \approx (1-r)^k n$ processes selects a bin uniformly at random from the k-th r-segment consisting of the first rm_k bins of the remaining $m_k = (1-r)^k m$ bins. This process continues as long as $rm_k > 1$. At that point each process without key tries the remaining $1/r$ bins sequentially.

The idea behind the protocol is that every bin will be hit (every test-and-set will be invoked), and hence an expected number of $\alpha m > n$ will be successful (capture a ball). The choice of $\beta = \alpha(1-\delta)$ makes sure that in spite of fluctuations below the expected value all n balls are captured.

In Figure 2 the protocol for n processes is given in pseudocode (use the $\lfloor \cdot \rfloor$ function to round off fractional quantities). The variables a and b denote the keys of the first and last bins of the current r-segment. In the (very) unlikely event that the probabilistic strategy fails (Phase 1), the protocol has a back-up routine consisting of scanning the whole key-space starting with the last unused part (Phase 2).

THEOREM 3 *Let $1/2 < \alpha \leq 1$, and $\beta = (1-\delta)\alpha$, with δ an arbitrary constant greater than*

[3] This can be seen either by direct computation or by using the Poisson approximation to the binomial distribution.

```
procedure SQUEEZE(β, r)
 real constant r, β ∈ (0,1];                                {Declarations}
 integer variable a, b, m, i, key ∈ {0..m};
 begin
     a := 1; m = ⌊n/β⌋; b := ⌊rm⌋; key := 0;                 {Initialize}
     while key = 0 & b ≤ m                              {Phase 1: squeeze}
         i := random ∈ {a..b};              {Choose random key in {a..b}}
         key := TS(i);                     {Execute test-and-set for key}
         a := b + 1; b := b + ⌊r(m − b)⌋;           {Select next r-segment}
      while key = 0                                {Phase 2: linear search}
         key := TS(a); a := (a + 1)  (mod m)
      return key
 end
```

Figure 2: SQUEEZE Protocol

0. Then, SQUEEZE *is a naming protocol for n processes to n/β keys, using n/β test-and-set objects with success probability α. With high probability, for* each *of the n processes,* SQUEEZE *requires at most $\Theta(\log^2 n)$ steps.*

PROOF. (SKETCH) We assume the notation and explanation above. Let $r \in (0,1)$ be a constant to be determined later. If n balls are thrown uniformly at random at $rm = rn/\beta$ bins, then the probability that a particular bin is *not* hit is $\leq (1 - 1/(rn/\beta))^n$ which can be estimated by $e^{-\beta/r}$. We consider a $(p, 1-p)$ Bernoulli process with $rm = rn/\beta$ trials with success probability

$$p \geq \alpha(1 - e^{-\beta/r}), \tag{3.2}$$

where α is the probability of a bin capturing a ball when it is actually hit. We want to show that at this stage at least rn bins are successful (capture a ball). Let $\epsilon = \alpha\delta$. By recalling Equation 1.1 and plugging in actual numbers we obtain (s denotes the actual number of successful bins)

$$P(s \leq rn) \leq P(|s - rmp| \geq \epsilon rm) < 2e^{-\epsilon^2 rm/(4p(1-p))}$$

which is exponentially small in n, even if we set r as small as $1/\log n$. In the next round the ratio of free balls per number of bins in the next r-segment is with high probability the same as in the first round yielding *mutatis mutandis* the same result, and so on for all the rounds.

Our choice of r ensures that the number of iterations of the protocol is at most $O(\log^2 n)$. To see this, set $k = 1/r^2$, then

```
procedure SEGMENT(m,key):
  integer constant m ∈ {n..2n};                              {Declarations}
  integer variable i, key ∈ {0..m};
  begin
      key := 0; i := random ∈ {1..m};          {Choose random key in {1..m}}
      while key = 0
          key := TS(i); i := (i + 1)  (mod m)                 {Linear search}
      return key
  end
```

Figure 3: SEGMENT Protocol

$$rm_k = rm(1-r)^{1/r^2} = rm(1-1/\log n)^{\log^2 n} \approx rme^{-\log n} < 1, \tag{3.3}$$

for sufficiently large n. (Recall 'log' denotes the binary logarithm.) $\square$

COROLLARY 1 *Using n strong test-and-set objects ($\alpha = 1$), by approximately the same proof with $\beta = \alpha$, with high probability, for all n processes SQUEEZE using only $m = n$ keys still finishes in $\Theta(\log^2 n)$ steps.*

3.3 Segment Protocol

THEOREM 4 SEGMENT *is a naming protocol which for n processes to $(1+c)n$ keys ($c > 0$) using $(1+c)n$ test-and-set objects with success probability α ($1/2 < \alpha \leq 1$) requires at most $\Theta(\log n)$ steps with high probability.*

PROOF. The SEGMENT protocol is as follows. Set $r = a \log n/m$, where $a > 1$. To do the estimate below, conceptually divide the key-space of $m = (1+c)n$ bins, with $c > 0$, into overlapping *segments* of size rm bins each. A segment is simply a collection of contiguous bins. Each ball picks a segment uniformly at random independently of other processes. The expected number of balls falling into any particular segment is hence rn. A straightforward application of the Chernoff-bounds of Equation 1.1 shows that large deviations from this value are extremely unlikely. More precisely, for each fixed $\epsilon > 0$ the probability of having more than $(1+\epsilon)rn$ balls in *any* segment is exponentially small.

Namely, fix a segment S and consider a $(r, 1-r)$ Bernoulli process with success probability r for a ball selecting S. Then, the number of successes s in n trials satisfies by Chernoff's bounds

$$P(|s - rn| > \epsilon n) < 2e^{-\epsilon^2 n/(4r(1-r))}$$

The probability that *any* segment has more than $(1+\epsilon)rn$ balls is at most half the upper bound times the number of segments, namely

$$e^{-\epsilon^2 n/(4r(1-r))} 1/r < e^{-\epsilon^2 n/4}.$$

Once the segment is chosen, the process scans the bins in the segment sequentially, until a key is obtained. (Bins are scanned from lower to higher index.) After the $rm = r(1+c)n$ bins of a segment are invoked, we expect an α fraction of them to be successful (they will assign their key). Again, by a straightforward application of the Chernoff bounds we see that the probability that in *any* segment there are less than $(1-\epsilon)rm$ successful invocations is exponentially small. What we need is for the number of successful invocation in a segment to take care of all balls which fell in a segment. That is,

$$(1-\epsilon)\alpha rm = (1-\epsilon)\alpha r(1+c)n \geq (1+\epsilon)rn$$

which is satisfied if and only if

$$\alpha \geq \frac{1+\epsilon}{(1-\epsilon)(1+c)} \approx \frac{1+2\epsilon}{1+c}.$$

□

The last equation shows a trade-off between α, the success probability of the Test-and-Set once it is invoked, and c, the fractional amount of extra key-space. Conforming to intuition, by increasing the reliability we can decrease the key-space size. In particular, we can push it arbitrarily close to optimal. Notice that with high probability no ball needs more than $rm = a \log n$ Test-and-Set invocations. The pseudocode for the protocol is given in Figure 3.

REFERENCES

1. AFEK, Y., H. ATTIYA, D. DOLEV, E. GAFNI, M. MERRITT, AND N. SHAVIT Atomic snapshots of shared memory. *Proc. 9th ACM Symposium on Principles of Distributed Computing*, 1990, pp. 1-14.
2. ANDERSON, J. Composite registers. *Proc. 9th ACM Symposium on Principles of Distributed Computing*, 1990, pp. 15-30. (Also in *Distributed Computing.*
3. ATTIYA, H., BAR-NOY, A., DOLEV, D., PELEG, D., AND REISCHUK, R. Renaming in an Asynchronous Environment. *J. ACM 37*, 3 (July 1990), 524–548. Also appeared as: Achievable cases in Asynchronous Environment. In *Proceedings of 28th IEEE Symposium on Foundations of Computer Science*, 1987, pp. 337–346.
4. AFEK, Y., GAFNI, E., TROMP, J., AND VITÁNYI, P.M.B. Wait-free test-and-set In *Proceedings of the 6th International Workshop on Distributed Algorithms*, volume 647 of *Lecture Notes in Computer Science*, Springer-Verlag 1992, pp. 85–94.
5. BAR-NOY, A., AND DOLEV. D. Shared Memory vs. Message-passing in an Asynchronous Distributed Environment. In *Proceedings of the 12th ACM Symposium on Principles of Distributed Computing*, 1993, pp. 41–52.
6. BOROWSKY. E., AND GAFNI. E. Immediate Atomic Snapshots and Fast Renaming. In *Proceedings of the 8th ACM Symposium on Principles of Distributed Computing*, 1989, pp. 307–318.
7. W. FELLER An Introduction to Probability Theory and Its Applications, vol 1. 1968. (Third edition.)
8. FISHER, M.J., LYNCH, N.A., AND PATERSON, M.S. Impossibility of Distributed Consensus with One Faulty Processor. *J. ACM 32*, 2 (Apr. 1985), 374–382.
9. HERLIHY, M. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems 13*, 1 (Jan. 1991), pp. 124–149.
10. HERLIHY, M., AND SHAVIT, N. The Asynchronous Computability Theorem for t-Resilient Tasks. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, 1993, pp. 111–120.

11. TANENBAUM, A. *Computer Networks*. Prentice-Hall, 1981.

12. KUTTEN S., OSTROVSKY R. AND PATT-SHAMIR B. The Las-Vegas Processor Identity Problem (How and When to Be Unique). In *Proceedings of the 2nd Israel Symposium on Theory of Computing and Systems*, 1993.

13. L. LAMPORT. On Interprocess Communication. *Distributed Computing 1*, 1, 1986, pp. 86–101.

14. LOUI, M.C., AND ABU-AMARA, H.H. Memory Requirements for Agreement Among Unreliable Asynchronous Processes. *Advances in Computer Research*, vol. 4, JAI Press, Inc. 1987, pp. 163–183.

15. Ö. EĞECIOĞLU AND A. K. SINGH, Naming Symmetric Processes Using Shared Variables, *Distributed Computing*, 8:1(1994).

16. RABIN, M.O. The Choice Coordination Problem. *Acta Informatica 17*, (1982), 121–134.