Article

Naming in Multichannel with Beeps in the Strong Model

Layla S. Aldawsari ^{1,2,*} and Tom Altman ¹

¹ Department of Computer Science and Engineering, College of Engineering, Design and Computing, University of Colorado Denver, Denver, CO 80204, USA; tom.altman@ucdenver.edu

² College of Computer and Information Sciences, Princess Nourah Bint Abdulrahman University, Riyadh 11671, Saudi Arabia

* Correspondence: layla.aldawsari@ucdenver.edu or lsaldossary@pnu.edu.sa

Received: 17 September 2020; Accepted: 8 October 2020; Published: 14 October 2020

Abstract: In this paper, a system of anonymous processes is considered that communicates with beeps through multiple channels in a synchronous communication model. In beeping channels, processes are limited to hearing either a beep or a silence from the channel with no collision detection. A strong model is assumed in which a process can beep on any single channel and listen on any specific channel during a single round. The goal is to develop distributed naming algorithms for two models where the number of processes is either known or unknown. A Las Vegas algorithm was developed for naming anonymous processes when the number of processes is known. This algorithm has an optimal time complexity of $\mathcal{O}(n \log n)$ rounds and uses $\mathcal{O}(n \log n)$ random bits, where n is the number of processes for the largest group. For the model with an unknown number of processes, a Monte Carlo algorithm was developed, which has an optimal running time of $\mathcal{O}(n \log n)$ rounds and a probability of success that is at least $1 - \frac{1}{2} \Omega(\log n)$. The algorithms solve the naming problem in new models where processes communicate through multiple channels.

Keywords: anonymous processes; naming algorithm; beeping channels; distributed naming

1. Introduction

Processes in a distributed system must be assigned unique names to communicate and perform the required computations. Naming anonymous processes is a critical problem and is at the core of building distributed systems. The assignment of unique names to anonymous processes is known as the *naming problem* [1–3]. The naming problem is also known as an initialization protocol for ad hoc networks, where no central control or infrastructure exist. This implies the necessity of an initialization step for assigning unique names to all processes [4–8]. Having unique names for all processes is necessary for them to communicate and perform useful computations.

In wireless communication, processes typically communicate by transmitting messages consisting of binary bits. A new topic of research was introduced in which the processes have limited communication capabilities [9]. This type of communication is known as the beeping model or beeping channels, where processes are limited to carrier sensing. Compared to messaging models, the beeping model is typically more reliable and has significantly less energy consumption [9]. Beeping channels are also known as *Beep Listen* (BL), where a process can either listen or beep on the channel. Two models for beeping channels have been studied in which a channel can either have collision detection or not [10,11]. We consider beeping channels without collision detection, which makes the design of naming algorithms more challenging.

The beeping model received significant attention in recent years due to its simplicity and generality, which makes it applicable in many scenarios [12–15]. It provides an abstraction that is appropriate

for representing wireless communication networks. One of the applications of beeping model is wireless sensor networks which have grown in popularity due to the low cost of manufacturing. Many applications exist for such beeping channels in different fields such as biological sciences [16]. An example of work in this field is the development of an MIS algorithm as part of the fly's nervous system [17]. In [18,19], ant colonies were studied in which their communication mechanism resembles a beeping channel through the perception of emitted pheromones.

Existing naming algorithms were developed in [20] for single beeping channels. In [21], new energy-efficient naming algorithms were developed that minimize the number of rounds in which processes have to be active with similar time complexity of $\mathcal{O}(n \log n)$ rounds. The limitation of communication on a single shared channel is performance bottlenecks when the number of communicating processes is high, which causes a high number of collisions. Thus, it is only natural to extend the naming of anonymous processes to multiple channels beeping model.

In this paper, we consider a system of anonymous processes that communicate through multiple beeping channels. The goal is to assign unique names to all processes using a distributed algorithm. The system consists of N anonymous processes that are partitioned into k channels, such that the number of processes for channel j , $j \in 1, 2, \dots, k$, is n_j . The strong model is assumed. Here, a process can only listen and transmit on a single channel. The communication between processes is clearly more difficult if it is carried over multiple beeping channels than a single one. The main challenge for naming in multiple beeping channels originates from the fact that processes cannot identify the source of beeps. This makes it difficult for processes in different groups to synchronize and share their knowledge about their naming compared to the single beeping channel. Moreover, the inability to detect collisions further complicates the design of the naming algorithms.

We developed two novel naming algorithms for assigning unique names to anonymous processes for the models where the number of processes is either known or unknown, respectively. For the model with a known number of processes, a Las Vegas algorithm is developed, called the Multichannel Beep Naming (MCBN) algorithm, which assigns unique names to all processes with an optimal time complexity of $\mathcal{O}(n \log n)$ rounds. The Unknown Multichannel Beep Naming (UMCBN) algorithm is developed for the case in which the number of processes is not known. It is a Monte Carlo algorithm that assigns unique names to all processes with a probability of success that is at least $1 - \frac{1}{2} \Omega(\log n)$. The algorithm has an optimal time complexity of $\mathcal{O}(n \log n)$ rounds. These are the first naming algorithms that work in the multiple beeping channel environment, and there is no existing work on such a model. Our developed naming algorithms allow anonymous processes to have unique names even when they are assigned to different channels and communication capability is very limited. For the single beeping channel case, our naming algorithms have the same time complexity as [20,21], whereas the authors in [21] developed deterministic and randomized algorithms that require processes to have internal unique IDs. However, their randomized algorithm does not guarantee consecutive unique names in the range $(1, n)$. The MCBN and UMCBN algorithms will generally speedup the initialization step in real-world applications and solve the performance bottleneck of a single beeping channel. These algorithms will be useful for many applications in several fields such as wireless sensor networks and distributed cloud computing.

This paper is organized as follows. Section 2 presents an overview of related work from the literature. Section 3 describes some technical preliminaries of the studied model. After that, the MCBN algorithm is described in Section 4 including the proof of correctness and analysis. Then, in Section 5, the UMCBN algorithm is described along with the proof of correctness and analysis. Finally, in Section 6 we present the conclusions of our work.

2. Related Work

Naming in a single beeping channel was studied by Chlebus et al. [20]. The problem of naming anonymous processes was examined in the synchronous communication model when the number of processes is either known or unknown. They developed a Las Vegas algorithm for naming anonymous

processes when the number of processes is known. The concept for the algorithm is to simulate the assignment of balls into bins, where a beep in a round represents the selection of a bin. A process is assigned the first name in the namespace when it is the only beeping process. The algorithm iterates until every process has been assigned a unique name in the range $(1, n)$ where n is the number of processes. The algorithm has a run time complexity of $\mathcal{O}(n \log n)$. The Monte Carlo algorithm was developed for the model with an unknown number of processes and has a run time complexity of $\mathcal{O}(n \log n)$. In this algorithm, the execution is run in phases where a process selects a random string of k bits in each phase. The random string is sent on the channel by each process. A process becomes a winner when the binary string from the channel matches the generated binary string from the process. The algorithm terminates when no additional processes remain and when no collisions are detected.

Similar work that focuses on energy efficiency was done by Andriambolamalala and Ravelomanana on the same model [21]. They developed energy-efficient randomized algorithms for both naming and counting that have an optimal running time of $\mathcal{O}(n \log n)$ and an optimal energy complexity of $\mathcal{O}(\log n)$. A system of anonymous processes was considered in which M nodes ($M \leq n$) have unique IDs. The naming algorithms are built on a deterministic algorithm that runs as *seasons*. For each *season*, the processes contend for the next name by sending their unique IDs bit by bit through beeps. The process with the maximum IDs obtains the next label.

Naming of anonymous processes was studied in ad-hoc networks by Nakano and Olariu, in which they communicate through broadcasting of messages [4]. They developed randomized initialization protocols for two models with either a known or unknown number of processes. The naming protocol has a probability of success that exceeds $1 - \frac{1}{n}$ to assign unique names in $en + \mathcal{O}(\sqrt{n \log n})$ time slots when the number of processes is known, whereas it requires $10n/3 + \mathcal{O}(\sqrt{n \log n})$ time slots when the number of processes is unknown. These naming protocols were used for naming n processes in k channels that use collision detection, which requires $10n/3k + \mathcal{O}(\sqrt{(n \ln n)/k})$ time slots. Nakano and Olariu have also extended the work and proposed energy-efficient initialization protocols for both single and multiple channels without collision detection, where the awake time for the processes is minimized [5].

Bordim et al. constructed energy-efficient initialization protocols for nodes communicating through either single or multiple-channel radio networks [7]. These initialization protocols require $\mathcal{O}(\log \log n)$ awake time slots, which is better compared to the number of awake time slots of the existing developed initialization protocols.

Myoupo developed several initialization protocols for multiple channels even when the group sizes are not uniform [22]. The algorithm consists of a single-channel initialization protocol and a k -channel initialization protocol. The single-channel initialization protocol requires $\mathcal{O}(n / \log 2)$ time slots on average, whereas the k -initialization protocol requires $\mathcal{O}((n - k + 1) / \log 2)$ time slots in the worst case.

Dufoulon et al. studied leader election in multi-hop networks in which nodes communicate with beeps [12]. A deterministic leader election was developed for a synchronous beeping model when the nodes have unique identifiers (IDs), whereas both the size of the network and degree of a node are unknown. The algorithm has an optimal time complexity of $\mathcal{O}(D + \log n)$ rounds, where D is the diameter of the network and n is the number of nodes in the network. Forster et al. has conducted similar work on leader election [23].

Brandes et al. developed an algorithm for estimating the size of a single-hop radio network that consists of n nodes [24]. All nodes have an identical state and no identifiers. The algorithm offers a $(1 + \epsilon)$ approximation of the network size with a probability of success that is at least $1 - 1/f$. Casteigts et al. analyzed the problem of counting nodes in single-hop radio networks [25]. A counting algorithm was developed for a stronger beeping model where nodes can identify collisions. The algorithm has a lower bound of $\Omega(n)$ on the number of rounds needed to count the nodes. They also developed a counting algorithm for the beeping model without collision detection, which has a $\log n$ increase in the number of required rounds and a probability of error ϵ where $\epsilon > 0$.

Much work has been done on single beeping channels to solve different distributed system problems. Fundamental tasks such as broadcasting and multi-broadcast have been studied by Czumaj and Davies in beeping channels [13]. They have presented broadcasting and multi-broadcasting algorithms for both directed and undirected networks that are in most cases optimal. Several improvements to the multi-broadcast algorithms were proposed in [14]. They enhance the required time for the source identification part of the algorithm. The new algorithms use efficient and simple group testing strategies known as *list disjoint matrices* (LDM) and CLDM, which is an extended scheme of LDM.

3. Technical Preliminaries

In this work, a system of anonymous processes communicates through multiple channels with beeps. A beeping channel represents a limited form of communication in which a process can only sense the presence/absence of a signal on the channel. Therefore, when a process hears a beep on the channel, it represents a signal on the channel. Otherwise, when no beep is detected, it represents silence. The processes communicate through beeping on any of the k channels, and each channel is identified by its unique name from the range $(1, k)$. The number of channels k is significantly less than n , where n is the number of processes for the largest group. All processes know k , the number of channels, and they can beep on any specific channel using the channel's name.

A distributed system is considered that consists of N anonymous processes that are partitioned into k channels, see Figure 1. Each channel j has n_j processes, where $j \in (1, k)$ such that $\sum_{j=1}^k n_j = N$. The N processes are arbitrarily assigned to k channels. A process that is assigned to channel j is said to be a member of group j . The number of processes for group j is denoted by n_j , where $n_j \geq 1$.

Figure 1. Distributed system overview with anonymous processes assigned to k channels.

The model of communication is synchronous and the time is partitioned into equal time slots, also known as rounds. The clocks of all processes are synchronized such that each process knows the start and the end of each round. All processes begin execution simultaneously with synchronized clocks and identical initial states.

A strong communication model is assumed in which a process can only beep on a single channel in any given round. In addition, the processes can only listen for beeps on a single channel in a single round. Although a process is assigned to a specific channel, it can choose to listen to any other channel during an execution. During a round, when two or more processes beep on the same channel, each process listening to the channel cannot identify the source of the beep. Otherwise, when no beep is transmitted on a channel, the feedback is silence.

4. Known Number of Processes Multichannel Naming with Beeps

In this section, the MCBN algorithm is described for assigning unique names to all processes when they communicate in multiple channels with beeps and the number of processes is known. The algorithm is a Las Vegas algorithm, which is guaranteed to terminate, and on termination all processes are assigned unique names with no errors. In the MCBN algorithm, the problem of assigning unique names to all processes is divided into two sub-problems: assigning unique names to each group individually, and forming a consecutive set of unique names. Therefore, the MCBN algorithm consists of two main parts, see Figure 2. In the first part, all processes in all channels execute the Single Group Beep Naming (SGBN) algorithm to assign unique names for each group individually, see Figure 3. The SGBN algorithm is designed based on the algorithm Beep-Naming-LV, which was developed in [20] for assigning unique names to all processes in a single channel. When the SGBN algorithm terminates successfully for group j , all processes that are members of group j are assigned unique names in the range $(1, n_j)$.

Figure 2. Flow chart of the two parts in the MCBN algorithm.

Figure 3. How names are assigned in the first part of the MCBN and the UMCBN algorithms.

Next, how the SGBN algorithm assigns unique names to all processes is described in detail. The pseudocode for the algorithm is shown in Algorithm 1. In the beginning, a process randomly chooses its round, say r , uniformly at random from the range $(1, \alpha n \log n)$, where n is the number of processes and α is an arbitrary value greater than or equal to 1. Each process keeps track of the round number through an internal variable *round* initialized to 0, which is incremented by 1 for every elapsed round. Therefore, when the value of *round* matches the selected random round r , the process transmits a beep on its assigned channel. Because all channels have no collision detection, a beep that is heard on the channel during round j may come from one or more transmitting processes in the same

round. Therefore, any beep heard in the round loop must be verified for the collision of two or more processes, so that a unique name is assigned to a single process.

Algorithm 1: SINGLE GROUP BEEP NAMING

```

repeat
  /* algorithm repeat loop */
   $j \leftarrow$  current group number
   $n \leftarrow$  number of processes in the group
   $round \leftarrow 0$ 
   $FB \leftarrow \emptyset$ 
   $c_p \leftarrow 1$ 
   $rangeEnd \leftarrow \alpha n \log n$ 
   $name_p \leftarrow \emptyset$ 
   $r_p \leftarrow \text{random}(1, rangeEnd)$ 
  repeat
    /* round loop */
    if  $r_p = t$  then
      beep
    /* round ended */
     $round \leftarrow round + 1$ ;
     $FB \leftarrow \text{getChannelFeedback}(j)$ 
    if feedback is a beep then
      /* start of collision-checking segment */
       $collision \leftarrow false$ 
      for  $i \leftarrow 1$  to  $\alpha \log n$  do
        if Check_Collision() then
           $collision \leftarrow true$ 
      if  $collision = false$  then
         $c_p \leftarrow c_p + 1$ 
         $name_p \leftarrow c_p$ 
      else
         $r_p \leftarrow \text{random}(round, rangeEnd)$ 
    /* end of segment */
  until  $round = \alpha n \log n$ 
until  $c_p = n$ 

```

The Check-Collision procedure in Algorithm 2 verifies whether one or more processes have chosen the same round. Thus, all processes enter a collision-checking segment in which the counting of the round is temporarily paused. In the collision-checking segment, all processes that have chosen the same round participate in the execution of the Check-Collision procedure, which is executed $\alpha \log n$ times, where n is the number of processes in the current group. Each call to the Check-Collision procedure consists of two rounds. The Check-Collision procedure begins by using a random coin flip method, which determines whether the process beeps in the first or the second round. The process continues listening to the channel on both rounds. A collision is detected when a beep is heard in both rounds, which causes the termination of the collision-checking segment. Therefore, all processes that have participated in the collision-checking segment choose new random rounds.

After $\alpha \log n$ calls to the Check-Collision procedure with no collisions detected, the collision-checking segment is terminated and the participating process increments the internal counter c , which was originally 0, and assigns itself the new value of c as a name. Because all other processes continue listening to the channel, they can identify that a process has successfully assigned itself a name. Thus,

all processes in the group increment the internal counter value by 1. This ensures that all processes are assigned unique names from a consecutive set of names starting from 1. Similarly, all processes beep during their specifically selected random round and verify whether a collision has taken place.

Algorithm 2: CHECK-COLLISION PROCEDURE

```

round  $\leftarrow$  0
x  $\leftarrow$  random(0, 1)
j  $\leftarrow$  current group number
FB1  $\leftarrow$   $\emptyset$ 
FB2  $\leftarrow$   $\emptyset$ 
do
  if round = x then
    beep
  if round = 0 then
    FB1  $\leftarrow$  getChannelFeedback(j)
  else if round = 1 then
    FB2  $\leftarrow$  getChannelFeedback(j)
  round  $\leftarrow$  round + 1
until round = 2
if FB1 is beep and FB2 is beep then
  return true
else
  return false

```

One additional step remains regarding the termination condition, where each group only knows about the termination of the SGBN algorithm for that same group but not for the other groups. Therefore, the Termination Notification (TN) algorithm allows each group of processes to be notified about the termination of all other groups, see Algorithm 3. This is a necessary step so that all groups of processes know when to begin the second part of the naming algorithm in which all processes communicate to form a contiguous set of unique names with a namespace size of n . At the start of the TN algorithm, a process is designated as a leader for each group. Because each group of processes has been assigned unique names, designating a leader process becomes trivial. The process with the lowest name (ID) is designated as the group leader. The leader process executes the TN algorithm, which is described below.

The process of notifying other processes about the termination requires the use of a channel dedicated for control communication. Therefore, Channel 0 is designated as a control channel through which the leader processes from each channel group will use to notify other group leaders about the termination of the SGBN algorithm. The execution of the TN algorithm is partitioned into segments, where each segment consists of a cycle of k rounds. For each group j , the leader process P_j will beep on the j^{th} round of a segment, while listening for beeps on all other rounds. The termination of the SGBN algorithm for all groups is detected when a total of k beeps have been heard in a segment. Otherwise, when the number of beeps is less than k , each leader process keeps beeping on its designated round in each segment. The algorithm is executed repeatedly for each segment until a beep is heard in all k rounds. This means that all group leaders are notified about the termination of the SGBN algorithm in all groups.

Algorithm 3: TERMINATION NOTIFICATION

```

totalGroups ← number of groups
name ← current name of the process
j ← current group number
n ← the number of processes for group j
totalBeeps ← 0
round ← 1
FB ← ∅
/* waiting for k beeps to be heard */
repeat
    if round = j then
        BeepOnChannel(0)
        FB ← getChannelFeedback(0)
        if FB is beep then
            totalBeeps ← totalBeeps + 1
            round = round + 1
        if round = (k + 1) then
            round = 1
            totalBeeps ← 0
until totalBeeps = k

```

The second part of the algorithm begins when each group of processes detects the termination condition. In this part of the algorithm, all processes execute the Multi Group Renaming (MGR) algorithm shown in Algorithm 4. The goal is to make the set of all names for all processes form a consecutive set of unique names starting from 1 until N . The previously designated leaders communicate with other processes in other groups. The leader of group j sends the total number of processes n_j to the processes in the next group, i.e., group $j + 1$. The procedure of sending the number of processes n_j is performed by encoding the number to its binary form: representation $Binary(n_j)$. The binary representation is then sent to group $j + 1$ through a special encoding of bits called *binary beep encoding*. Sending a single bit requires a pair of rounds, such that a single beep in the first round represents 0, whereas a single beep in the second round represents 1 (similar to Morse code). The end of a binary number is encoded in a pair of rounds with a beep in the first and second rounds. All processes that are members of group $j + 1$ receive the number of processes from the previous group and update their assigned names as follows,

$$P_{id} = P_{id} + n_j,$$

where P_{id} is the current name of the process P .

The MGR algorithm for forming a consecutive set of names begins from the first group in the group set (Group 1). The leader of Group 1 starts sending the number of processes to the next group in sequence, which is Group 2, as soon as the termination condition is detected for the first part of the naming algorithm. Because all processes in Group 2 continue listening to their assigned channel, each process receives the number of processes of Group 1 as a binary number encoded using beeps. When the number of processes is received from the previous group, all processes in Group 2 update their names accordingly. Similarly, the leader of Group 2 sends the total number of processes for Groups 1 and 2 to the next group in sequence. The next group, Group 3, receives the value n_y , where $n_y = n_1 + n_2$, which represents the total number of processes in all previous groups. This process continues until Group k receives the total number of processes in all previous groups. Afterward, all processes from all groups are assigned unique names with an optimal namespace size of N , where $N = \sum_{i=1}^k n_i$.

Algorithm 4: MULTI GROUP RENAMING

```

name ← current name of the process
j ← current group number
n ← the number of processes for group j
totalProcessss ← 0
previousCount ← ∅
isNumberReceived ← false
if j ≠ 1 then
    /* waiting for group j − 1                                     */
    repeat
        isNumberReceived ← ListenAndProcessBinary(previousCount)
    until isNumberReceived = true
    name ← name + previousCount
    totalProcessss ← previousCount + n
    if IsLeader(j) = true then
        TransformAndSend(totalProcessss, j + 1)
    else if IsLeader(j) = true then
        TransformAndSend(n, j + 1)

```

Proof of Correctness and Analysis

In this section, we prove the correctness of both parts of the MCBN algorithm and analyze the time complexity and the number of random bits needed. The procedure for the SGBN algorithm is represented as the operation of assigning a process to a random round until it is singleton. A process is a singleton when no other processes have selected the same round. Checking for a collision in the selected round requires a maximum of $\alpha \log n$ calls to the Check-Collision procedure.

Lemma 1. *After i processes execute the Check-Collision procedure k times, then the probability of failing to detect a collision is $\frac{1}{2}^{(i-1)k}$.*

Proof. There are i processes calling the Check-Collision procedure. Each call consists of two rounds. For each call, two possibilities exist for the outcome of choosing which round to beep in, which is determined by a random coin flip. Thus, a process can either beep in the first or second round, but not both. If the outcome of the first process is denoted as R , then the probability of another process choosing R is $\frac{1}{2}$. A collision is not detected when all remaining $i - 1$ processes choose the same round R . When the number of processes i is equal to 1, the probability of detecting a collision is 0, because no other processes call the Check-Collision procedure. Otherwise, the probability of not detecting a collision when i processes call the Check-Collision procedure is $\frac{1}{2}^{(i-1)}$, given $i \geq 2$. To derive the probability of k different calls of the Check-Collision procedure, the probability is multiplied k times. Therefore, the probability of failing to detect a collision for k Check-Collision procedure calls is $\frac{1}{2}^{(i-1)k}$. $\square$

Next, the number of times a process chooses a random round before getting a name is estimated. We recall the result from [20], which states:

Lemma 2. *The number of times a process chooses a random round is at most $3n$ with a probability of $e^{-\frac{n}{4}}$.*

The authors in [20] also proved the following theorem, which we refer to regarding the correctness of the SGBN algorithm.

Theorem 1. *The SGBN algorithm is guaranteed to terminate, and on termination, it assigns unique names to all processes that require at most $cn \log n$ rounds and uses at most $cn \log n$ random bits with a probability of success at least $1 - \left(\frac{1}{2}\right)^{\alpha(n \log n - \log n)^m}$.*

Based on Lemma 1, the time complexity of the SGBN algorithm is computed. At most $3n$ round selections are needed before all processes are assigned unique names. For each round selection, $\alpha \log n$ Check-Collision procedure calls are made, and each call takes exactly two rounds. Therefore, the time complexity is $\mathcal{O}(n \log n)$ rounds because the algorithm requires at most $6\alpha n \log n$ rounds. Thus, at most, $3\alpha n \log n$ Check-Collision procedure calls are made, which implies that the algorithm uses at most $cn \log n$ random bits. The SGBN algorithm continues repeating the procedure of choosing unique rounds for all processes until they are all assigned unique names in the range $(1, n)$ and $counter = n$.

Next, the correctness of the second part of the algorithm is proved, which consists of the TN algorithm and the MCBN algorithm. First, the correctness of the TN algorithm is demonstrated in which each group is notified about the termination of the SGBN algorithm by every other group. We begin with the initial step when the SGBN algorithm terminates for group j , and the process with the lowest name is designated as the leader for the group. Afterward, it is demonstrated that exactly one process is assigned as a leader for each group j . From Theorem 1, all processes are assigned unique names from the range $(1, n)$. Thus, a process knows that it is the leader of its group when its name is 1. Assuming that two or more processes are assigned as group leaders, it would imply they are assigned the same name of 1. This contradicts Theorem 1, which states that when the algorithm terminates all processes are assigned unique names. Therefore, exactly one process is assigned as a leader for group j .

Theorem 2. *The Termination Notification algorithm terminates after $\mathcal{O}(n \log n)$ rounds and when it terminates, the processes in all groups are notified about the termination of the SGBN algorithm.*

Proof. It is shown that the leader of each group j for all $j \in (1, k)$ obtains the knowledge about the termination in all other groups. First, we begin with the initial state where the SGBN algorithm has terminated for group h , $1 \leq h \leq k$. Then, the leader process of group j begins executing the TN algorithm on which the leader keeps counting the number of beeps heard on Channel 0 for every k rounds. The leader process of group j beeps on Channel 0 during the j^{th} round. When the total number of beeps is less than k , the same process is repeated for every k rounds. When the leader process of group j has heard k beeps during the current cycle, this implies that group j has finished executing the TN algorithm. Because the leader process of group j has heard k beeps, it implies that all other processes in other groups have also heard k beeps during the current cycle of k rounds. Thus, the algorithm terminates for all groups, and when it terminates, the processes of each group obtain knowledge regarding the termination of the SGBN algorithm in all other $k - 1$ groups.

Next, the time required for the TN algorithm to terminate is analyzed. Assuming that group h has finished the execution of the SGBN algorithm and began executing the TN algorithm, the number of times the TN algorithm repeats is estimated. From Theorem 1, the SGBN algorithm has a time complexity of $\mathcal{O}(n \log n)$ rounds, which implies that the leader process of group j must wait for at most $\mathcal{O}(n \log n)$ rounds. Because the leader processes of each group run the TN algorithm simultaneously, the time complexity for the algorithm is $\mathcal{O}(n \log n)$ rounds. Two cases need to be examined to calculate the time complexity. If the number of processes n is the same for all groups, then the time complexity is $\mathcal{O}(n \log n)$ rounds. Otherwise, the time complexity is $\mathcal{O}(n_i \log n_i)$ rounds, where i is the group with the highest number of processes. $\square$

Theorem 3. *The Multichannel Beep Naming algorithm terminates with a time complexity of $\mathcal{O}(n \log n)$ rounds, and when it terminates, all processes are assigned unique names from the range $(1, N)$.*

Proof. Initially, from Theorems 1 and 2 it is known that all processes in each group are assigned unique names from the range $(1, n_j)$ where $1 \leq j \leq k$, and that the leader process of each group knows

about the termination of the SGBN algorithm for all other groups. Therefore, all processes in Group 1 are assigned unique names in the range $(1, n_1)$, where n_1 is the number of processes in Group 1.

Next, it is demonstrated that when the number of processes is sent by the leader process of group x to the next group in the sequence, the combined names of the processes in the two groups form a unique set of names in the range $(1, n_y)$, where n_y is the total number of processes for all groups from 1 to n_{x+1} . In the multichannel naming algorithm, each group apart from Group 1 waits until the total number of processes is received. The leader of Group 1 immediately begins the transmission of the total number of processes n_1 to the next group in sequence, which is Group 2. The leader process transforms n_1 into its binary form and sends it using the previously described binary beep encoding. When the processes of Group 2 receive the number of processes, each process updates its name by adding the total number of processes n_1 received from Group 1 to its name. Now, all processes in Groups 1 and 2 are assigned unique names in the range $(1, (n_1 + n_2))$. Next, it is illustrated that, when the leader process of group x sends the total number of processes to group $x + 1$, the combined names of all processes in all groups from 1 to $x + 1$ are unique. When the processes of group $x + 1$ receive the total number of processes of all groups from 1 to x , each process that is a part of group $x + 1$ updates its name by adding the total number of processes. This results in the assignment of new unique names for the processes in group $x + 1$. All processes in all groups are assigned unique names in the range $(1, n_z)$, where n_z is the total number of processes for all groups from 1 to $x + 1$.

Finally, it is demonstrated that all processes in all groups from 1 to k are assigned unique names in the range $(1, N)$, where $N = \sum_{i=1}^k n_i$. It was proved that when the leader process of group x sends the total number of processes to group $x + 1$, the combined names of all processes are assigned unique names. Therefore, when the leader process of group $k - 1$ sends the total number of processes to group k , the processes in group k update their names, which results in unique names that are different from the existing names in the groups $(1, (k - 1))$. Thus, all processes in all groups are assigned unique names from the range $(1, N)$, which completes the uniqueness part of the proof.

Next, the expected number of rounds needed to finish assigning unique names to all process in all groups $(1, k)$ is computed. The time complexity for the MGR algorithm is computed as the total sum of rounds needed to transmit the number of processes from each group. The binary representation of each number is transmitted using $\log n + 1$ rounds. The number of rounds required to finish forming a consecutive set of unique names is $(k - 1)(\log n + 1)$ rounds. Thus, the time complexity of the second part of the naming algorithm is $\mathcal{O}(n \log n)$ rounds. The total time complexity is the sum of the time complexities for each part of the MCBN algorithm, which is $\mathcal{O}(n \log n)$ rounds.

As the lower bounds for the single channel naming algorithm is $\mathcal{O}(n \log n)$ [20], then that bound carries to the multiple beeping channels problem addressed in this paper, thus making the asymptotic optimality of our algorithm obvious. $\square$

5. Unknown Number of Processes Multichannel Naming with Beeps

In this section, the UMCBN algorithm is described for the model with unknown number of processes. The naming of anonymous processes in multiple channels comprises two parts, see Figure 4. First, unique names are individually assigned to each group of processes, where a group represents processes assigned to the same channel, see Figure 3. Second, processes from all groups communicate to construct a consecutive set of unique names.

In the first part, a Monte Carlo algorithm was developed for assigning unique names to all processes with a very small probability of error. The concept of the algorithm resembles the assignment of balls into several bins. The goal is for each ball to be a singleton in its own bin. When one of the balls is not a singleton in its bin, the number of bins is incremented by an exponential factor, and the assignment of processes to bins is repeated. This process continues until all balls become singletons in their bins. In the Unknown Single Group Beep Naming (USGBN) algorithm, each process chooses a random round from a predefined range of rounds. When two or more processes are assigned to the

same round, the assignment of processes to random rounds is repeated using a larger range of rounds. This operation is repeated until all processes become singletons in their selected rounds.

Figure 4. Flow chart of the two parts in the UMCBN algorithm.

Next, how the USGBN algorithm works is described. It is based on the pseudocode in Algorithm 5. The algorithm consists of two main loops, which are referred to as the repeat loop and the round loop. In the repeat loop, the processes are repeatedly assigned to random rounds when a collision is detected. The algorithm is repeated for a larger range of rounds, and the range is incremented based on the exponent t and a base of 2. The exponent is incremented by a factor of 2. If no collision is detected, then all processes have unique names and the main repeat loop is terminated.

Before beginning the execution of the round loop, each process chooses a random round from the range of rounds. During the round loop, each process waits for its selected round before beeping on the channel. When the round number matches the selected round, the process beeps on the channel. For each beep on the channel, a collision-checking segment begins. The objective for the collision-checking segment is to verify that a single process has chosen the selected round so that it is assigned a unique name. In the collision-checking segment, the procedure Check-Collision is run for αt times. When no collision is detected, all processes which selected that particular round are assigned a name based on the value of the counter. The value of the counter is identically incremented by all processes because all processes identify that no collision is detected. Otherwise, when a collision is detected, the round loop is restarted by setting the value of *reset* to “true”. Before beginning a new round loop, the range of rounds is obtained as follows,

$$range_size = 2^t,$$

where t is doubled for every restart of the round loop. After that, all processes choose random rounds from the new range of rounds. The round loop is repeated until no collision is detected for each beep that is heard during its execution. This implies that all processes have been assigned unique names.

Algorithm 5: UNKNOWN SINGLE GROUP BEEP NAMING

```

 $t \leftarrow 1$ 
 $round \leftarrow 0$ 
 $counter \leftarrow 0$ 
 $collision \leftarrow false$ 
 $reset \leftarrow false$ 
 $j \leftarrow \text{current group number}$ 
repeat
/* algorithm repeat loop */
 $range\_size \leftarrow 2^t$ 
 $name_p \leftarrow \emptyset$ 
 $r_p \leftarrow \text{random}(1, range\_size)$ 
repeat
/* round loop */
if  $r_p = round$  then
    beep
/* end of round loop */
 $round \leftarrow round + 1$ 
 $FB \leftarrow \text{getChannelFeedback}(j)$ 
if  $FB$  is beep then
/* start of collision-checking segment */
 $collision \leftarrow false$ 
for  $i \leftarrow 1$  to  $range\_size$  do
    if Check_Collision then
         $collision \leftarrow true$ 
         $reset \leftarrow true$ 
/* end of segment */
if  $collision = false$  then
     $counter \leftarrow counter + 1$ 
     $name_p \leftarrow counter$ 
if  $name_p = \emptyset$  then
        beep
     $FB \leftarrow \text{getChannelFeedback}(j)$ 
until  $reset = true$  or  $FB$  is silence
 $t \leftarrow t * 2$ 
until  $reset = false$ 

```

Proof of Correctness and Analysis

In this section, the correctness of the USGBN algorithm is proved. To do this, it is shown that the algorithm has a probability of success that is at least $\frac{1}{2}$.

Theorem 4. *The USGBN algorithm assigns unique names to a set of anonymous processes, which has a time complexity of $\mathcal{O}(n \log n)$ rounds and uses at most $cn \log n$ random bits, and terminates with a high probability of success that is at least $1 - \frac{1}{2}^{\Omega(\log n)}$.*

Proof. In the round loop of the algorithm, a process chooses a random round from the range of rounds denoted as $range_size$. The value of $range_size$ is derived based on the following equation,

$$range_size = 2^t,$$

where the value t denotes the t^{th} iteration of the main loop.

For each beep heard during the round loop within the $range_size$, the Check-Collision procedure is executed αt times. The probability of error is derived based on the number of times the Check-Collision procedure is executed. Therefore, two cases must be considered in which $range_size < n$ or $range_size \geq n$ where n is the number of processes. In each case, the probability of error is estimated in which a collision is not detected.

From Lemma 1, the probability of failing to detect a collision is obtained as follows,

$$\frac{1}{2}^{(i-1)k},$$

where i is the number of processes, and k is the number of times the Check-Collision procedure is called. In the first case, the upper bound of failing to detect a collision is estimated when $t = 1$, which is the initial value. The probability of error is at its highest value when $n/2$ processes choose the same round. The probability of failing to detect a collision is estimated to be at most

$$\frac{1}{2}^{\alpha \frac{n}{2}} = \frac{1}{2}^{\Omega(n)}.$$

Therefore, the probability of successfully assigning unique names is at least

$$1 - \frac{1}{2}^{\Omega(n)},$$

which is greater than $\frac{1}{2}$.

When the $range_size \geq n$, processes have a higher chance of choosing unique rounds. Thus, the upper bound of failing to detect a collision is obtained when exactly two processes choose the same round. When the range of rounds is greater than or equal to the number of processes then $t = \log n$. From Lemma 1, the upper bound for the probability of error is derived as follows,

$$\frac{1}{2}^{\alpha \log n} = \frac{1}{2}^{\Omega(\log n)}.$$

Therefore, the probability of successfully assigning unique names is $1 - \frac{1}{2}^{\Omega(\log n)} \geq \frac{1}{2}$. Thus, the USGBN algorithm assigns unique names to all processes with a probability of success that is at least $\frac{1}{2}$. This concludes the uniqueness of names portion of the proof.

Now, we estimate the time complexity of the algorithm to successfully assign unique names to all processes by computing the number of rounds needed. For each beep heard during an iteration of the round loop, the Check-Collision procedure is called αt times. The algorithm terminates with a high probability of success when $2^t \geq n$, i.e., $t \geq \log n$. Thus, the total number of rounds required to assign unique names is obtained as follows,

$$\sum_{t=1}^{\log n} 2^t \alpha t = \mathcal{O}(n \log n).$$

Therefore, the algorithm has a time complexity of $\mathcal{O}(n \log n)$ rounds for assigning unique names to n processes with a high probability of success. As it was shown that the algorithm requires $\mathcal{O}(n \log n)$ when $t \geq \log n$, then the expected number of bits is computed based on the number of times each process calls the Check-Collision procedure. Each process calls the Check-Collision procedure for at most $\alpha t = \alpha \log n$ and each call requires one random bit. Therefore, for n processes, the required random bits is $\alpha n \log n$. $\square$

Theorem 5. *The Termination Notification algorithm terminates after $\mathcal{O}(n \log n)$ rounds, and when it terminates, the processes in all groups are notified about the termination.*

Proof. To prove the correctness of the algorithm, each group must obtain knowledge regarding the termination of the USGBN algorithm. When the USGBN algorithm terminates with a high probability of success, a leader process is designated for group j , where $1 \leq j \leq k$. The process with the name 1 is designated as a leader process for each group. The TN algorithm is executed, as in the model with known number of processes. For every cycle of k rounds, the leader process continues beeping on the

control channel at the j^{th} round, while it continues counting the number of beeps. This process repeats until the number of beeps equals k . We know that the leader of group j has obviously terminated the execution of the TN algorithm in which k beeps occurred during a cycle of k rounds. It implies that the leader process of other groups have also been notified about the termination of all other groups. Thus, all groups obtain knowledge concerning the termination of the USGBN algorithm of all remaining groups.

Next, the time complexity of the algorithm is estimated. If group j has finished assigning unique names to all processes, then the leader process continues repeating the algorithm until the USGBN algorithm is terminated in all other groups. Thus, the time complexity is estimated based on the number of rounds needed to finish assigning unique names to a single group, which is $\mathcal{O}(n \log n)$ rounds. $\square$

Theorem 6. *The UMCBN algorithm assigns unique names to all processes in all channels with a high probability of success that is at least $1 - \frac{1}{2}^{\Omega(\log n)}$ and has a time complexity of $\mathcal{O}(n \log n)$ rounds.*

Proof. In order to prove the correctness, it is demonstrated that all processes are assigned unique names. At the beginning, from Theorem 4, all processes in each group are assigned unique names from the range $(1, n)$ with a high probability of success. In addition, based on Theorem 5, all groups have obtained knowledge regarding the termination of the USGBN algorithm for all other groups. All processes in all groups, except Group 1, wait until the value n_y is received through the binary beep encoding. The value n_y represents the total number of processes that have been assigned unique names. The leader process of Group 1 begins executing the algorithm in which the number of processes for Group 1 is sent to the next group in the sequence. Before transmitting the number of processes of Group 1, the number of processes n_1 is transformed into its binary form and sent using the binary beep encoding as described in Section 4. When the number of processes is received by the processes in the next group, all processes update their names by adding the number of processes n_1 received from the first group. Thus, the combined names of all processes in the first two groups form a contiguous set of unique names with a namespace size of $n_1 + n_2$. Similarly, the leader process of the next group begins transmitting the total number of processes for Groups 1 and 2, and so on. Therefore, for each group that receives its total number of processes, the combined names of these processes form a set of unique names for all processes. When group k receives the total number of processes $\sum_{i=1}^{k-1} n_i$, all processes in group k update their names by adding the total number of processes. As a result, all processes are assigned unique names. The expected run time of the MGR algorithm is $(k - 1) \log n$ because $k - 1$ number of processes are transmitted and each transmission requires $2(\log n + 1)$ rounds. Thus, the time complexity of the second part of the algorithm is $\mathcal{O}(n \log n)$ rounds.

Next, we compute the time complexity of assigning unique names to all processes when the number of processes is unknown. The complexity is computed as the sum of the time complexities for the first and second parts of the naming algorithm. Thus, the total time complexity of the UMCBN algorithm is $\mathcal{O}(n \log n)$ rounds. The optimality of the algorithm is shown by considering the special case of a single group when $k = 1$. We refer to Proposition 3 in [20], which states that a randomized algorithm has an expected run time of $\Omega(n \log n)$ rounds whether it is Las Vegas or Monte Carlo. Therefore, the time complexity of the UMCBN algorithm is optimal since it is designed for the general case when $k \geq 1$. $\square$

6. Conclusions

We proposed two naming algorithms for assigning unique names to anonymous processes that communicate in multiple channels with beeps in the strong model. To our knowledge, no such algorithms have been proposed for naming anonymous processes in models wherein they communicate with beeps on multiple channels. For the model with a known number of processes, a novel Las Vegas algorithm, known as the MCBN, has an optimal time complexity of $\mathcal{O}(n \log n)$ rounds and uses $cn \log n$ random bits. When the number of processes is not known, a novel Monte Carlo algorithm,

known as the UMCBN, can assign unique names with a high probability of success. Its time complexity is $\mathcal{O}(n \log n)$ rounds with a probability of success that is at least $\frac{1}{2} \Omega(\log n)$. The algorithms solve the naming problem in two new models and enable the design of solutions for other distributed system problems. They facilitate the deployment of anonymous processes in environments with limited communication capabilities and guarantee the assignment of unique names to all processes with a high probability of success for cases in which the number of processes is unknown. Solving the anonymity problem in multiple channels with beeps leads to the possibility for designing solutions to other distributed system problems in the same model, such as consensus and leader election. A future direction for this work could be to design naming algorithms that are energy efficient for the same models. Moreover, it will be interesting to find how having collision detection can make the algorithms more efficient. Another direction for the work is analyzing other communication models such as ones with extended or multiple beeps consisting of two or more signals.

Author Contributions: Conceptualization, L.S.A.; methodology, L.S.A.; theoretical analysis and proofs, L.S.A. and T.A.; manuscript writing and figures preparation, L.S.A.; writing—review and editing, L.S.A. and T.A.; Supervision, T.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Attiya, H.; Snir, M.; Warmuth, M.K. Computing on an Anonymous Ring. *J. ACM* **1988**, *35*, 845–875. [\[CrossRef\]](#)
2. Lipton, R.J.; Park, A. The processor identity problem. *Inf. Process. Lett.* **1990**, *36*, 91–94. [\[CrossRef\]](#)
3. Eğecioğlu, O.; Singh, A.K. Naming Symmetric Processes Using Shared Variables. *Distrib. Comput.* **1994**, *8*, 19–38. [\[CrossRef\]](#)
4. Nakano, K.; Olariu, S. Randomized initialization protocols for ad hoc networks. *IEEE Trans. Parallel Distrib. Syst.* **2000**, *11*, 749–759. [\[CrossRef\]](#)
5. Nakano, K.; Olariu, S. Energy-efficient initialization protocols for radio networks with no collision detection. In Proceedings of the 2000 International Conference on Parallel Processing, Toronto, ON, Canada, 21–24 August 2000; pp. 263–270.
6. Hsu, C.S.; Sheu, J.P. Initialization protocols for IEEE 802.11-based ad hoc networks. In Proceedings of the Ninth International Conference on Parallel and Distributed Systems, Taoyuan City, Taiwan, 17–20 December 2002; IEEE: Piscataway, NJ, USA, 2002; pp. 273–278.
7. Bordim, J.L.; Cui, J.; Ishii, N.; Nakano, K. Doubly-Logarithmic Energy-Efficient Initialization Protocols for Single-Hop Radio Networks. In Proceedings of the 16th International Parallel and Distributed Processing Symposium, IPDPS '02, Fort Lauderdale, FL, USA, 15–19 April 2002; IEEE Computer Society: Piscataway, NJ, USA, 2002; p. 123.
8. Ravelomanana, V. Optimal Initialization and Gossiping Algorithms for Random Radio Networks. *IEEE Trans. Parallel Distrib. Syst.* **2007**, *18*, 17–28. [\[CrossRef\]](#)
9. Cornejo, A.; Kuhn, F. Deploying wireless networks with beeps. International Symposium on Distributed Computing, Cambridge, MA, USA, 13–15 September 2010; Springer: Berlin/Heidelberg, Germany, 2010; pp. 148–162.
10. Schneider, J.; Wattenhofer, R. What Is the Use of Collision Detection (in Wireless Networks)? In *Distributed Computing*; Lynch, N.A., Shvartsman, A.A., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 133–147.
11. Afek, Y.; Alon, N.; Bar-Joseph, Z.; Cornejo, A.; Haeupler, B.; Kuhn, F. Beeping a Maximal Independent Set. In *Distributed Computing*; Peleg, D., Ed.; Springer: Berlin/Heidelberg, Germany, 2011; pp. 32–50.
12. Dufoulon, F.; Burman, J.; Beauquier, J. Beeping a deterministic time-optimal leader election. In Proceedings of the 32nd International Symposium on Distributed Computing (DISC 2018), New Orleans, LA, USA, 15–19 October 2018; Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik: Wadern, Germany, 2018.
13. Czumaj, A.; Davies, P. Communicating with beeps. *J. Parallel Distrib. Comput.* **2019**, *130*, 98–109. [\[CrossRef\]](#)

14. Beauquier, J.; Burman, J.; Davies, P.; Dufoulon, F. Optimal Multi-broadcast with Beeps Using Group Testing. In *Structural Information and Communication Complexity*; Springer International Publishing: Cham, Switzerland, 2019; pp. 66–80.
15. Efremenko, K.; Kol, G.; Saxena, R.R. Noisy beeps. In Proceedings of the 39th Symposium on Principles of Distributed Computing, Virtual Event, Italy, 3–7 August 2020; pp. 418–427.
16. Navlakha, S.; Bar-Joseph, Z. Distributed Information Processing in Biological and Computational Systems. *Commun. ACM* **2014**, *58*, 94–102. [[CrossRef](#)]
17. Afek, Y.; Alon, N.; Barad, O.; Hornstein, E.; Barkai, N.; Bar-Joseph, Z. A Biological Solution to a Fundamental Distributed Computing Problem. *Science* **2011**, *331*, 183–185. [[CrossRef](#)] [[PubMed](#)]
18. Fonio, E.; Heyman, Y.; Boczkowski, L.; Gelblum, A.; Kosowski, A.; Korman, A.; Feinerman, O. A locally-blazed ant trail achieves efficient collective navigation despite limited information. *eLife* **2016**, *5*, e20185. [[CrossRef](#)] [[PubMed](#)]
19. Alqurashi, R.; Alsultaan, H.; Greene, M.; Altman, T. Agent-Based Models Predicting Collective Behaviors. In Proceedings of the 2017 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 14–16 December 2017; pp. 297–302.
20. Chlebus, B.S.; De Marco, G.; Talo, M. Naming a channel with beeps. *Fundam. Informaticae* **2017**, *153*, 199–219. [[CrossRef](#)]
21. Andriambolamalala, N.A.; Ravelomanana, V. Energy Efficient Naming in Beeping Networks. In *Ad-Hoc, Mobile, and Wireless Networks*; Palattella, M.R., Scanzio, S., Coleri Ergen, S., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 355–369.
22. Myoupo, J.F. Dynamic initialization protocols for mobile ad hoc networks. In Proceedings of the 11th IEEE International Conference on Networks, ICON2003, Sydney, NSW, Australia, 28 September–1 October 2003; pp. 149–154.
23. Förster, K.T.; Seidel, J.; Wattenhofer, R. Deterministic Leader Election in Multi-hop Beeping Networks. In *Distributed Computing*; Kuhn, F., Ed.; Springer: Berlin/Heidelberg, Germany, 2014; pp. 212–226.
24. Brandes, P.; Kardas, M.; Klonowski, M.; Pająk, D.; Wattenhofer, R. Approximating the Size of a Radio Network in Beeping Model. In *Structural Information and Communication Complexity*; Suomela, J., Ed.; Springer International Publishing: Cham, Switzerland, 2016; pp. 358–373.
25. Casteigts, A.; Métivier, Y.; Robson, J.; Zemmari, A. Counting in one-hop beeping networks. *Theor. Comput. Sci.* **2019**, *780*, 20–28. [[CrossRef](#)]

Publisher’s Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).