Distributed Global ID Assignment for Wireless Sensor Networks *

ElMoustapha Ould-Ahmed-Vall, *Douglas M. Blough*, *Bonnie Heck Ferri*, *George F. Riley*

School of Electrical and Computer Engineering, Georgia Institute of Technology
Atlanta, GA 30332-0250
{eouldahm,doug.blough,bonnie.heck,riley}@ece.gatech.edu

Abstract

A sensor network consists of a set of battery-powered nodes, which collaborate to perform sensing tasks in a given environment. It may contain one or more base stations to collect sensed data and possibly relay it to a central processing and storage system. These networks are characterized by scarcity of resources, in particular the available energy.

We present a distributed algorithm to solve the unique ID assignment problem. The proposed solution starts by assigning long unique IDs and organizing nodes in a tree structure. This tree structure is used to compute the size of the network. Then, unique IDs are assigned using the minimum number of bytes. Globally unique IDs are useful in providing many network functions, e.g. configuration, monitoring of individual nodes, and various security mechanisms.

Theoretical and simulation analyses of the proposed solution have been performed. The results demonstrate that a high percentage of nodes (more than 99%) are assigned globally unique IDs at the termination of the algorithm when the algorithm parameters are set properly. Furthermore, the algorithm terminates in a relatively short time that scales well with the network size.

1 Introduction

A sensor network consists of a set of battery-powered nodes, which collaborate to perform sensing tasks in a given environment. It may contain one or more base stations to collect sensed data and possibly relay it to a central processing and storage system.

The communication range of individual sensor nodes is generally limited, and communication is often carried out in a multi-hop manner. There is a need to have a unique identifier in the header of every unicast packet. In fact, routing protocols need to uniquely identify the final destination as any node in the network can be a potential destination. Several routing protocols use attribute-based routing and therefore can use attributes as global identifiers. However, even these protocols require the existence of unique IDs at a local level. This is the case for directed diffusion [1] and geographical routing protocols

*This work is supported in part by NSF under contract numbers ANI-9977544, ANI-0136969, ANI-0240477, ECS-0225417, CNS 0209179, and DARPA under contract number N66002-00-1-8934.

such as [2]. Network-wide unique IDs are beneficial for administrative tasks requiring reliability, such as configuration and monitoring of individual nodes, and download of binary code or data aggregation descriptions to sensor nodes [3]. Network-wide unique IDs are also required when security is needed in sensor networks [4]. Several MAC protocols requiring the preexistence of network-wide unique IDs have also been proposed for sensor networks [5].

Assumption of the preexistence of network-wide IDs is not realistic in the case of sensor networks. The preexistence of network-wide global IDs requires hard-coding these IDs on nodes prior to the deployment. This is costly in terms of time and effort when a network contains thousands to hundreds of thousands of nodes. Another alternative is to have MAC addresses that are unique for every manufactured sensor node, as is the case for Ethernet cards [6]. This is not a desirable approach because of the coordination it requires and the fact these IDs would have to be lengthy and therefore costly to use in packet headers.

An obvious ID assignment strategy is to have each node randomly choose an ID such that the probability of any two nodes choosing the same ID is very low. However, for this probability to be low, we need the IDs to be very long, which is again costly in terms of energy [7]. Any ID assignment solution should produce the shortest possible addresses because sensor networks are energy-constrained. The usage of the minimum number of bytes required is motivated by the need to limit the size of transmitted packets, in particular the header. In fact, communication is usually the main source of energy drain in a sensor node [8]. For this reason, sensor networks are designed to limit the amount of data transmitted, for example through data aggregation. This reduces the payload of transmitted packets, which makes the header size even more significant.

In this paper, we introduce an algorithm that assigns unique IDs to sensor nodes using only the minimum number of bytes required to assign each node in the network a unique ID. The algorithm does not assume the pre-existence of any type of identification and scales well with the size of the network. We also do not assume the existence of any specific communication protocol. In particular, the preexistence of a specific collision avoidance mechanism is not assumed. The algorithm handles collisions through avoidance and recovery. Collisions are avoided through the scheduling of transmissions at random times. If collisions occur, they are detected through a confirmation mechanism, and recovery is performed by retransmitting colliding packets. Our algorithm can handle the case of asynchronous wake-up. Nodes can join the network after the execution of the algorithm and still obtain a unique ID in an energy efficient manner. The handling of asynchronous wake-up is particularly important in the case of sensor networks where many nodes may be in a sleep state during the initialization phase for the purpose of energy saving and network lifetime optimization [9].

The algorithm can be divided into three main phases. In the first phase, a tree structure is established and, at the same time temporary long IDs are assigned. These temporary IDs are used for reliable communication during the remaining two phases. In the second phase, the size of sub-trees is reported bottom-up from leaf nodes to the root. In the third phase, the final short IDs are assigned. Each node participating in the initial phase of the algorithm requests additional spare IDs that are used to locally assign unique IDs to neighboring nodes that asynchronously join the network.

We analytically prove the correctness and termination of the algorithm. We also assess its performance in terms of the execution time and the probability that a node is left without an assigned ID at the end of the algorithm.

It must be noted that a shorter paper describing an initial version of the proposed algorithm was published in [10]. The main differences between the two papers are the addition in this paper of the extension

of the algorithm to handle nodes that join the network in an asynchronous manner, and inclusion of a more comprehensive set of simulation results of the algorithm.

The rest of the paper is organized as follows. The related work is discussed in Section 2. In Section 3, the proposed algorithm is discussed for the case of sensor networks with synchronized deployment. The theoretical analysis of the proposed algorithm is presented in Section 4. Section 5 discusses extensive simulation results of the algorithm. The algorithm is extended in Section 6 to handle the case of sensor networks with asynchronous deployment. Section 7 presents simulation results for the extended algorithm. Section 8 discusses the costs and benefits of using our ID assignment algorithm. The paper is concluded in Section 9.

2 Related Work

In general, network-wide unique addresses are not needed to identify the destination node of a specific packet in sensor networks. In fact, attribute-based addressing fits better with the specificities of sensor networks [11]. In this case, an attribute such as node location and sensor type is used to identify the final destination. However, different nodes can have the same attribute value, in particular in the same neighborhood. Thus, there is a need to uniquely identify the next hop node during packet routing [12]. Furthermore, it is possible that two neighboring nodes have the same attributes. For instance, it is likely that some nodes will have the same location in a dense sensor network. In addition, the number of bits required to represent attribute information (for example the node geographical coordinates) may be large rendering this approach less attractive from a communication energy point of view [13].

In [14], the authors propose an algorithm that assigns globally unique IDs. Like our algorithm, it uses a tree structure to guarantee the uniqueness of each ID. The algorithm is similar to the first phase of our algorithm. It starts with the sink node broadcasting a message that contains its ID and a parameter b given the size in bits of one-hop ID. Successive nodes choose a parent node among their neighbors that already have an ID. The node then randomly chooses an ID of size b bits and relays on its parent to guarantee no other node has chosen the same ID. The node then appends its chosen ID to the ID of its parent to create a unique ID. The main difference between this algorithm and ours is that it does not use the network size to minimize the size of node IDs. Our algorithm, in contrast, not only assigns unique IDs but also guarantees that these IDs are of minimum length. This is is a considerable advantage considering that sensor networks are energy-sensitive.

Several schemes have been proposed to assign locally unique addresses in sensor networks. In [8], Schurgers, et al., developed a distributed allocation scheme where local addresses are spatially reused to reduce the required number of bits. The preexisting MAC addresses are converted into locally unique addresses. Each locally unique address is combined with an attribute-based address to uniquely determine the final destination of a packet. This use of locally unique addresses instead of global addresses does not affect the operations of the existing routing protocols. This solution assumes the preexistence of globally unique addresses, which our algorithm does not assume. Our solution can be used to assign these global addresses prior to use of the method in [8]. Our proposed solution allows for asynchronous wake-up. When a new node joins the network, it chooses a random address and shares it with its neighbors. The neighbors compare the new address with the addresses of other neighbors to detect and resolve any conflicts.

In [12], Ali, et al., proposed an addressing scheme for cluster-based sensor networks [15]. To prevent

collisions, nodes within the same cluster are assigned different local addresses. Non-member one-hop and two-hop neighbors must also have different local addresses to avoid the hidden-terminal problem. The network is divided into hierarchical layers where the number of layers increases with the number of nodes in the network. Global IDs are obtained by concatenating the local address and the addresses of the head nodes of the different layers. This solution suffers from the fact that the address size increases with the number of layers as 6 bits are added for each layer. This makes this solution less attractive due to the energy cost of using global IDs in the case of large sensor networks. In addition, this solution can be used only with cluster-based routing and does not extend to the case of multi-hop routing [16].

In [3], Dunkels, et al., developed a spatial IP addressing scheme using node location. The (x, y) coordinates of a node are used as the two least significant bytes of its spatial IP. This solution is particularly attractive since it can facilitate the interaction between sensor networks and other types of networks. However, it also suffers from the large size of generated addresses leading to higher overhead. It also requires the existence of a localization mechanism since it assumes that nodes are location-aware.

In [13], the authors propose a local ID assignment scheme where address conflicts are resolved in a reactive way. Nodes randomly choose an address that is likely to be unique within a 2-hop neighborhood. No conflict resolution is performed until nodes need to enter in a communication. For instance, interest broadcasting in directed diffusion can be used to resolve conflicts. In this case, the sink node discovers the existence of identical IDs between its immediate neighbors. The conflicting nodes are notified and choose new IDs. Each node that forwards the interest message uses the response messages to detect ID conflicts among its neighbors. The delaying can help save energy by avoiding any unnecessary conflict resolution. In particular, if two neighbors have the choose the same local ID but are never active at the same time, resolving such a conflict amounts to a waste of energy resources. However, resolving ID conflicts reactively can be problematic if the sensor network requires time-sensitive exchange of information, since messages can be delayed to resolve an ID conflict.

In [17], Motegi et al. propose an on-demand address scheme. To reduce the number of bits required to represent addresses and the number of control messages needed to establish these addresses, the authors propose to assign temporary addresses to nodes detecting an event on an on-demand basis. When a node becomes active (detects an event), it chooses a random network-wide ID and sends a route request to the sink node using the AODV protocol [18]. The intermediate and the sink node perform a conflict resolution. Once the node communication with the sink node terminates (the node is no longer active), its address goes back to the free address pool and can be used by newly active nodes. This approach can effective reduce the number of bits required for globally unique IDs. However, it assumes the pre-existence of some of form of network-wide unique IDs (e.g., MAC or IP addresses). The pre-existing addresses are needed to establish locally unique and permanent addresses used by the proposed algorithm for neighbor identification and collision avoidance.

In [19], the authors present the GREENWIS algorithm designed to assign group IDs. The objective is to assign group IDs rather than individual node IDs. Neighboring nodes share the same group ID, which allows the number of bits required to represent each to be much smaller. The main problem with this approach is that it can be used to provide functionalities such as collision avoidance as all neighbors are assigned the same ID.

The ID assignment problem is related to the overall sensor network initialization. Initialization can be viewed as the mechanism needed for individual sensor nodes to become an integral part of a sensor network. When a sensor network is initially deployed, there is no established structure to allow nodes to efficiently communicate [9]. The initialization has the objective of transitioning from this unstructured

state to a structured network and the establishment of a MAC protocol to allow efficient information dissemination. An important way to structure sensor nodes into a network is through the use of clustering techniques [20, 21, 9]. Many of the clustering algorithms that have been proposed for sensor networks assume the pre-existence of unique node IDs. For example, in the approach proposed in [9] a newly active node that wants to join the network waits for messages from neighboring dominators (cluster heads) before trying to become a dominator. A unique ID is needed to distinguish between the different dominators in the neighborhood. The ID assignment algorithm proposed here can be used as part of the overall network initialization phase. In this way, the initialization does not need to require the pre-existence of node IDs. Like the work in [9], our approach does not assume that all nodes wake-up at the same time. We also do not assume the existence of a specific collision avoidance protocol.

3 Unique ID Assignment Algorithm

We present a distributed algorithm that assigns globally unique IDs to sensor nodes. Initially, we assume that all nodes are awake during the execution of the algorithm. This assumption is relaxed later in this paper to accommodate a dynamic network where nodes can join the network at any time during the initial execution of the algorithm or after its termination. The algorithm can be divided into three main phases. In the first phase, the objective is to assign temporary unique identifiers in the form of potentially long vectors of bytes. A tree structure rooted at the node initiating the algorithm is established during this phase. The main difficulty here is to guarantee the uniqueness of the assigned IDs and to minimize cost by controlling the probability of message collisions.

In the second phase, the temporary identifiers are used to reliably compute the size of each sub-tree and report it to the parent node. This process is done for each sub-tree from leaf nodes until the root node. At the end of this phase, the initiator knows the total size of the network. This allows the initiator to compute the minimum number of bytes required to give a unique ID to each node in the tree.

The third phase consists of assigning final IDs to each node in the network going from the root to the leaf nodes. In this phase as in the previous one, it is necessary to control the parameters of the algorithm to minimize the communication energy cost while keeping the execution time of the algorithm short.

These different phases are now described in detail.

3.1 Phase 1: Tree Building and Temporary ID Assignment

In this phase, temporary IDs are assigned and a tree structure is established. The temporary ID of a particular node is a vector of bytes that uniquely identifies it. The temporary ID of a child node has one byte more than that of its parent. We assume a network density, such that no node has more than 256 neighbors. However, for networks of higher density, temporary IDs can be modified to be vectors with elements of 2 or 3 bytes as needed. The algorithm starts with the initiator node, typically the base station, choosing its temporary ID to contain one byte of value 0, and broadcasting an initialization message of type 1. Each node receiving an initialization message for the first time considers its parent to be the sender of the message and initializes its temporary ID to that of its parent node.

The receiving node then chooses a 4-byte integer unfirormly at random and prepares a message of type 2 to send to its parent node. This message also contains a retry counter. The sending of this message is scheduled at a random time uniformly distributed between $timeWait$ and $2 \times timeWait$ from the time

of reception of the initialization message to avoid collision with messages sent by other children. Upon reception of a new message of type 2, the parent node checks if any other child node had already chosen the same random number. If so, a reinitialization message of type 3 is immediately sent to the child node. If no other child had chosen the same number, the parent node immediately sends a message containing an assigned ID of one byte that is different from the ones sent to other children nodes. This message is of type 4. The reception of this message is immediately confirmed to the parent by a confirmation message of type 5.

After receiving the message of type 4 containing the 1-byte unique child ID, this byte is added at the end of the temporary ID. The child node then schedules the sending of an initialization message at random time uniformly distributed between $timeWait$ and $2 \times timeWait$ from the time of reception of the type 4 message. At the scheduled time, the node sends an initialization message of type 1 and waits for a certain amount of time ($5 \times timeWait$) to hear from any potential children. If any child responds within this period, the previous procedure of assigning one byte ID repeats itself. If no child responds, the node considers itself a leaf node.

All messages except the ones of type 1 are exchanged in a reliable way. A message of type 2, which contains the random 4-byte integer chosen by a child node, is confirmed by the reception of a message of type 3 (reinitialization message) or type 4 (containing a 1-byte assigned ID). A message of type 3 is resent if the parent node receives a second message of type 2 with the same 4-byte ID. A message of type 4 is confirmed through the reception of the confirmation message of type 5. If a message is not confirmed within a random period chosen uniformly between $timeWait$ and $2 \times timeWait$, the message is resent. The node keeps checking for a confirmation and resending until the message is confirmed.

Figure 1 illustrates the messages exchanged between a parent node and a child node during phase 1 when no reinitialization message is sent. A reinitialization message makes the child node resends the message of type 2. At the end of this exchange, the child node has a temporary ID that is 1 byte longer than the one of its parent node. The child node then sends its own message of type 1. Algorithm 3.1 gives the pseudo code of the first phase. Note that at the end of this phase every node knows the temporary ID of its parent node. The parent ID is equal to the node ID without the last byte.

3.2 Phase 2: Collecting the Sub-Tree Sizes

In this phase, nodes report their sub-tree sizes from the leaf nodes to the root node. The sub-tree size of a particular node is the number of nodes contained in the tree rooted at that node at the end of phase 1. A node that is declared a leaf at the end of phase 1 considers its sub-tree size to be 1 and sends it as a message of type 6 to its parent. A non-leaf node waits until it receives sub-tree sizes from all of its children nodes before sending its sub-tree size to its parent. Sub-tree size messages are confirmed by the parent node with a confirmation message of type 7. Figure 2 illustrates the message exchange during phase 2 to collect the sub-tree sizes.

When the initiator receives sub-tree size messages from all of its children, it knows the total number of nodes in the network. This total is used to compute the minimum number of bytes needed to code a unique final ID for each node in the network. These IDs are assigned in phase 3 of the algorithm. Algorithm 3.2 gives the pseudo code of the second phase. Note that at the end of this phase every node knows its sub-tree size as well as the sub-tree size of each of its children nodes.

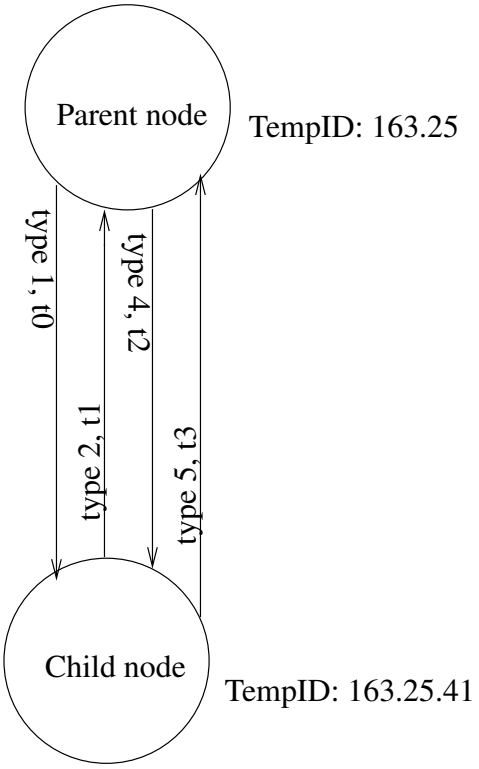

Figure 1: One step of phase 1 with no reinitialization message

3.3 Phase 3: Final Unique ID Assignment

In this phase, the final unique IDs are assigned by each parent node to its children nodes starting from the root. Final IDs are coded using the same number of bytes for all nodes. The initiator is assigned an ID of 0. It sends a final ID message (message of type 8) to each of its children nodes. Each message contains a unique ID and the number of bytes to be used to code IDs. Final ID messages are confirmed with messages of type 9. Each node receiving a message of type 8 takes the ID it contains as its final ID and knows that a number of IDs starting from its ID and containing as many IDs as needed is reserved for the IDs of the nodes in its sub-tree. Each non-leaf node receiving a final ID message confirms it and assigns IDs to its children nodes in a similar way.

Figure 3 illustrates the message exchange during phase 3 to allocate the final IDs. Each node allocates its ID plus 1 to its first child and then allocates ID_i to the i^{th} child with $ID_{i+1} = ID_i + S_i$, where S_i is the sub-tree size of the i^{th} child. Algorithm 3.3 gives the pseudo code of the third phase. At the end of this phase, every node in the network knows its final ID. These final IDs are coded using the minimum number of bytes.

3.4 Collision Handling

Assuming a single channel, if a node n_s is transmitting a message to a node n_r, a collision occurs if n_r is already in the process of receiving from a different node. The algorithm does not assume the

Algorithm 1 Phase 1: Temporary ID Assignment

$ChildB := 1$
$idConfirmed := false$
if $initiator$ is true **then**
  $tempId := 0$
  send an initialization message of type 1
end if
if receive message msg of type 2 **then**
  if a child already have same intId **then**
    send reinitialization message of type 3
  end if
  if no child already has same $intId$ **then**
    add to children list
    choose a random time rt
    schedule checking for confirmation at rt
    send message of type 4 with ChildIdB
    $ChildIdB := ChildIdB + 1$
  end if
end if
if receive msg message of type 5 **then**
  if $msg.dest = tempId$ **then**
    find ch, the corresponding child
    $ch.tempIdConfirmed := true$
    $ch.sizeReceived := false$
  end if
end if
if receive first message msg of type 1 **then**
  $idAssigned := true$
  $tempId := msg.source$
  choose a random 4-byte $intId$
  choose a random time rt
  schedule checking for confirmation at rt
  send message of type 2 with $intId$
end if
if receive msg message of type 3 **then**
  choose a different random 4-byte $intId$
  choose a random time rt
  schedule checking for confirmation at rt
  send message of type 2 with $intId$
end if
if receive msg message of type 4 **then**
  if $idConfirmed$ is true **then**
    update $timeWait$
    resend message of type 5
  end if
  if $idConfirmed$ is not true **then**
    update $tempId$ and $idConfirmed$
    update $timeWait$
    choose a random time rt
    schedule sending message of type 1 at rt
    send message of type 5
  end if
end if

existence of any specific MAC protocol. In particular, no collision avoidance mechanism is required. Collision is handled in the sense that all messages except the initialization message (message of type 1) received by a node are confirmed by an acknowledgment message. Before sending a message, a node chooses randomly an integer number rn between 0 and $RANDMAX$, and waits for a time equal to $(1 + rn \div RANDMAX) \times timeWait$. If it does not receive the confirmation within the random waiting time, it resends the message and keeps doing so until receiving the confirmation.

The node adapts the parameter $timeWait$ to the traffic condition. In fact, this parameter is increased

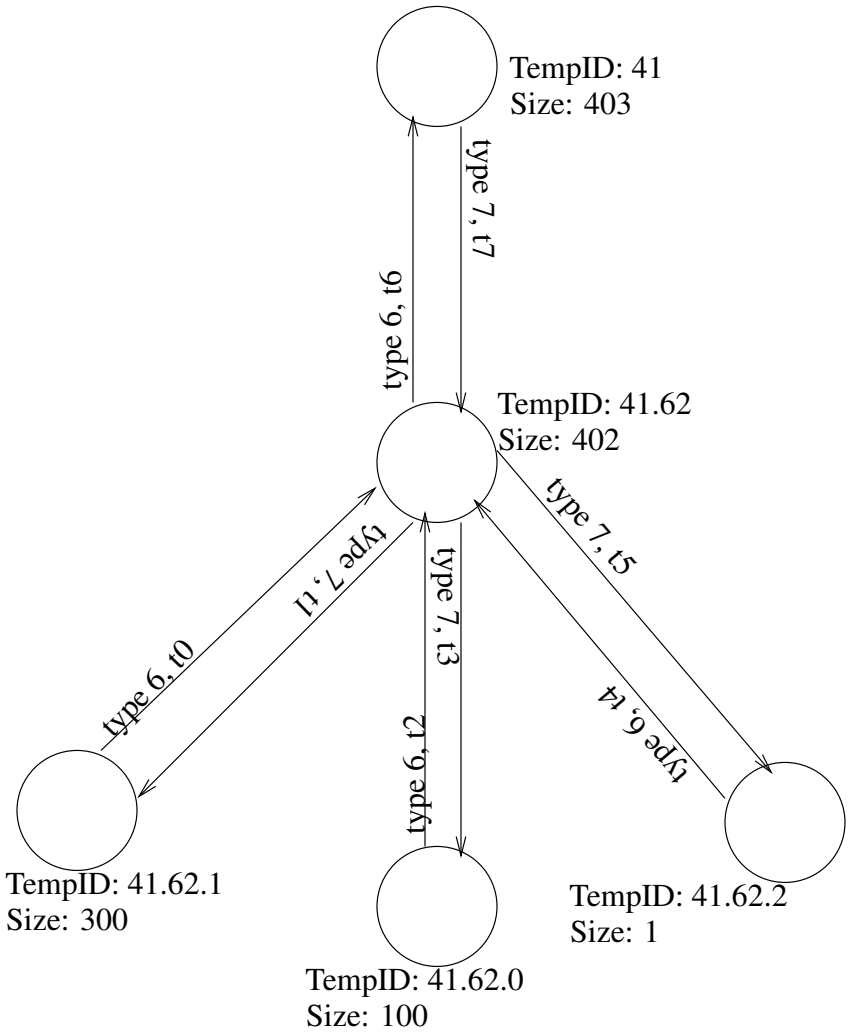

Figure 2: One step of phase 2

by half of its initial value ($timeWaitI$) every time an expected confirmation is not received, unless $timeWait$ has already reached an upper limit set to $5 \times timeWaitI$. Upon the reception of a message, $timeWait$ is reduced by half of $timeWaitI$, unless a lower bound, set to the initial value, is already reached.

For the message of type 1, it is assumed that every node has several neighbors. Each neighbor sends an initialization message at a different time (randomly chosen after the first phase) to reduce the probability of collision. Therefore, a node has several possibilities of receiving an initialization message.

4 Theoretical Analysis

This section contains the theoretical evaluation of the unique ID assignment algorithm. In particular, the correctness of the algorithm is analyzed. We also prove that the algorithm terminates naturally and give an upper limit on the average energy consumption per node. Since the initial assignment messages (of

Algorithm 2 Phase 2: Sub-tree Sizes Collecting

```
subtreeSize := 1
sizeConfirmed := false
if receive msg message of type 6 then
    find ch, the corresponding child
    if ch.sizeReceived is true then
        resend message of type 7
    end if
    if ch.sizeReceived is not true then
        ch.subtreeSize and ch.sizeConfirmed
        subtreeSize := subtreeSize + ch.subtreeSize
        send message of type 7
        choose a random time rt
        schedule checking if all sub-tree sizes received at rt
    end if
end if
if leaf is true then
    choose a random time rt
    schedule checking for confirmation at rt
    send message of type 6
end if
if receive msg message of type 7 then
    if sizeConfirmed is not true then
        update sizeConfirmed
    end if
end if
if sub-tree size messages received from all children and initiator is not true then
    choose a random time rt
    schedule checking for confirmation at rt
    send message of type 6
end if
```

type 1) are sent unreliably, we also analyze the probability of a node being left out by the algorithm. Such a node does not participate in the algorithm and is not assigned an ID.

In this section, we study the case of a synchronized wake-up where all nodes composing the network participate in the initial phase of the algorithm. In particular, the following properties are assumed. The case of asynchronous wake-up is handled in Section 6 later in the paper.

Assumption 1. *All nodes are assumed to be awake during the initial phase of the algorithm.*

This assumption implies that no node is allowed to join the network after the initial phase of the algorithm. A node joining the network after all messages of type 1 have been sent by its neighbors cannot participate in the algorithm and will not receive an ID.

Assumption 2. *All nodes that participate in the initial phase of the algorithm are assumed to remain active and maintain parent-child node connectivity until the termination of the algorithm.*

This assumption is needed to avoid a parent or a child node waiting indefinitely for a message from a node that is no longer active. The primary failure mode we consider in this work is node death due to energy depletion. As is shown later in the theoretical and simulation analyses, the initialization algorithm terminates in a short time, as compared to the average lifetime of a node. Therefore, Assumption 2 is very unlikely to be violated due to nodes dying. Another possible way in which this assumption could be violated is due to drastic changes in environmental (e.g. terrain) conditions that would cause some communication links to be lost. We assume that this situation is fairly rare and handle it in the asynchronous wakeup discussing in Section 6.

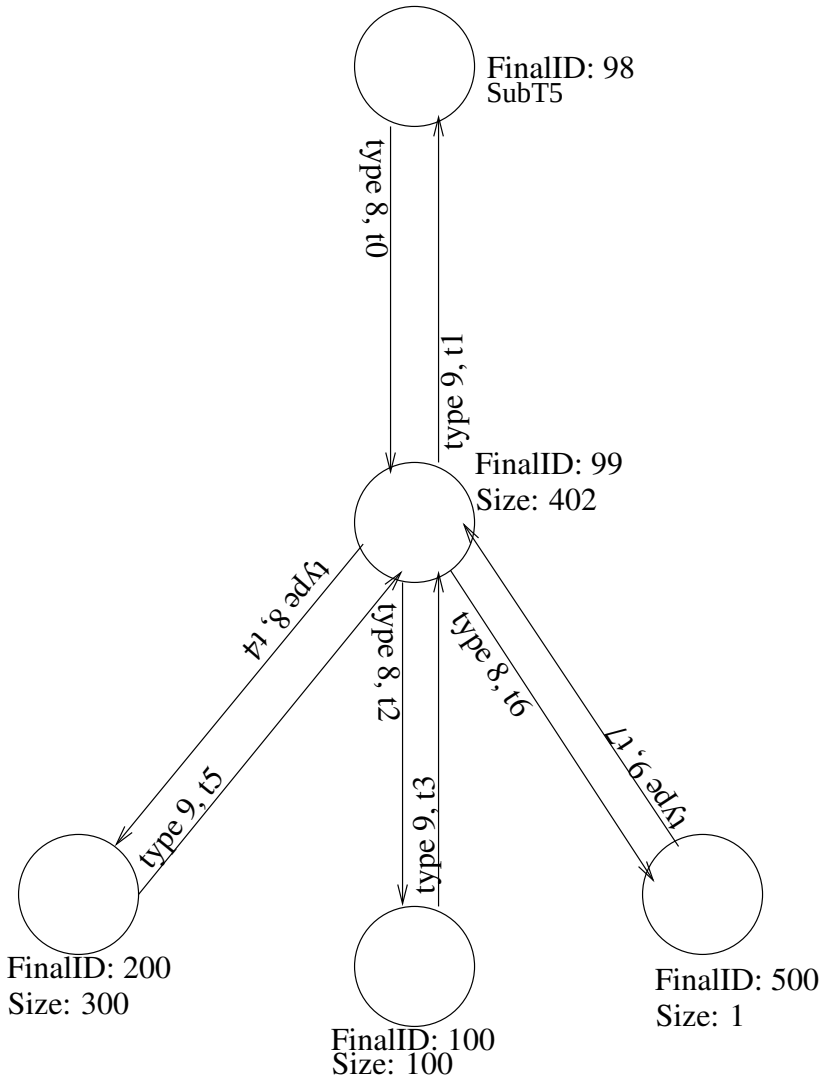

Figure 3: One step of phase 3

The two assumptions above are necessary primarily for the formal proof of correctness that follows in this section. Both of these assumptions are relaxed in Section 6.

4.1 Model

The evolution of each node, except for the initiator, is modeled as a stochastic process with state space of $s = \{0, 1, 2, 3, 4, 5\}$. The different states are defined as follows:

1. State 0: A node is in state 0 if it did not yet receive any initialization message (message of type 1).

2. State 1: A node is in state 1 if it has already received an initialization message, is still waiting for its temporary ID to be confirmed by its parent node.

Algorithm 3 Phase 3: Final IDs Assignment

```
if sub-tree size messages received from all children and initiator is true then
    compute nbBytes, the number of bytes
    myId := 0
    idAssignedF := true
    currentId := 1
    choose random time rt
    schedule checking for confirmation at rt
    send message of type 8 to first child with currentId
end if
if receive msg message of type 9 then
    find ch, the corresponding child
    if ch.idFinalConfirmed is true then
        ignore
    end if
    if ch.idFinalConfirmed is not true then
        ch.idFinalConfirmed := true
        currentId := currentId + ch.subtreeSize
        if more nodes in the children list then
            choose random time rt
            schedule checking for confirmation at rt
            send message of type 8 to next child with currentId
        end if
    end if
end if
if receive msg message of type 8 then
    if idAssignedF is true then
        resend final ID confirmation message of type 9
    end if
    if idAssignedF is not true then
        idAssignedF := true
        myId := msg.minId
        currentId := myId + 1
        send message of type 9 to parent node
        choose random time rt
        schedule checking for confirmation at rt
        send message of type 8 to first child with currentId
    end if
end if
```

3. State 2: A node is in state 2 if its temporary ID has been confirmed by its parent node, but it did not yet send a message of type 1.

4. State 3: A node is in state 3 if its temporary ID has been confirmed by its parent node, it has sent a message of type 1, but did not yet send its sub-tree size message. This could be because it is still waiting to know if it is a leaf, or is still waiting for at least one child node to report the size of its sub-tree. It could also be during the period after receiving all sub-tree sizes, but the scheduled time to send its sub-tree message has not been reached

5. State 4: A node is in state 4 if it has already reported its sub-tree size to its parent node but is still waiting to receive its final ID.

6. State 5: A node is in state 5 if it has already received its final ID.

Clearly, state 5 is a stable state after which the node does not go back to any other state. It is also clear that a node can only go to the next higher state or remain in its current state. That is, for example, a node in state 3 can only go to state 4 or remain in state 3. The probability that a node changes its state depends on its current state as well as the states of the neighboring nodes. In fact, the neighbors influence the

node state in several ways. A non-initiator node currently in state 0 can go to state 1 only if at least one of its neighbors is already in state 2. A non-leaf node in state 3 can change to state 4 only if all of its children nodes (a sub-set of its neighbors) are already in state 4. A non-initiator node currently in state 4 can go to state 5 only if its parent node is already in state 5. More generally the neighbors affect the probability of change in the sense that they can cause collisions if transmitting simultaneously. Collisions cause messages to be retransmitted and delay state changes.

We define the probability $p_i(t)$ as the probability that when the node is in state i at time t, it goes to state $i+1$ in the next step $(t+1)$ with i between 0 and 4. As stated earlier, the value of $p_i(t)$ depends on the current state of the node as well as the current states of its neighbors, in particular its parent and children nodes. Figure 4 gives the state diagram.

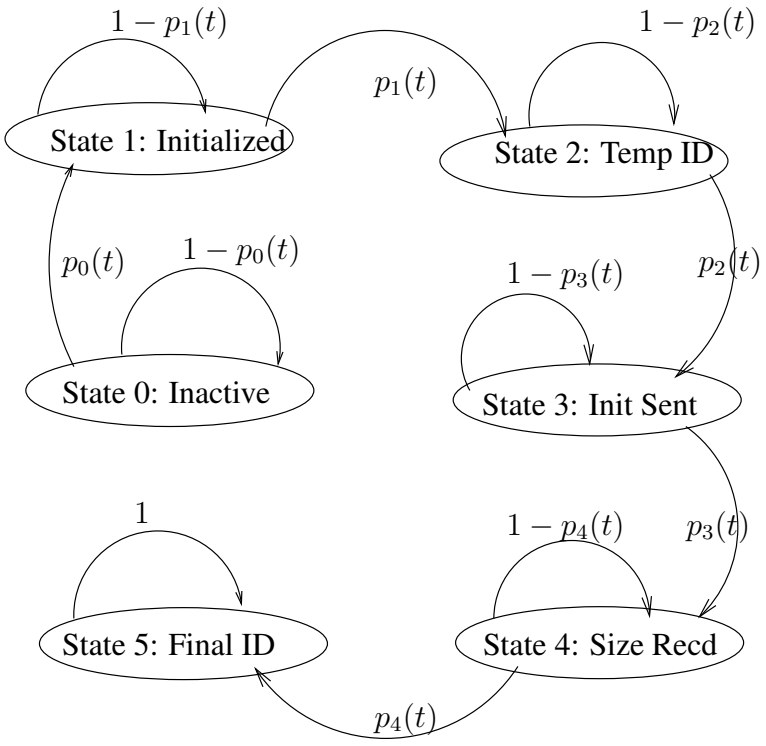

Figure 4: States diagram

4.2 Performance of the Algorithm

In this subsection, several properties of the algorithm are proved. In particular, the termination and correctness of the algorithm are studied. We also study the probability of a node not being assigned an ID at the end of the algorithm. This is a measure of the effectiveness of the algorithm. We also look at the energy cost of the algorithm.

The following lemma shows that if nodes do not die during the execution of the algorithm and remain connected to their parent and children nodes, then the algorithm terminates.

Lemma 1. *Under Assumption 2, the algorithm terminates.*

Proof. The statement in this lemma is equivalent to declaring that any node reaching state 1 in the states diagram will eventually reach state 5 if no node dies during the algorithm execution and parent-child connections are maintained. This is clearly the case when the probabilities $p_1(t)$, $p_2(t)$, $p_3(t)$, and $p_4(t)$ are each greater than 0. Each of these probabilities is permanently equal to 0 only if a neighboring node with which the node interacts in the current state (parent or child node) is no longer alive or can no longer receives messages (e.g., change of terrain or new object between the two nodes causing loss of connectivity). In fact, if such a node dies or is no longer reachable, the dependent node can continue sending a message indefinitely while waiting for a confirmation. It can also indefinitely wait for a size message (in case of parent node) or a final ID message (in a case of a child node). If no node in the network dies or looses connection, each of the probabilities $p_1(t)$, $p_2(t)$, $p_3(t)$, and $p_4(t)$ does not remain equal to 0 all the time. Therefore, each node reaches the final state. The algorithm terminates when all leaf nodes reach the final state (state 5 in the states diagram). □

Before studying the correctness of the algorithm, we look at the possibility of two nodes with the same parent receiving the same temporary ID. This occurs only when two children of the same node choose the same 4-byte ID in first phase and respond simultaneously with messages of type 2 to the initialization message and their parent receives only one of the two messages. For the two nodes to end up with the same temporary ID, they need also to send simultaneously the confirmation message of type 5 and for their parent to receive one and only one of these messages.

We define P_{i2} as the probability of two nodes having two identical temporary IDs. As we can see, P_{i2} is very low because the occurrence of two nodes having two identical temporary IDs is conditioned on the joint occurrence of a successive number of independent events each having a very low probability. In fact, $P_{i2} \leq P_f \times P_s$, where P_f is the probability of any two nodes in the network with the same parent choosing the same 4-byte integer and P_s is the probability of the two nodes sending messages of type 2 during the same time window. For a numerical example, let assume that each node has a radio with a capacity of 100 kbps and the each message of type 2 is no more than 25 bytes. It can easily be proven that in this case, $P_s \leq (25 \times 2^3) \div (100 \times 2^{10}) = 1.95 \times 10^{-3}$. It can also be proven that for a network of n nodes each having no more than $d = 50$ neighbors, we have $P_f \leq (n \times d) \div 2^{32}$. For $n = 10,000$, we obtain $P_f \leq 1.16 \times 10^{-4}$. It follows that in this conservative example, $P_{i2} \leq 2.27 \times 10^{-7}$. In other words, the probability of any two nodes receiving the same ID is less than 1 in a million. The following lemma states the correctness of the algorithm.

Lemma 2. *Under Assumptions 1 and 2, each node receives a unique ID with a high probability of $1 - P_{i2}$, where P_{i2} is as defined above.*

Proof. The proof comes from the nature of the assignment of temporary and final IDs. Since temporary IDs are assigned in a hierarchical way with children of same node all having different IDs, phase 1 ends with every node having a different temporary ID. The only exception is when two nodes receive the same temporary ID. In phase 3, each parent node, starting from the initiator, reserves a different set of final IDs for each of its children nodes having different temporary IDs to assign to its sub-tree. Therefore, every node having a unique temporary ID ends with a unique final ID. This proves the correctness of the algorithm. □

We now determine the probability that a message is successfully transmitted by the first trial and the probability of reception by the second trial. A message is not successfully transmitted if a collision

occurs or a bit error prevents the successful interpretation of the message or the corresponding acknowledgment.

A collision is detected by the sender node, n_s, when it does not receive the corresponding confirmation message in a randomly predetermined time period. As explained in Subsection 3.4, the length of this time period is uniformly distributed between $timeWait$ and $2 \times timeWait$. If no confirmation is received, the message is resent at the end of this period. A collision occurs if the receiving node, n_r, is currently in the process of receiving a different message. It also occurs if a different neighbor of n_r (i.e., in its interference range) broadcasts a message while the current message is being received. If the size of the current message is S bytes and the capacity of the radio is B kbps, the transmission (reception) time of the message is given by: $T_t = 8S \div (1,000 \times B)$. Suppose that the transmission starts at t_0, the current message is not received (collision) if at least one of the other neighbors in the interference range of n_r transmits a message in the time interval $[t_0 - T_t, t_0 + T_t]$. If n_r has k neighbors in its interference range, including the sender n_s, each neighbor transmits at most one message during each period of length tw_0, where tw_0 is the initial value of $timeWait$. Consequently, there are at most $k-1$ messages sent by the other neighbors. Each message is followed by a confirmation message except for a message of type 1 or when confirming a previous message from n_r. Therefore, there are at most $k-1$ confirmation messages and a total of $2 \times (k-1)$ messages. Assuming that all messages have approximately the same size S, the current message encounters a collision if its reception starts in one of at most $2 \times (k-1)$ transmission periods of length $2 \times T_t$.

A message cannot be correctly interpreted if it experienced a bit error, which has the same effect as a collision : the message must be retransmitted. If we consider a bit error rate of BER, it can be easily verified that the probability of a message and corresponding acknowledgment of size $2 \times S$ experiencing a bit error is given by: $P_e = 1 - (1 - BER)^{S \times 8}$.

The following lemma bounds the probabilities that a message is received successively after one or two trials.

Lemma 3. *If tw_0 is such that $P_c = 32 \times (k-1) \times S \div (1,000 \times B \times tw_0) \leq 1$, then:*

- *The probability of a message successfully received upon the first transmission is at least $P_1 \geq 1 - (P_c + P_e)$.*
- *The probability of a message successfully received within two transmissions is at least $P_2 \geq 1 - (P_c + P_e)^2$.*

Proof. This follows from the fact that a collision occurs if the reception of the message starts during one of at most $2 \times (k-1)$ transmission periods each lasting $2 \times T_t$. Since each node sends at most one message in each time interval of length $timeWait \geq tw_0$, we obtain the maximum collision probability: $P_c = 4 \times (k-1) \times T_t \div tw_0 = 4 \times (k-1) \times 8 \times S \div (1,000 \times B \times tw_0) = 32 \times (k-1) \times S \div (1,000 \times B \times tw_0)$. If we consider the events of bit error and collision as independent, the probability of a message not received in the first trial is $P_c + P_e$. Two unsuccessful transmissions must occur for the message not to be received by the second trial. Consequently, the probability of successful transmission by the first trial is given by: $P_1 \geq 1 - (P_c + P_e)$. In the same way, $P_2 \geq 1 - (P_c + P_e)^2$. □

This demonstrates that by appropriately setting tw_0 to limit the collision probability, we can guarantee a high probability of transmission of messages by the second trial. For a numerical example, we assume that the radio transmission rate is $B = 100kbps$, which is reasonable for current technology since

transmission rate for MICAz motes for example is $250kbps$. We also assume that $k = 21$, the network having a density of 21 neighbors in each node's radio range (intereference range). The message size S is function of the number of hops from the base station since each address is composed of one byte per hop. Let assume that at most $S = 100$. This limit holds even for large networks with low densities. Assuming that $BER = 2 \times 10^{-4}$ and $tw_0 = 2seconds$, then we have: $P_1 \geq 85.19\%$ and $P_2 \geq 97.81\%$. These values are conservative lower limits since most messages closer to the root node will be of much smaller size. As can be seen in the simulation results, realistic scenarios give much smaller values. For instance, P_1 is above 92% in all our simulation scenarios.

By following the same reasoning as above, we can bound the probability of a node not assigned an ID at the end of the algorithm. This occurs if the messages of type 1 (initialization messages) from all of its neighbors are lost. Since each of these messages is sent at a random time, we obtain the following lemma.

Lemma 4. *If k neighbors of a node are assigned IDs, then the probability of the node being left out is at most* $P_l \leq 32 \times (k-1) \times S \div (1,000 \times B \times tw_0) + 1 - (1 - BER)^{S \times 8}$.

Proof. The node is left out if all the messages of type 1 sent by its neighbors experience collision or a bit error. The probability of loss of each of these messages is bounded by: $P_c + P_e = 32 \times (k-1) \times S \div (1,000 \times B \times tw_0) + 1 - (1 - BER)^{S \times 8}$. Since the different collision events are independent, the joint probability is equal to the product of the different probabilities. Henceforth, we obtain: $P_l \leq (P_c + P_e)^k \leq P_c + P_e = 32 \times (k-1) \times S \div (1,000 \times B \times tw_0) + 1 - (1 - BER)^{S \times 8}$. □

Again, this probability can be controlled through the parameter tw_0. For a numerical example, if we assume that all the parameters have the same values as in the numerical example given for Lemma 3, then the upper bound on P_l is 14.79%. Again, this a very conservative upper bound that is almost never reached. For instance, in all our simulation scenarios, the percentage of nodes left without ID is always less than 8% as shown in Figures 8, 9 and 10.

A performance measure of the algorithm is the amount of energy consumed per node as a result of executing the algorithm. Since the processing energy is negligible compared to the communication energy, we take into account only the latter. The following lemma bounds the average communication energy consumption per node.

Lemma 5. *If on average each node has* d *neighbors and the average message size in the network is* S_a *bits, then the average communication energy consumption is bounded by* $E_a \leq 15 \times S_a \times (E_t + d \times E_r)$*, where* E_t *and* E_r *are respectively energy consumption per bit for transmission and reception, with probability* $P_e \geq (1 - P_2)^7$*, where* P_2 *is as defined in lemma 3.*

Proof. The proof of this lemma uses the fact that each message is transmitted successfully within two trials with a high probability P_2 as proved in lemma 3. We also assume that the the probability of two children nodes choosing the same integer ID (in Phase 1) is negligible since these IDs are randomly chosen from a large set. In this case every node causes the sending of 8 different messages: messages of all types except the reinitialization message. All of these messages potentially require resending except the initialization message. Therefore, the number of messages per node is less than or equal to 15, with high probability $P_e \geq (1 - P_2)^7$. Since every message is a broadcast, we have every node in the neighborhood receiving the message. A message is, consequently, transmitted by 1 node and received on average by d nodes. Therefore, the average consumption per node is bounded by: $E_a \leq 15 \times S_a \times (E_t + d \times E_r)$. □

5 Simulation Results

In this section, we show the performance of the algorithm under different simulation settings. We study the effect of different parameters on the performance. In particular, we study the effect of the network size, network density, bit error rate (BER) and the initial value of $timeWait$ on the execution time, the percentage of nodes assigned an ID at the end of the algorithm, and the probability of a message being retransmitted.

The simulations were conducted using the Georgia Tech Sensor Network Simulator (*GTSNetS*) [22, 23]. Nodes are distributed in an equi-distant fashion in a square region with the initiator located at the center of the region. The distance between two successive nodes is fixed at 20 meters. The network density is changed by modifying the node's radio range: a radio range of 21 meters for a density of 4, 30 meters for a density of 8 and so on. Messages exchange is performed entirely using broadcasts. Channel sensing is performed before sending a message, which reduces the collision probability. Under each setting, each simulation was run 10 times. An average for these 10 runs is used as the final result. It must be noted that the observed variance from run to run for the 10 runs is very low. For example, for example for the case of a network of 1,000 nodes with a density of 4 and BER of 10^{-4}, the variance on the execution time is 0.0014 (with a mean of 5.81 minutes. The variance on the percentage of nodes assigned an ID at the end of the algorithm is 0.20 (with a mean of 99.7%) and the variance for the retransmission rate is 0.00098 (with a mean of 7.57%). The low variance has been observed for all the simulation scenarios.

Figure 5 plots the execution time of the algorithm as a function of the network size while maintaining a fixed network density of 4 neighbors and a typical sensor node bit error rate of 10^{-4} [24]. By the execution time, we do not mean how long the simulation takes to complete. Rather, the execution is a simulation measure of how long the algorithm would have taken to run on a real sensor network with the simulated configuration. It must be noted here that the algorithm execution time is dominated by the $timeWait$ parameter. In particular, it is clear that the time wait duration ranging from 1.5 to several seconds is more than enough for the node to carry any message processing. The communication time is taken into account by the simulator in the computation of the execution time. The simulation scenarios in this section assume that nodes have a transmission rate of $250kbps$, which is the rate for MICAz sensor nodes. As expected, execution time increases with the network size, but remains relatively short even in the case of long initial $timeWait$ value (less than 30 minutes for a network of 10,000 nodes).

Figure 6 plots the execution time of the algorithm as a function of the network density for a network of 1,000 nodes and a bit error rate of 10^{-4}. We can see that the execution time decreases as the density increases. This is due to the fact that density is increased by increasing the communication range, which reduces the number of hops between the initiator and the leaf nodes. The execution time remains relatively short even for a network of low density.

Figure 7 gives the execution time as a function of the bit error rate for a network of 1,000 nodes and a density of of 4. Compared to the network size and density, the bit error rate has a negligible effect on the execution time.

Figures 5, 6 and 7 show that the execution time increases with the initial value of $timeWait$. This is expected since nodes wait longer before resending lost messages and before forwarding the initialization messages. This makes the overall algorithm take more time to terminate. It is, therefore, desirable to keep the initial value of $timeWait$ as low as possible, within an acceptable ID assignment percentage.

Figure 8 plots the percentage of nodes assigned a unique ID at the end of the algorithm as a function

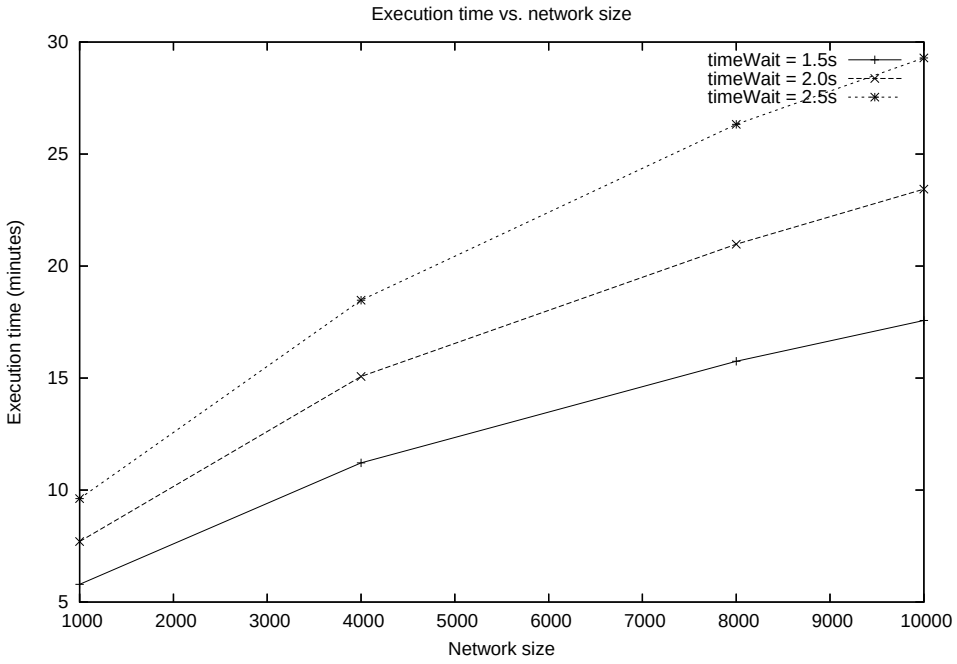

Figure 5: Execution time vs. network size

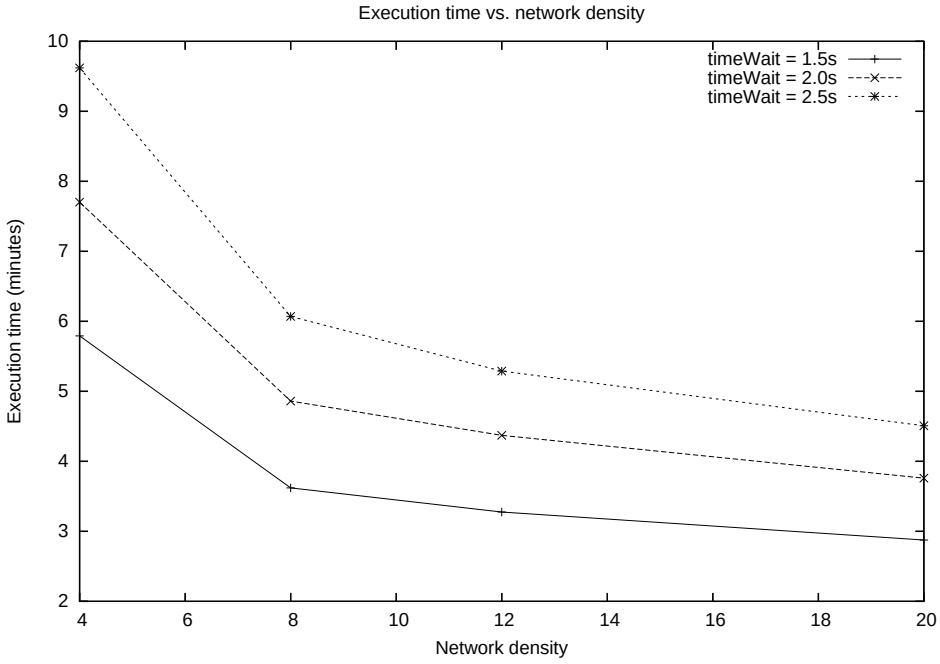

Figure 6: Execution time vs. network density in number of neighbors within a node's radio range

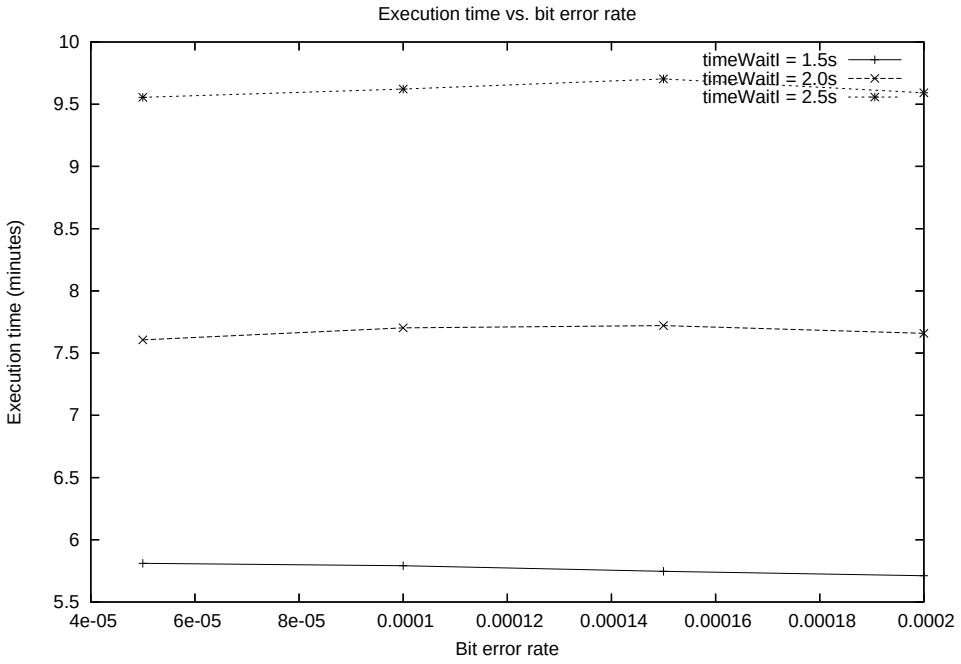

Figure 7: Execution time vs. bit error rate

of the network size with a network density of 4 neighbors and a BER of 10^{-4}. We can see that this probability decreases as the size of the network increases. We can also see that for a specific network size, we can obtain a very high percentage of ID assignments by increasing the initial value of $timeWait$ to a high enough value. However, as this value increases the execution time also increases. With an initial $timeWait$ value of 2.5 seconds, we can obtain a percentage of more than 98% even for a large network of $10,000$ nodes.

Figure 9 plots the percentage of nodes assigned a unique ID at the end of the algorithm as a function of the network density for a network of $1,000$ nodes and a BER of 10^{-4}. We can see that the percentage of nodes with an assigned ID at the end of the algorithm decreases as the density increases. This is due to the fact that higher density increases the probability of collisions, which reduces the probability of successful reception of messages of type 1 even though more messages are sent in each neighborhood. Messages of type 1 are not retransmitted, and their loss reduces the probability of a node participating in the algorithm. However, it can be seen that this reduction can be balanced by increasing the value of the $timeWait$ parameter.

Figure 10 plots the percentage of nodes assigned a unique ID at the end of the algorithm as a function of the bit error rate for a network of $1,000$ nodes and a density of 4 neighbors. We can see that the percentage of nodes with an assigned ID at the end of the algorithm decreases as the BER increases. This is due to the fact that higher BER reduces the probability of successful reception of messages of type 1. Again, the effect of BER is less significant than the effect of network size and density.

Figures 8, 9 and 10 indicate that we can increase the percentage of nodes receiving assigned IDs by increasing the initial value of $timeWait$. However, such an increase causes the execution time to increase, which is not desirable. Thus, there is a tradeoff between the percentage of assigned IDs and

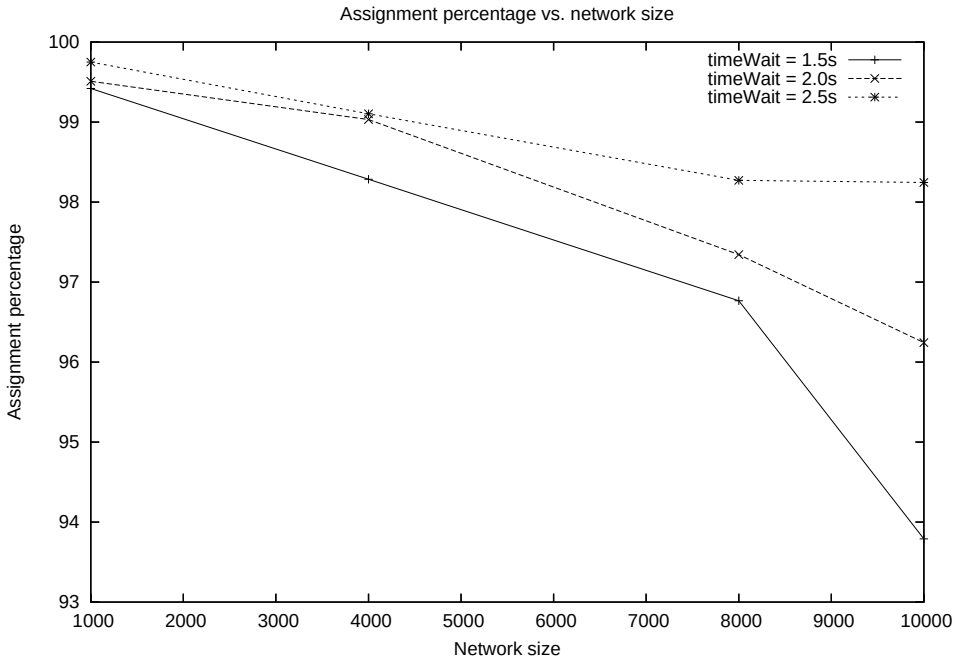

Figure 8: Assignment percentage vs. network size

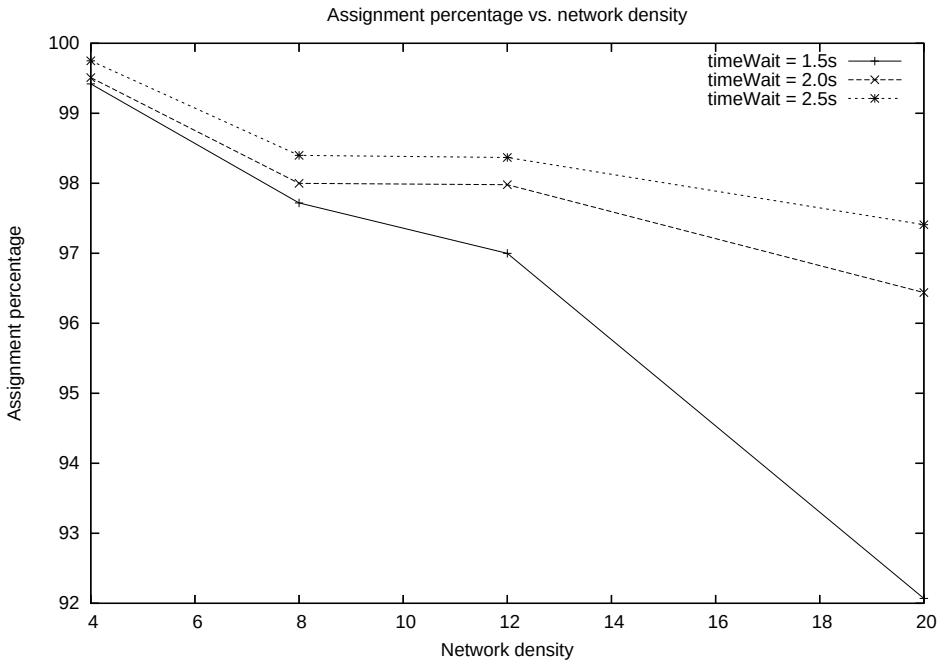

Figure 9: Assignment percentage vs. network density number of neighbors within a node's radio range

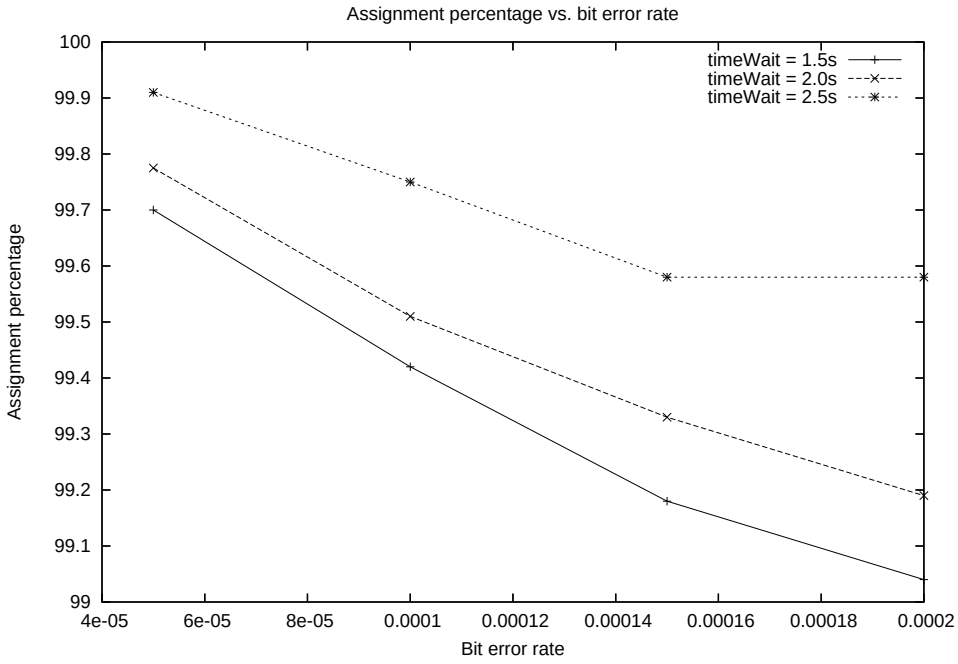

Figure 10: Assignment percentage vs. bit error rate

the execution time.

Finally, we study the percentage of message loss under various simulation settings. Figure 11 plots the percentage of messages being retransmitted because of a loss as a function of the network size. The network density and the BER are fixed, respectively, at 4 and 10^{-4}. We can see that this percentage increases with size. This is due to the fact that messages are longer on average since more nodes are located many hops away from the initiator. Longer messages take more time to transmit, which makes the occurrence of a collision more likely. As expected, the probability of loss diminishes, when the value of $timeWait$ increases.

Figure 12 plots the message loss percentage as a function of network density for a network of 1,000 nodes and a BER of 10^{-4}. We can see that a message loss is more likely in a network with higher density. This is not surprising since more nodes are competing for each channel, which increase the probability of collision. We can also see that the loss probability can be controlled by increasing the value of $timeWait$.

Figure 13 gives the message loss percentage as a function of bit error rate for a network of 1,000 and a density of 4 neighbors. As expected, a message is less likely to be properly received during the first transmission (higher loss percentage) when the value of the BER is higher.

Based on the results, we can state that the initial value of $timeWait$ plays a central role in the algorithm. It needs to be set appropriately so as to maximize the probability of nodes being assigned an ID at the and of the algorithm and minimize the message loss probability while keeping the execution time under control. In a deployed network, $timeWait$ value can be set based on a simulation study such as the one presented herein based on parameters such as network density size, network density and BER.

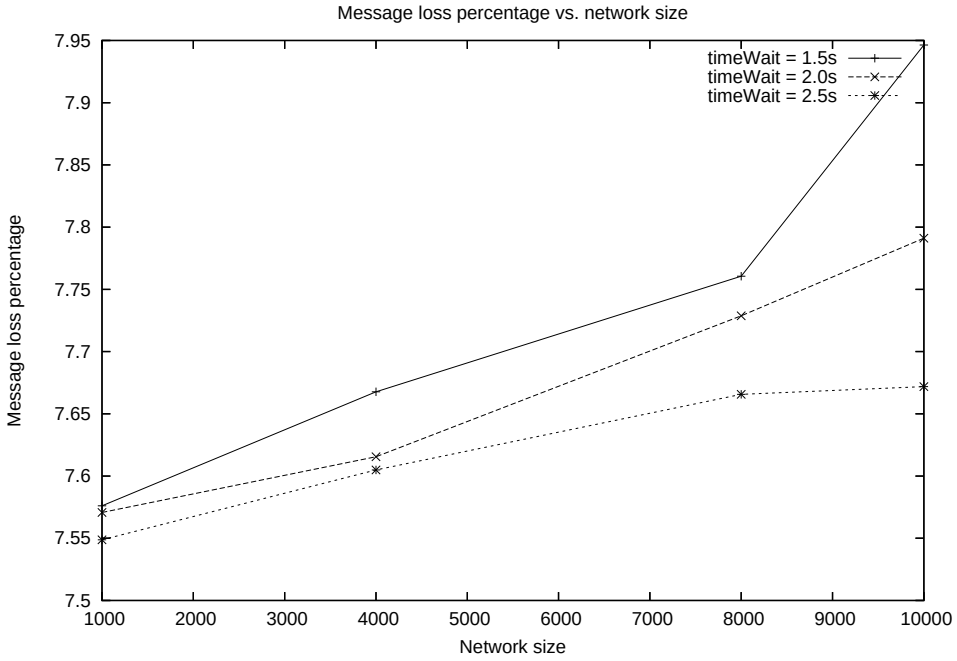

Figure 11: Message loss percentage vs. network size

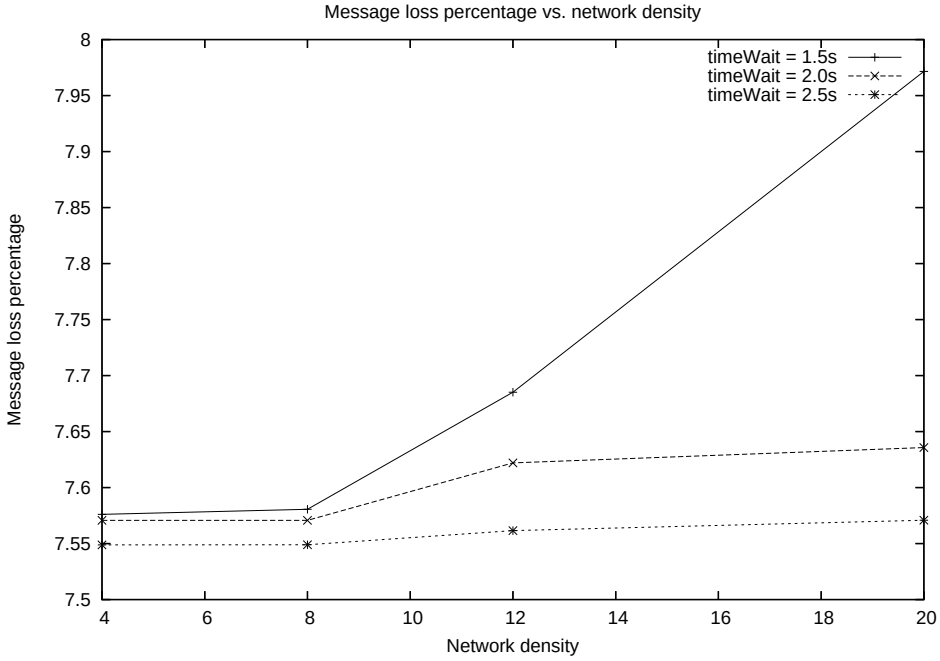

Figure 12: Message loss percentage vs. network density in number of neighbors within a node's radio range

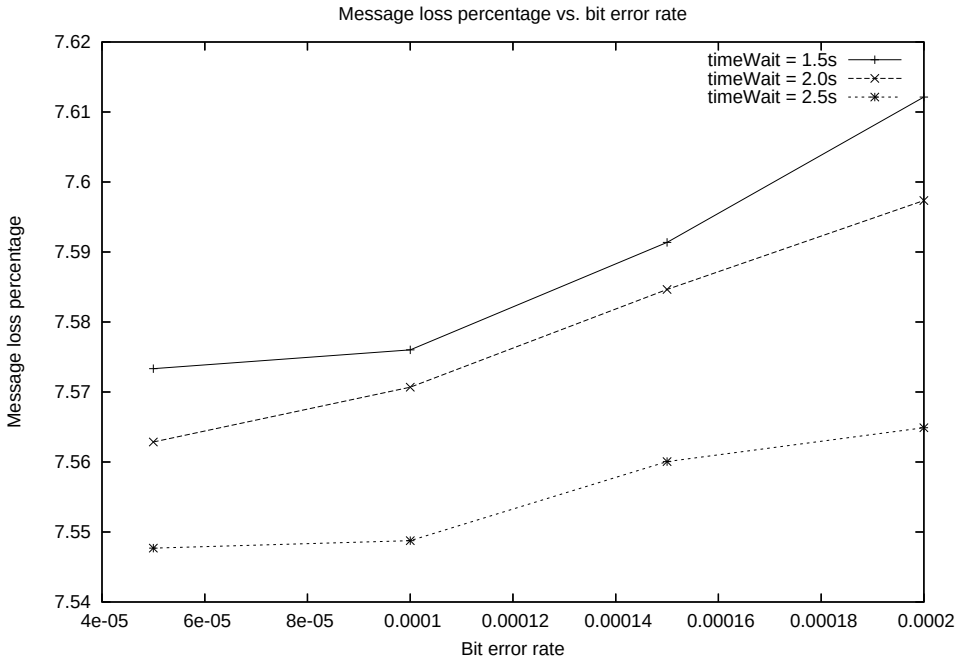

Figure 13: Message loss percentage vs. bit error rate

6 Case of Asynchronous Wake-Up

In this section, we extend the unique ID assignment algorithm to cover the case of asynchronous wake-up of nodes and node connectivity changes during the initial (synchronous) execution of the algorithm.

Nodes are no longer assumed to all be awake at the beginning of the algorithm. This implies that the initiator can no longer determine the exact size of the network when it starts assigning final IDs, since new nodes can wake up later during the initial execution of the algorithm or after its termination.

Nodes initially active execute the algorithm in the same way as in a synchronized deployment, except that Assumption 2 is no longer assumed to hold. This means that initially active nodes do not have to remain active throughout the algorithm execution and communication links can be lost during execution as well. A node that loses its connection with a parent or a child detects such a loss by repeatedly attempting to send a message (e.g., final ID assignment message) and not receiving a confirmation message. When this happens, the node broadcasts a message declaring its loss of connection to a parent or a child. The initiator node keeps track of how many nodes end up not able to receive a final ID during the initial phase. If this number reaches a high threshold (e.g., 5% of all nodes), the initialization phase is restarted. We emphasize that node death due to energy depletion should not occur during the initialization phase and we assume that other types of node failures and drastic environmental changes are relatively rare. Thus, the practical impact of this algorithm modification should be minimal.

To conservatively estimate the final size of the network (including the nodes that wake-up after the initial deployment phase), a new parameter ($nbSpareIds$) is introduced to allow each node participating in the initial deployment to receive a number of spare IDs from its parents. This results in the initiator node receiving size messages that add up to a total size representing the number of nodes initially de-

ployed and their respective spare IDs. The second phase of the algorithm is modified to account for the spare IDs in the computation of sub-tree sizes. Each node initializes its sub-tree size to $1 + nbSpareIds$ instead of 1 and adds the different sub-tree sizes received from its children nodes.

When a node receives its final ID, it knows that it has a number of IDs equal to its sub-tree size to accommodate its needs and the needs of its children nodes. In particular, the node updates its ID and stores a list of spare IDs to use when new neighbors wake-up and request an ID. The spare ID list at each node keeps track of which IDs have been assigned and to which nodes they have been assigned. It must be noted that when a node dies (e.g., runs our of energy), it takes away with its spare IDs that it has not assigned to any of its neighbors. This is done to avoid the cost of centrally tracking unused spare IDs and which nodes hold each. This should not significantly impact the probably of node ID assignmnet since the number of spare IDs is determined conservatively so as to leave enough IDs to cover for such situations.

A new phase is added to assign an ID to a node that joins the network asynchronously. When a node joins the network, it chooses a random 4-byte temporary ID and sends a message requesting a unique ID. Any neighbor that still has one or several spare IDs responds with a spare ID assignment message and temporarily marks the corresponding spare ID as assigned. The requesting node chooses the node from which it received the first spare ID assignment message as its parent and responds with a confirmation message. When the parent node receives the confirmation message, it marks the assigned spare ID as assigned and confirmed (permanently assigned). Any other node that over-hears the confirmation message marks the spare ID initially assigned to the requesting node as available (no longer assigned). If the confirmation message is not heard correctly, the node retransmits its offer of a spare ID several times, before marking the spare ID as available if no confirmation message is received after any of the successive transmissions. This ID can now be used for a new ID assignment request. If a new node does not receive a spare ID assignment message as a response to its request, it resends the request after a random waiting time.

It is possible that a node becomes active before any of its neighbors is assigned an ID. This can happen, for example, if the node joins the network before the termination of the initial round of the algorithm. In this case, the requesting node does not receive a spare ID assignment message. The node waits for a random time and if no initialization message is received (in this case the node acts as an initially active node), it resends its request for a spare ID. The pseudo-code below summarizes the spare ID assignment phase (phase 4).

The number of spare IDs requested by each node that is active at the beginning of the ID assignment algorithm is node-specific. This parameter can vary from node to node depending on the anticipated network deployment patterns. Following are examples of different methods to choose appropriate values of the number spare IDs per node.

1. All nodes request the same number of spare IDs: This simple method can be used when the network is expected to have a uniform density during the initial deployment phase and after taking into account the nodes waking up after the initial execution of the algorithm. In this case, the number of nodes expected to wake-up in the neighborhood of all nodes initially deployed is expected to be roughly a constant.

2. A node requests a number of spare IDs that is proportional to the number of neighbors it has during the initial deployment phase. This method could be used when there are different density levels in various areas of the deployment region. It assumes that a fixed percentage of nodes in all regions

Algorithm 4 Phase 4: Spare ID Assignment

```
if Newly active node then
    idAssignedF := false
    choose tempId, random 4-byte temporary ID
    choose random time rt
    schedule checking for receiving a spare ID assignment message at rt
    send a spare ID assignment request message
end if
if receive msg a spare ID assignment request message then
    if idAssignedF is true and no corresponding assigned spare ID then
        if Available spare ID then
            mark available spare ID as assigned
            choose random time rt
            schedule checking for confirmation at rt
            send a spare ID assignment message
        end if
    end if
    if idAssignedF is true and msg.tempId corresponds to an assigned spare ID then
        choose random time rt
        schedule checking for confirmation at rt
        resend a spare ID assignment message
    end if
end if
if receive msg a spare ID assignment message then
    if idAssignedF is true then
        resend a spare ID confirmation message
    end if
    if idAssignedF is false then
        myId := msg.spareId
        idAssignedF := true
        send a spare ID assignment confirmation message
    end if
end if
if receive msg a spare ID assignment confirmation message then
    if myId == msg.dst then
        mark corresponding spare ID as assigned and confirmed
    end if
end if
```

participate in the initial deployment phase and uses this percentage to predict the expected number of neighbors per node that will ultimately wake-up and request IDs.

3. A node requests a number of spare IDs that is inversely proportional to the number of neighbors it has during the initial deployment phase. This method is appropriate when the final network density is constant throughout the network. This implies that the number of initial neighbors a node has is inversely proportional to the number of neighbors expected to wake-up and request a spare ID.

In the last two cases, a node needs to know the number of active neighbors it has during the initial deployment phase. This number can be easily determined by counting the number of initialization messages sent in its neighborhood. In fact, each node that participates in the algorithm after the initial deployment phase sends exactly one initialization message.

7 Simulation Results for the Asynchronous Wake-Up Case

In this subsection, we use the same network configuration used to study the ID assignment algorithm in the case of synchronous wake-up. The only difference is that nodes may not all be awake at the beginning of the algorithm. Each node is randomly configured to be initially awake or not. The asynchronous wake-up probability, $pAsynchronous$, is the probability of the node not being awake during the initial phase of the algorithm. All the simulations data was collected by averaging values obtained from 10 different runs. Like in Section 5, the variance of the values from the 10 runs is found to be small.

7.1 Effects of the Asynchronous Wake-Up

Here, we try to assess the effect of asynchronous wake-up on the overall performance of the ID assignment algorithm measured by the percentage of nodes that end up being assigned a unique ID. This percentage is measured for networks of various sizes, densities and values of $pAsynchronous$. We also look at the effect of parameter values for the different methods used to compute the number of spare IDs requested by each node participating in the initial phase of the algorithm.

Figure 14 plots the assignment percentage (percentage of nodes assigned a unique ID at the end of the algorithm) for networks of various sizes. The network density was fixed at 20 neighbors per node and the initial value of the $timeWait$ parameter was set to 2.5. The asynchronous wake-up probability is fixed at 60%. For comparison, the assignment percentage is also plotted for the synchronized wake-up case. In both cases, the assignment percentage decreases as the network size increases. The main difference consists of the higher assignment percentage values for the asynchronous wake-up case. This is due to a lower collision rate during the phase of the algorithm involving the nodes initially awake in the case of asynchronous wake-up. In fact, only 40% of the nodes participate in the initial phase of the algorithm, which results in a lower network density and therefore, lower collision rate and higher assignment percentage.

Figure 15 gives the assignment percentage for various network densities. Here, the network consists of $1,000$ nodes with an initial $timeWait$ value of 2.5 seconds. Each node participating in the initial phase of the algorithm requests 4 spare IDs. The data is plotted for two different values of the asynchronous wake-up probability, in addition to the synchronized wake-up case. A threshold effect can be seen from the case of 60% asynchronous wake-up. In fact, for networks of low densities, having only 40% of the nodes initially awake results in networks with very low density during the initial phase. This in turn can result in disconnected networks, which potentially leaves without an assigned ID a large portion of the nodes initially awake and their asynchronous neighbors that try to obtain an ID upon waking-up. A high assignment probability is obtained when the percentage of nodes initially awake is sufficient to maintain a high connectivity as is the case for high density values and/or low asynchronous wake-up probability.

Figure 16 gives the assignment percentage as a function of the percentage of nodes not participating in the initial phase of algorithm. The network size is maintained at $1,000$ nodes and the network density is 20 neighbors per node. Each node requests 4 spare IDs during the initial phase of the algorithm. Again, we can see that the assignment percentage increases as the asynchronous wake-up probability increases for moderate $pAsynchronous$ values. When too few nodes are awake during the initial phase (high $pAsynchronous$ values), the network density becomes too low during the initial phase, which results in high collision rates and low assignment probability.

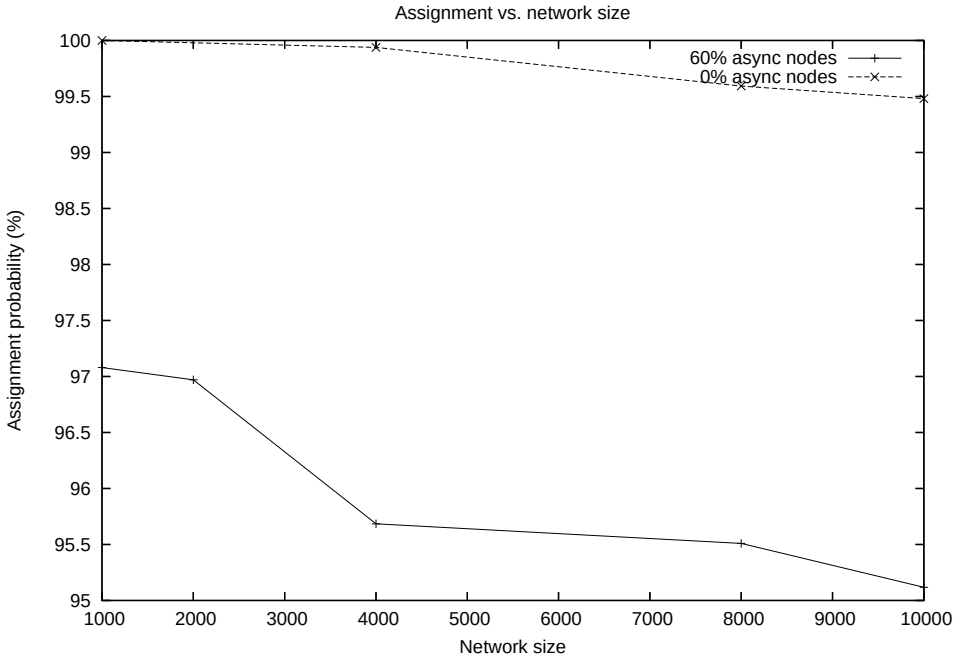

Figure 14: Assignment percentage vs. network size with asynchronous wake-up

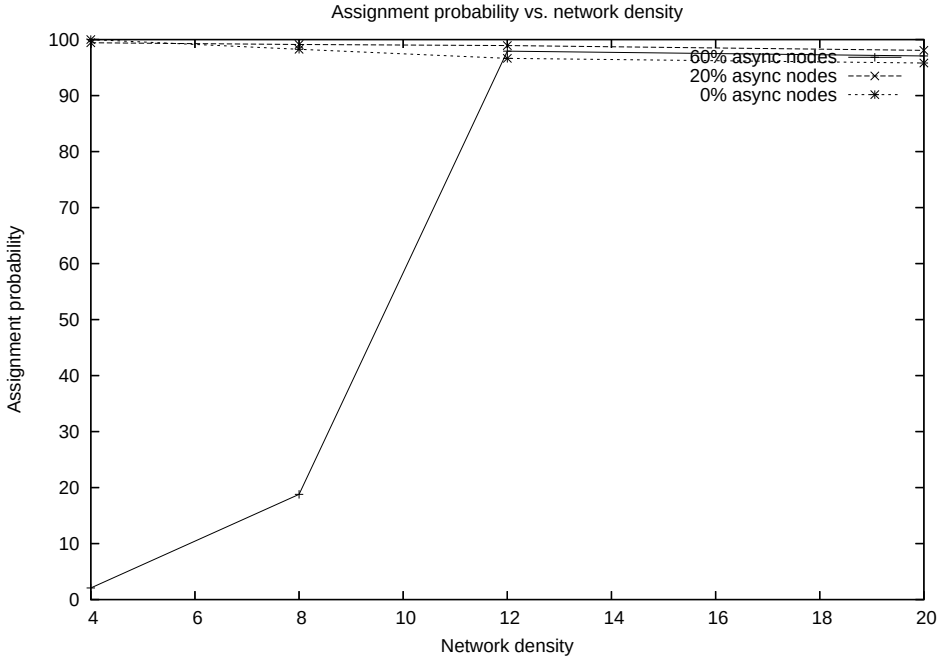

Figure 15: Assignment percentage vs. network density in number of neighbors within a node's radio range with asynchronous wake-up

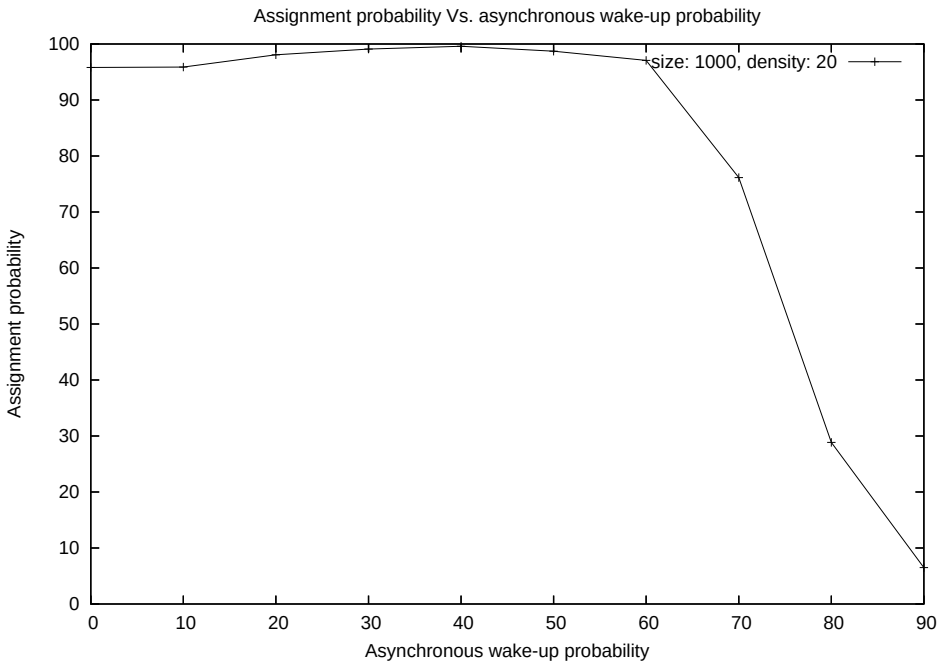

Figure 16: Assignment percentage vs. asynchronous wake-up probability

7.2 Effects of the Number of Spare IDs and Its Method of Allocation

Simulations were also conducted to illustrate the importance of the number of spare IDs that each node initially requests for use when neighboring nodes joining the network after completion of the initial phase of the algorithm. We simulate a network of $1,000$ nodes and a density of 20 neighbors. The network is split into two halves with different values of the $pAsynchrounous$ parameter. In the first half, each node has probability of 40% not participating in the initial phase of the algorithm. This probability is 60% in the second half. Two metrics are of importance here. The most important metric is the percentage of nodes in the network that end up receiving an ID. Another important metric is how efficiently this assignment was accomplished in terms of the ratio between the number of allocated IDs (including spare IDs) and the number of assigned IDs. Of course, the ideal situation is one where all allocated IDs are needed (assigned). This metric is important since the number of allocated IDs determines the ID size needed to code each unique ID.

Table 1 gives the assignment percentage and the ratio of the number of allocated IDs and the number of assigned IDs when using the first method presented in the previous subsection. Here, all nodes request the same number k of spare IDs. The table provides the results as a function of the requested number of spare IDs, k. As expected, the percentage of nodes successfully assigned an ID increases with the value of the coefficient m. However, the ratio between the number of allocated IDs and the number of IDs assigned to nodes increases. This means that the number of wasted IDs (spare IDs that end up not needed for unique identification of nodes asynchronously joining the network) increases with k, which can result in the number of bits required to represent each ID increasing. It is possible that in certain cases, a lower probability of assignment is preferable to a slightly higher probability that requires larger ID size. For example, changing k from 2 to 4 results in an increase of 60% in the ratio between total number of IDs

allocated and the the number of IDs used, while only improving the assignment percentage by less than 5%.

Table 1: Assignment percentage and ratio between allocated and assigned IDs vs. coefficient k when each nodes requests an equal number k of spare IDs

Equal number of spare IDs	Assignment percentage (%)	Ratio of allocated and assigned IDs
1	77.82	1.04
2	95.15	1.29
4	99.65	2.04

Table 2 give the results when using the second method, where each node requests a number of spare IDs that is proportional to the of its neighbors initially active. That is the node determines the number of its neighbors by counting the number of initialization message that it receives. It requests a number of spare IDs that is equal to the number of its neighbors multiplied by a parameter m. The table provides the ID assignment percentage and the ratio between the number of allocated and the number of assigned IDs as a function of the parameter m. Again, the percentage of nodes successfully assigned an ID increases with the value of the multiplicative coefficient corresponding to an increase in the number of requested spare IDs per node. At the same time, the algorithm allocates a higher number of unnecessary IDs as the coefficient m increases.

Table 2: Assignment percentage and ratio between allocated and assigned IDs vs. the coefficient m when each node request a number of spare IDs that is proportional to the number of its neighbors

Coefficient m	Assignment percentage (%)	Ratio of allocated and assigned IDs
0.1	77.6	1.05
0.5	90.37	1.47
1	97.45	2.05
1.5	99.02	3.04

Table 3 illustrates the results when using the third method, where each node requests a number of spare IDs that is inversely proportional to the of its neighbors initially active. In particular, each node requests a number of spare IDs that is equal to the inverse of the number of its neighbors multiplied by a parameter d. The table provides the ID assignment percentage as a function of the coefficient d. Again, the percentage of nodes successfully assigned an ID increases with the value of the number of requested spare IDs per node. As can be seen from the results, this method is ideal for the deployment scenario used here. In fact, since the network density is fixed, it is clear that the number of node neighbors that will wake-up asynchronously and request IDs is inversely proportional to the number of neighbors initially active.

A way of comparing the performance of the three methods of spare ID allocation is to look at the ratio of the number of allocated IDs and the number of assigned IDs for a given percentage of ID assignment. For example, in the previous deployment strategy we can see that the third method has a ratio of only 1.97 for a percentage assignment of more than 99% while the second method gives a ratio of 3.04 for a similar assignment percentage. This means that for this specific deployment strategy the best method is the third method where nodes request a number of spare IDs that is inversely proportional to the

Table 3: Assignment percentage and ratio between allocated and assigned IDs vs. coefficient d when each node requests a number of spare IDs that is inversely proportional to the number of its neighbors

Coefficient d	Assignment percentage (%)	Ratio of allocated and assigned IDs
1	78.57	1.03
5	98.93	1.31
10	99.61	1.97

number of neighbors that are active at the beginning. Indeed, this not surprising since in this case we have two clusters with different values of the probability $pAsynchronous$ of nodes participating in the initial round of the algorithm. As the overall density in both clusters is identical, this indicates that the higher the number of neighbors that are active in the beginning, the smaller the number of nodes that will join the network asynchronously and request an ID. We call this deployment scenario a "clustered activation" since the network is composed of clusters with nodes that have different probabilities of being active during the initial deployment phase. Table 4 summarizes the ratio between the number of allocated and assigned IDs for the three methods in the case of an assignment percentage of at least 99%.

Table 4: Ratio between allocated and assigned IDs for different spare ID determination methods for the clustered activation case with a 99 % assignment

Method	Parameter value (k, m or d)	Ratio of allocated and assigned IDs
First method (equal)	4	2.04
Second method (proportional)	1.5	3.04
Third method (inversely pr)	10	1.97

In contrast with the case of a "clustered activation", a "clustered network" corresponds to the case where different clusters have different densities. We simulate the sensor network described before, except that now all nodes have the same activation probability ($pAsynchronous = 50\%$ but the two clusters have on average different densities. In particular, 10% of the nodes where randomly chosen and removed from the first cluster to be added to the second. The nodes removed from the first cluster are randomly place in the second following a uniform distribution. This situation corresponds to one cluster where nodes have on average a density of 18 neighbors while in the second cluster nodes have an average density of 22 neighbors. The probability of a node being active in the initial round of the ID assignment algorithm is the same in both clusters. Table 5 gives the ratio between the number of allocated and assigned IDs for three methods when the assignment percentage is at least 99%. As can be seen, the natural method for this case where the network is composed of clusters of different densities is the second method where nodes request a number of spare IDs that is proportional to the number of active neighbors in the beginning. This result is not surprising and comes from the fact that the $pAsynchronous$ value is fixed across clusters and, therefore, the total number of a node's neighbors that will join the network asynchronously is, on average, proportional to the number of neighbors participating in the initial round of the algorithm.

Another scenario is the simple "uniform deployment" where regions of the network have the same density and all nodes have the same probability of being initially active. The results for this scenario are presented in Table 6. Not surprisingly, the best method for this scenario is for each node to request an equal number of spare IDs. However, the two other methods still give reasonably low values of

Table 5: Ratio between allocated and assigned IDs for different spare ID determination methods for the clustered network case with a 99 % assignment

Method	Parameter value (k, m or d)	Ratio of allocated and assigned IDs
First method (equal)	4	1.98
Second method (proportional)	1	1.82
Third method (inversely pr)	10	2.06

the ratio between the number of allocated IDs and the number of assigned IDs. In the previous two scenarios (clustered activation and clustered network), there are two regions of the network with different requirements in terms of number of spare ID per active node. This situation resulted in the parameters (e.g., k for the first method) having to be set to the value corresponding to the value that can satisfy the cluster with the highest requirement in terms of number of spare ID. In contrast, in the case of the uniform deployment there is similar average requirement in terms of the number of spare IDs, and this network-wide requirement is lower compared to the clustered activation and the clustered network cases.

Table 6: Ratio between allocated and assigned IDs for different spare ID determination methods for the uniform deployment case with a 99 % assignment

Method	Parameter value (k, m or d)	Ratio of allocated and assigned IDs
First method (equal)	3	1.57
Second method (proportional)	0.8	1.73
Third method (inversely pr)	8	1.69

7.3 Discussion on Setting the Parameter of Spare ID Allocation Methods

The simulation results in the previous subsections demonstrate that our algorithm can assign unique IDs with a good performance level for networks with asynchronous wake-ups as long as enough nodes participate in the initial phase of the algorithm. It is clear that all the methods used to determine the number of spare IDs per active node can result in a good performance as long as the parameter used (e.g., k for the first method) is set properly. In addition, depending on the network topology (e.g., clustered versus uniform) and activation scenario (uniform versus different activation scenarios in different regions), one method or the other is preferable. In the absence of any prior knowledge of the network topology and activation scenario, the first method seems to be the best choice as it generally gives a ratio of the number of allocated IDs to the number of assigned IDs that is close to the one given by the best method.

To correctly set the parameter of the spare ID allocation method, the user can rely on prior knowledge of the network topology and node wake-up patterns. For example, if all that is known is the overall network density in each region and no information is available on nodes' wake-up patterns, a good strategy would be to use the inversely proportional allocation method. A conservative approach for each node can be to request a number of spare IDs that is equal to the total number of its neighbors (known from network density) minus the number of its neighbors participating in the initial round of the algorithm. This information is, of course, obtained by counting the number of initialization messages sent in the node neighborhood. This conservative approach ignores the fact that a node that wakes-up

asynchronously might have several neighbors that participate in the initial round of the algorithm. Only one of these neighbors ends up assigning a spare ID to the asynchronous node. This means that nodes do not need to require the maximum number of spare IDs, instead each can require a number of spare IDs that is equal to the estimated number of nodes that will wake-up asynchronously divided by the number of neighbors that are initially active. For example in the case of the "clustered activation" network used in the previous subsection, we know that the final density in the network is 20 neighbors. The probability of a node being active in the beginning is 40% in one of the first cluster and 60% in the second. This means that on average a node in the first cluster has 12 neighbors that will wake-up asynchronously and request a spare ID, while in the second cluster a node has 8 asynchronous neighbors. The parameter for the third method of spare ID allocation should be set to 12 in one cluster and 8 in the second cluster. This gives a network-wide average parameter of 10 neighbors divided among the nodes already awake in the neighborhood. This number corresponds to the result in Table 4.

Another situation is when the final density of the network is not known in advance, but the probability of a node being active in the beginning of the deployment is known. In this case each node can use the second method to request a number of spare IDs that is equal to the number of its active neighbors multiplied by a coefficient. This coefficient can be determined from the known percentage of nodes that are initially. For example, if it is known that 20% of the nodes will be active during the initial phase of the algorithm ($pAsynchronous = 0.20$), then 4 is the value of the coefficient needed to guarantee that there is enough spare IDs to cover the 80% of the nodes that will wake-up asynchronously. Again the number of active neighbors is obtained by counting the number of initialization messages. In the example presented in Table 5 the network is composed of two clusters having different densities (on average 18 and 22 nodes, respectively). On average a node has a probability of 50% of being active in the begining of the algorithm. This means that a coefficient of 1 is sufficient when using the second method of spare ID allocation.

A less likely case is when both the final density and the number of neighbors initially awake are known. In this case, the user knows before the deployment exactly for each node the maximum number of neighbors that will wake-up asynchronously and request a spare ID. In this case, each node can be set to request this number and no information (e.g., number of active neighbors) is needed after the deployment to determine the number of spare IDs needed per node. This case is similar to the "uniform deployment" presented in the previous subsection. The network has a density of 20 neighbors per node with a probability of 50% of a node being active in the first round of the algorithm. Theoretically, each active node only needs to request 1 spare ID since the number of asynchronous nodes is on average equal to the number of active nodes. However, since the exact number of asynchronous nodes is random it is possible that in a specific neighborhood more than half of the nodes wake-up asynchronously and request a spare ID. In Table 6, we see that a minimum of 3 spare IDs per active node is needed to guarantee a high percentage of ID assignment.

It must be noted here that to avoid excessive maintenance cost, our algorthm does not implement a spare ID maintenance mechanism. That is each node knows whether it currently has spare IDs and how many of them are still available for assignment to new neighbors. However, nodes do not have keep information on how many or which spare IDs are held by their neighbors. This means that if a node dies or goes to sleep, its spare cannot be used for new nodes. This could be a problem if the spare ID estimation is not conservative enough to assign enough spare IDs so that the network can sustain temporary or permenant loss of some.

8 Discussion

As can be seen from the theoretical and simulation results, the proposed ID assignment algorithm has a startup cost in terms of execution time and energy. During the execution of the initial round of the algorithm (for nodes initially active), the network is not being used for its intended final sensing task. For example, in the case of of a network of $10,000$ nodes the algorithm can use upto 30 minutes of the network lifetime. This becomes more problematic when the algorithm has to be restarted several times. Such a situation can occur when not enough spare IDs have been assigned and a large number of nodes waking up asynchronously cannot receive a unique ID. However, the probability of such a situation can be reduced by properly choosing the spare ID allocation method and setting its parameters as discussed in the previous section.

As stated in Lemma 5, the average energy consumption by each node during the initial phase is limited by $E_a \leq 15 \times S_a \times (E_t + d \times E_r)$, where S_a is the average ID assignment message size, d is the network density and E_t and E_r represent transmission and reception energy. It can be easily seen that this limit holds for nodes that join the network asynchronously also. The algorithm cost is compensated by the benefit of having IDs of minimum length that can be used during normal network operations. Having minimum-length IDs reduces the communication energy cost since transmitted packets are smaller, which results in extended lifetime.

Let us assume that the average message length during normal network operation is S_b received by its d immediate neighbors and the total node starting energy that can be used to send and receive messages is E. Then each node can send, in addition to its ID assignment messages, at most m messages over its lifetime where m is such that: $E = E_a + (m \times S_b) \times (E_t + d \times E_r)$. To maximize the node's lifetime is therefore equivalent to maximizing m, which can be achieved by either minimzing E_a or S_b or both, since E is fixed.

Of course, ID assignment algorithms that do not provide globally unique IDs and focus instead on locally unique IDs can minimize both the ID assignment energy, E_a, and the average message size since locally unique IDs require significantly fewer bits. However, the solution presented herein is specifically targeted for cases where globally unique addresses are necessary. We therefore compare it with the solution presented in [14] and random ID assignment, which can both achieve globally unique IDs.

In [14], an algorithm that uses a tree structure similar to the the first phase of our algorithm is presented. From an energy point of view, this solution has an initial energy consumption for ID assignment that is lower than our algorithm. This is because, it does not require the second and third phases used in our algorithm to determine the network size and assign IDs of minimum length. For example, assume a network of $10,000$ and a density such that messages exchanged during the tree-building phase of our algorithm and the algorithm in [14] are of length $S_a = 20$ bytes with an average hop count of 16. In this case, at the end of the tree building, each node has a unique ID of an average length of 25 bytes. In the case of [14], this long ID is the final node ID that will be used throughout network operation. For our algorithm, this long ID is used only during the remaining two phases of network size computation and final ID assignment to ensure reliable message exchange between parent and child nodes. During these two phases, messages still have an average length of 20 bytes or less (ID of length 16 bytes and at most 4 bytes of additional payload). As discussed in Lemma 5, in our algorithm each node sends at most 15 messages during the ID assignment phase of size 20 bytes. In the same way, 5 messages of the same length are sent in [14]. This means that each node transmits up to an additional 200 bytes in execution of our algorithm as compared to the algorithm in [14]. However, nodes end up with a unique

ID of only 2 bytes when using our algorithm while using [14] each node ends up with an ID of 16 bytes, on average. If we assume a message payload of 8 bytes during normal network operation (e.g., each node sending the location coordinates of a tracked object), then the average message size S_b is 24 bytes when using [14] and only 10 bytes when using our algorithm. Thus, once each node has sent more than $200/14 \approx 14.29$ messages during normal operation, the energy consumed in the initialization phase has been recouped. Since most sensor networks are expected to operate for weeks and in some cases much longer, the energy expended during initialization will be almost negligible. Therefore, for the example network just described, the savings achieved during normal operation will typically extend the network lifetime (in terms of number of messages sent) by very close to the 140% difference in packet sizes during normal operation.

An alternative approach is random ID assignment. In this approach, nodes can be deployed and allowed to randomly choose an ID without any need of coordination. Such an approach does not incur a startup cost. However, to avoid the need of coordination between the nodes it is necessary to choose the random IDs in a way that renders the probability of duplicate IDs negligible or provide a protocol to detect and correct duplicate IDs. This can be done by having nodes choose IDs using a large number of bits to be coded [7]. For example, for a network of $10,000$ nodes, each node needs to use IDs of 60 bits (derived in the same way as for the example network of 10^6 nodes discussed in [7]) for the probability of duplicate IDs to be sufficiently small. In this case, the benefit of eliminating the startup cost can be outweighed by the increased communication cost coming from the use of large IDs (about 8 bytes each).

To compare our approach with the randomized ID assignment, it is clear that our approach is suitable when unique IDs are used extensively during the regular network operations, e.g., for message source authentication when security is required or for unicast message exchanges. In the $10,000$ network case if we again assume a payload of 8 bytes, the use of our ID assignment approach results in a total message size of only 10 bytes, while the randomized approach gives a message size of 16 bytes. Assuming the initilization cost is small, as demonstrated above, this means a potential increase of 60% in node lifetime from using our method instead of random IDs.

The randomized approach is more suitable when the unique IDs are used on rare occasions during the normal network operations. For example if the unique IDs are only needed for rare node maintenance messages, then the network can afford the extra communication cost to avoid the initial startup time of our approach. It is also clear that if the message payload is large (e.g., when nodes are transmitting video data), the effect of large IDs in the header becomes less significant, which renders the randomized approach more atractive.

9 Conclusion

We presented a solution to the global ID assignment problem in sensor networks. Our solution aims at assigning unique IDs to each node using the minimum number of bytes required to code these IDs. This was obtained using a 3-phase approach. In the first phase, temporary long IDs are assigned. These temporary IDs are used in the second phase to reliably determine the exact size of the network and, therefore, the minimum number of bytes to use. In the third phase, final IDs coded using the minimum number of bytes are assigned.

The algorithm allows nodes to join the network and obtain unique IDs after the initial deployment phase. This is performed at the local level by allowing each node initially participating in the algorithm

to request several spare IDs to be assigned to neighbors joining after the initial deployment.

We demonstrated that the proposed algorithm can be tailored to obtain excellent results, both in terms of the percentage of participating nodes and the execution time.

References

[1] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks," in *Proceedings of the 6th annual international conference on Mobile computing and networking*, pp. 56–67, August 2000.

[2] D. E. Y. Yu, R. Govindan, "Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks," Tech. Rep. TR-01-0023, UCLA/CSD, 2001.

[3] A. Dunkels, J. Alonso, and T. Voight, "Making tcp/ip viable for wireless sensor networks," in *European Workshop on Wireless Sensor Networks (EWSN)*, 2004.

[4] C. Karlof and D. Wagner, "Secure routing in wireless sensor networks: Attacks and countermeasures," in *IEEE International Workshop on Sensor Network Protocols and Applications*, 2003.

[5] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient mac protocol for wireless sensor networks," in *INFOCOM*, 2002.

[6] A. S. Tanenbaum, *Computer Networks*. Englewood Cliffs, 1989.

[7] J. R. Smith, "Distributing identity," *IEEE Robotics and Automation Magazine, Vol.6, No.1*, March 1999.

[8] C. Schurgers, G. Kulkarni, and M. B. Srivastava, "Distributed on-demand address assignment in wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems, Vol.13, No.10*, October 2002.

[9] F. Kuhn, T. Moscibroda, and R. Wattenhofer, "Initializing newly deployed ad hoc and sensor networks," in *Proceedings of the 10th annual international conference on Mobile computing and networking (MobiCom'04)*, 2004.

[10] E. Ould-Ahmed-Vall, D. M. Blough, G. F. Riley, and B. S. Heck, "Distributed unique global id assignment for sensor networks," in *Proceedings of 2nd IEEE Conference on Mobile Ad-hoc and Sensor Systems (MASS)*, 2005.

[11] D. Estrin, J. Heidemann, and S. Kumar, "Next century challenges: Scalable coordination in sensor networks," in *MOBICOM*, 1999.

[12] M. Ali and Z. A. Uzmi, "An energy efficient node address naming scheme for wireless sensor networks," in *INCC*, 2004.

[13] H. Zhou, M. W. Mutka, and L. M. Ni, "Reactive id assignment for wireless sensor networks," *International Journal of Wireless Information Networks, Vol.13, No.4*, October 2006.

[14] S. PalChaudhuri, S. Du, A. K. Saha, and D. B. Johnson, "Treecast: A stateless addressing and routing architecture for sensor networks," in *Proceedings of the 4th IPDPS International Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN 2004)*, 2004.

[15] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan, "An application-specific protocol architecture for wireless microsensor networks," *IEEE Transactions on Wireless Communications, Vol.1, No.4*, October 2002.

[16] W. B. Heinzelman, J. W. Kulik, and H. Balakrishnan, "Adaptive protocols for information dissemination in wireless sensor networks," in *Proccedings of MOBICOM'99*, 1999.

[17] S. Motegi, K. Yoshihara, and H. Horiuchi, "Implementation and evaluation of on-demand address allocation for event-driven sensor network," in *Proceeding of the Symposium on Applications and the Internet (SAINT'05)*, 2005.

[18] C. E. Perkins and E. M. Royer, "Ad hoc on-demand distance vector routing.," in *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pp. 90–100, Feb 1999.

[19] Y. Liu and L. M. Ni, "Location-aware id assignment in wireless sensor networks," in *Proceedings of the The 3rd IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS'06)*, 2006.

[20] M. Kochhal, L. Schwiebert, and S. Gupta, "Role-based hierarchical self organization for wireless ad hoc sensor networks," in *Proceedings of the Second ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'03)*, 2003.

[21] K. Sohrabi, V. Ailawadhi, J. Gao, and G. Pottie, "Protocols for self organization of a wireless sensor network," *Personal Communication Magazine*, vol. 7, 2000.

[22] E. Ould-Ahmed-Vall, G. F. Riley, B. S. Heck, and D. Reddy, "Simulation of large-scale sensor networks using GTSNetS," in *Proceedings of Eleventh International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'05)*, 2005.

[23] E. Ould-Ahmed-Vall, B. S. Heck, and G. F. Riley, "Simulation of large-scale networked control systems using GTSNetS," in *Springer's Lecture Notes in Control and Information Sciences, Vol. 331* (P. J. Antsaklis and P. T. , eds.), Springer, 2006.

[24] J. Ammer and J. Rabaey, "Low power synchronization for wireless sensor network modems," in *IEEE Wireless Communication and Networking Conference*, 2005.