Distributed Global Identification for Sensor Networks *

ElMoustapha Ould-Ahmed-Vall, Douglas M. Blough, Bonnie S. Heck, 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 analysis of the proposed solution have been preformed. 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. For example, the algorithm terminates in about 5 minutes for a network of 1,000 nodes.

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 nodes is generally limited, and communication is often carried out in a multi-hop way. 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 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 realistic approach because of the coordination it requires and the fact these IDs would have to be very long 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

*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.

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. The algorithm does not assume the preexistence of any type of identification and scales well with the size of the network. We also do not assume the preexistence of any communication protocols. In particular, the preexistence of a specific collision avoidance mechanism is not assumed. The algorithm handles collisions through prevention and recovery. Collisions are prevented 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.

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.

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.

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 [9]. 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 [10].

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 is not realistic in the case of sensor networks. Our solution can be used to assign these global addresses prior to the use of the method in [8].

In [10], Ali, et al., proposed an addressing scheme for cluster-based sensor networks [11]. 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 putting together 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 [12].

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 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.

3 Unique ID Assignment Algorithm

We present a distributed algorithm that assigns globally unique IDs to sensor nodes. Initially, we assume that all nodes remain alive until the end of the algorithm. This assumption is relaxed later in this paper to accommodate a dynamic network where nodes join and leave at any time during the 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.

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. 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 randomly chooses a 4-byte integer and sends it in a message of type 2 to its parent node. This message also contains a retry counter. 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 sent to the child node. If no other child had chosen the same number, the parent node 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 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 $2 \times timeWait$. 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 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.

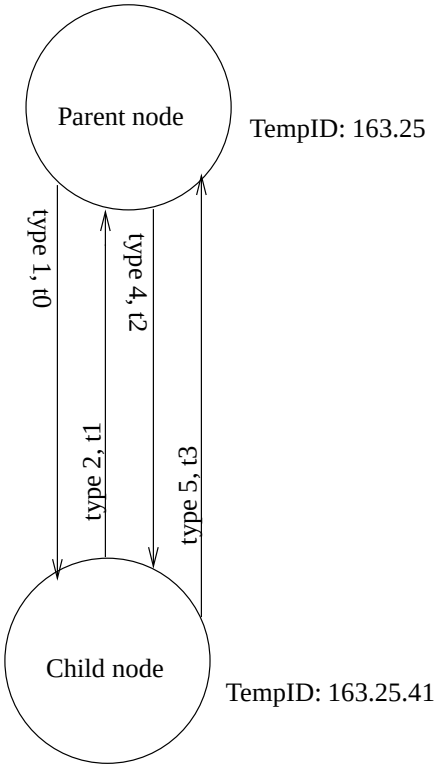

Figure 1: One step of phase 1 with no reinitialization message

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 (i.e., 1, 2, 3, or 4) 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 existence of any specific MAC address. 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 by half of its initial value ($timeWaitI$) every time an expected confirmation is not received, unless $timeWait$ has already reached an upper

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 confi rmation 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
        fi nd ch, the corresponding child
        ch.tempIdConfirmed := true
        ch.sizeReceived := false
    end if
end if
if receive fi rst 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 confi rmation 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 confi rmation 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
```

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 different times (randomly chosen after the first phase). 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 type 1) are sent in a reliably, 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.

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

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

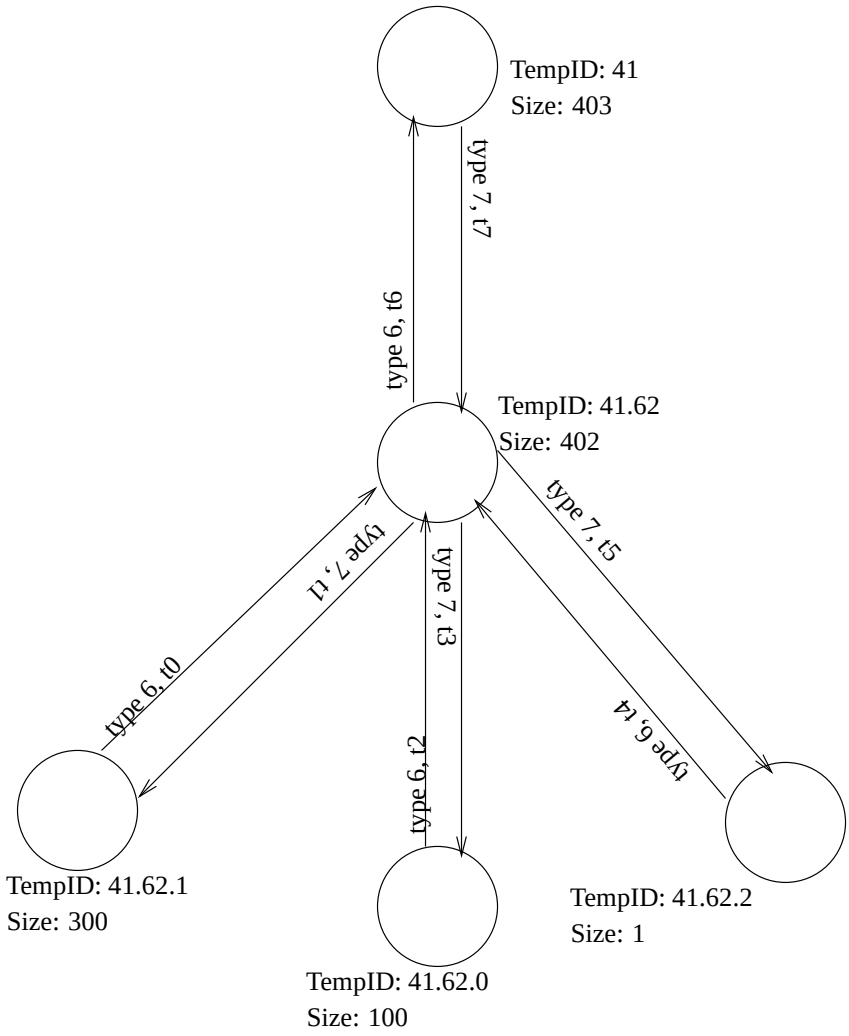

Figure 2: One step of phase 2

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
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

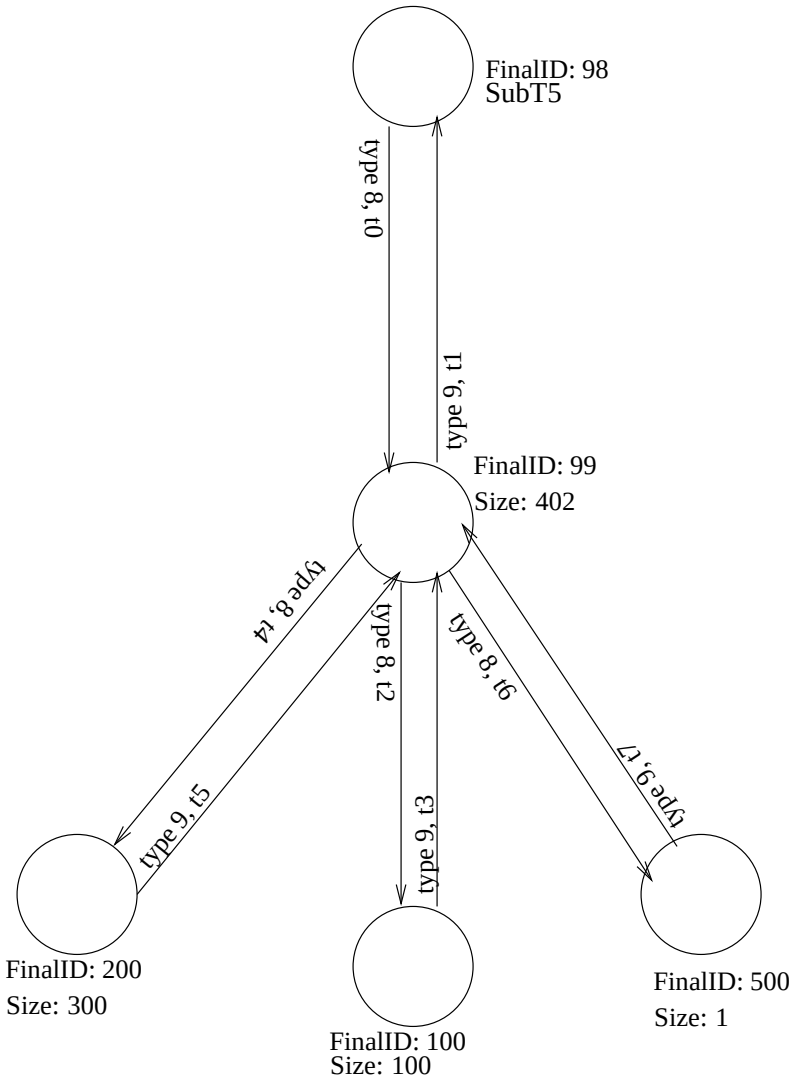

Figure 3: One step of phase 3

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.

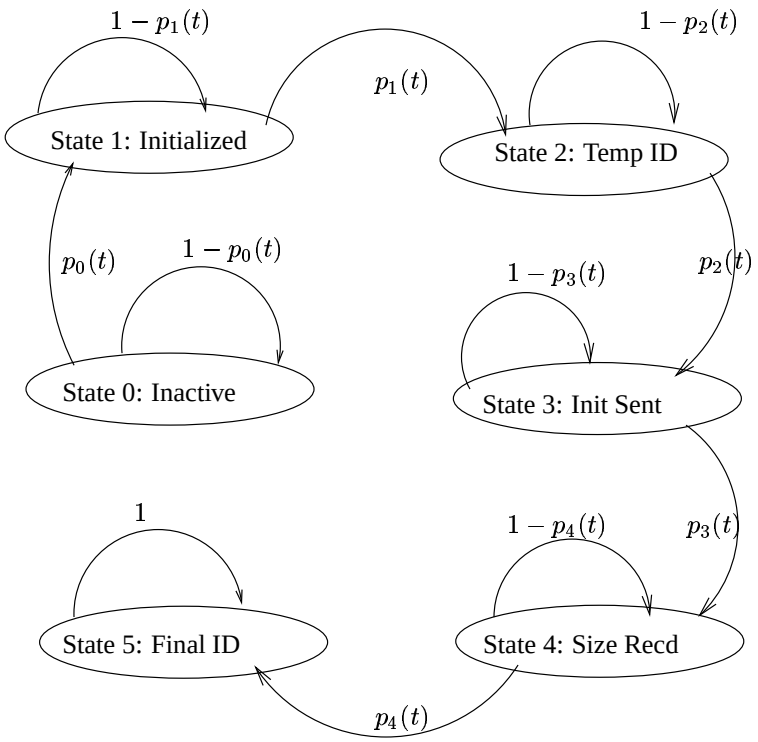

Figure 4: States diagram

4.2 Performance of the Algorithm

In this subsection, several properties of the algorithm are proved. In particular, the correctness and termination of 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.

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 to the occurrence of a successive number of independent events each having a very low probability. In fact, $P_{i2} \leq P_f$, where P_f is the probability of any two nodes in the network with the same parent choosing the same 4-byte integer. It can be proved that for a network of n nodes each having no more than 256 neighbors, we have $P_{i2} \leq P_f \leq (n \times 256) \div 2^32$. For $n = 10,000$, we obtain $P_{i2} \leq 5.96 \times 10^{-4}$. The following lemma states the correctness of the algorithm.

Lemma 1. *If the algorithm terminates, each participating node has 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. □

The following lemma shows that if nodes do not die during the execution of the algorithm, then the algorithm terminates.

Lemma 2. *If all nodes that received an initialization message remain alive, then the algorithm will terminate.*

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. 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. In fact, if such a node dies, 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, 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). □

We now determine the probability that a message is successfully transmitted by the second trial. A message is not successfully transmitted if a collision occurs. 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 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 n_r transmits a message in the time interval $[t_0 - T_t, t_0 + T_t]$. If n_r has k neighbors, 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 $2 \times T_t$. The following lemma bounds the probability that a message is received by the second trial.

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 second transmission is at least $P_2 \geq 1 - {P_c}^2$.*

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: $P_c = 4 \times (k-1) \times T_t \div tw_0 = 4 \times (k-1) \times 8 \times S \div (1,000 \times B \times tw_0) = 32 \times (k-1) \times S \div (1,000 \times B \times tw_0)$. Two collisions must occur for the message not to be received by the second trial. Consequently, the probability of successful transmission by the second trial is given by: $P_2 \geq 1 - {P_c}^2$. □

This demonstrates that by appropriately setting tw_0, 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: transmission rate for MICAz motes for example is 250 kbps. We also assume that $k = 21$, the network having a density of 21. 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. Then we have: $P_c = 0.00064 \div tw_0$. Clearly, we can see that by setting initially $timeWait$ to $tw_0 = 1s$, we obtain the following high probability of successful transmission by the second trial: $P_2 \geq 1 - 0.00064^2 = 1 - 0.41 \times 10^{-6}$.

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 $P_l \leq 32 \times (k-1) \times S \div (1,000 \times B \times tw_0)$.*

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

Again, this probability can be controlled through the parameter tw_0.

A performance measure of the algorithm is the amount of energy consumed per node during the execution of the algorithm. Since the processing energy is negligible compared to the communication energy, we are taking 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, and 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.

Simulations are performed using GTSNetS (the Georgia Tech Sensor Network Simulator) [13], an extension of GTNetS (the Georgia Tech Network Simulator) [14]. 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 transmission range: transmission 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.

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. As expected, the execution time increases with the network size, but remains relatively short (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. 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.

Both Figure 5 and Figure 6 show that the execution time increases with the initial value of $timeWait$. This is not surprising 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.

Figure 7 plots the percentage of nodes assigned a unique ID at the end of the algorithm as a function of the network size with a network density of 4 neighbors. 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 a $timeWait$ initially of 2.5 seconds, we can obtain a percentage of about 99.5% even for a large network of 10,000 nodes.

Figure 8 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. 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.

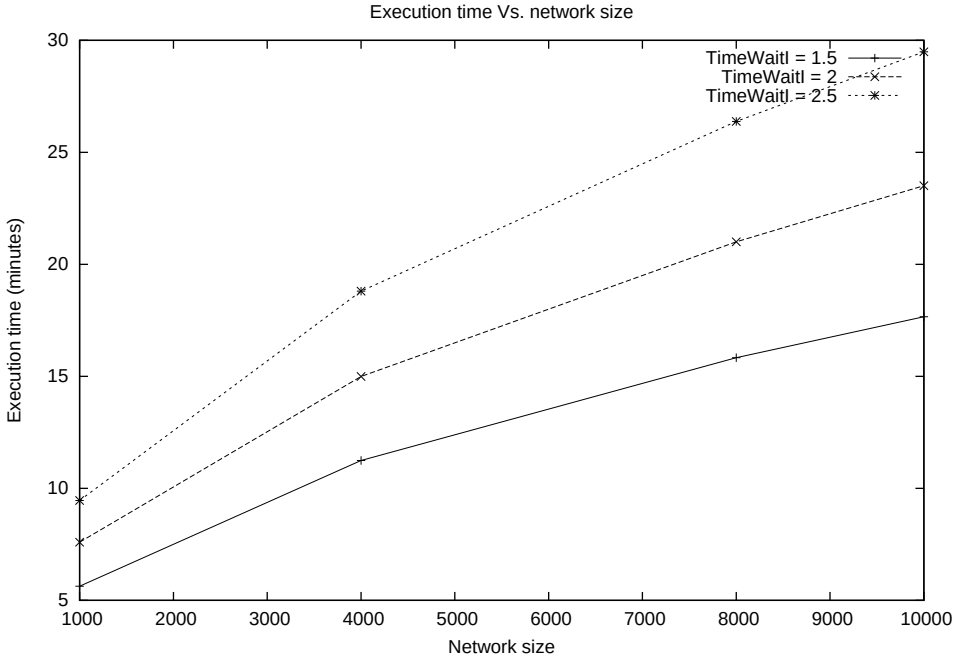

Figure 5: Execution time Vs. network size

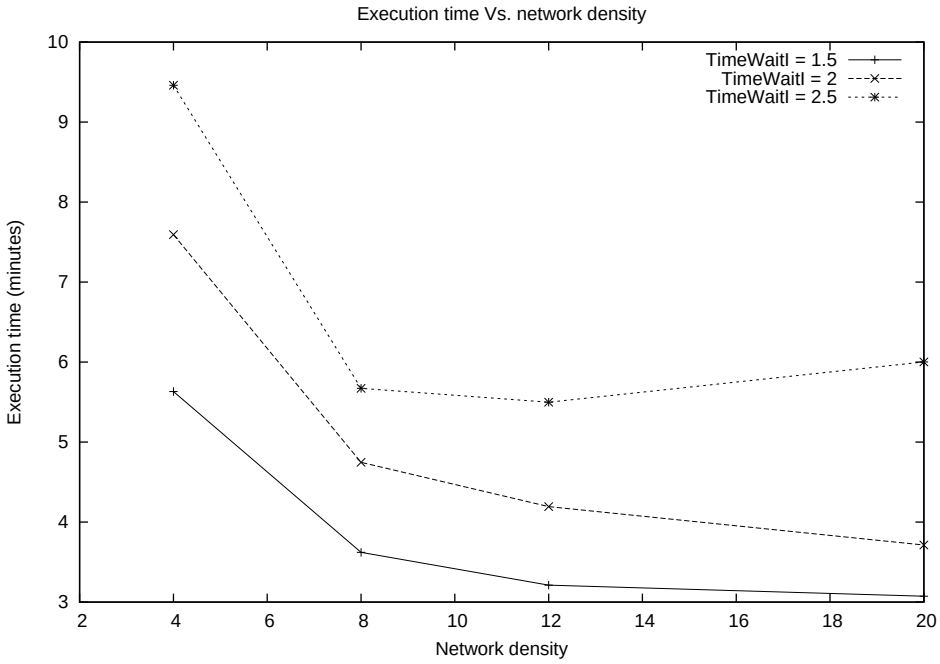

Figure 6: Execution time Vs. network density

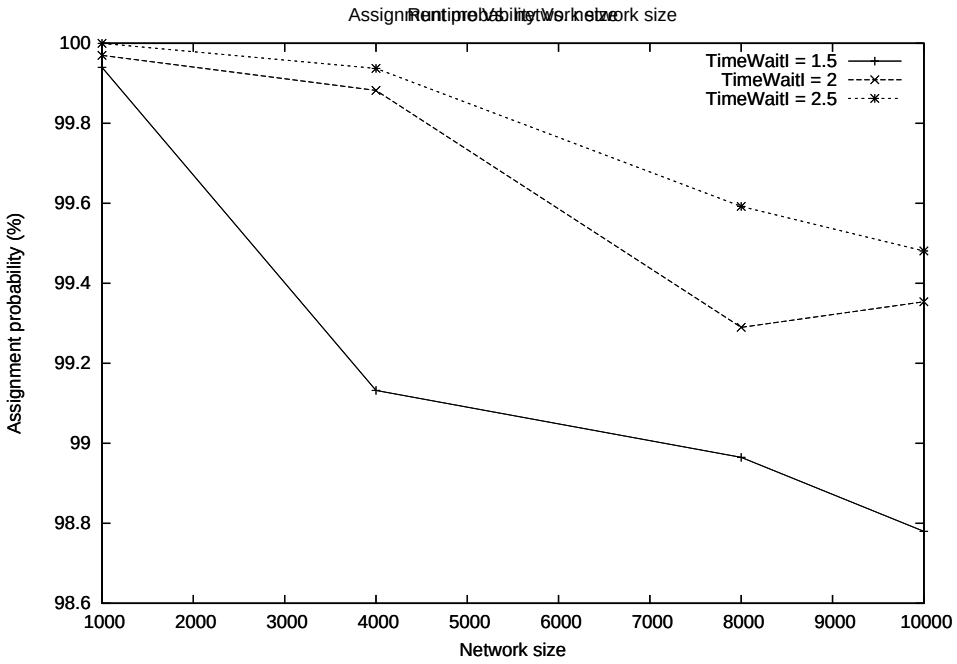

Figure 7: Assignment probability Vs. network size

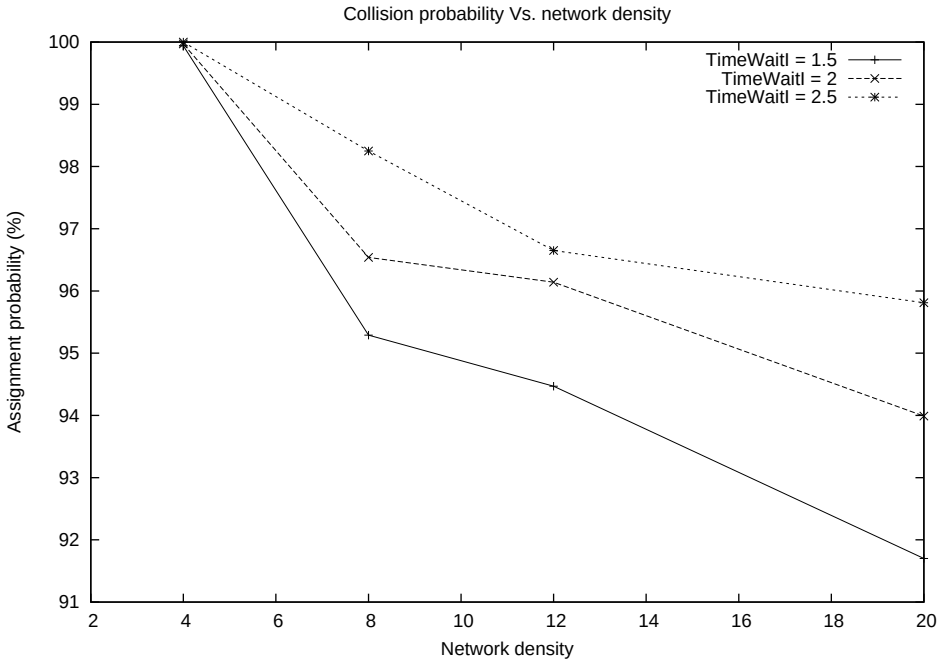

Figure 8: Assignment probability Vs. network density

Both Figure 7 and Figure 8 indicate that we can increase the probability of nodes participation in the algorithm 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 the execution time.

Finally, we study the probability of collisions under various simulation settings. Figure 9 plots the probability of a message being retransmitted because of collision as a function of the network size. We can see that this probability 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 collision diminishes, when the value of $timeWait$ increases.

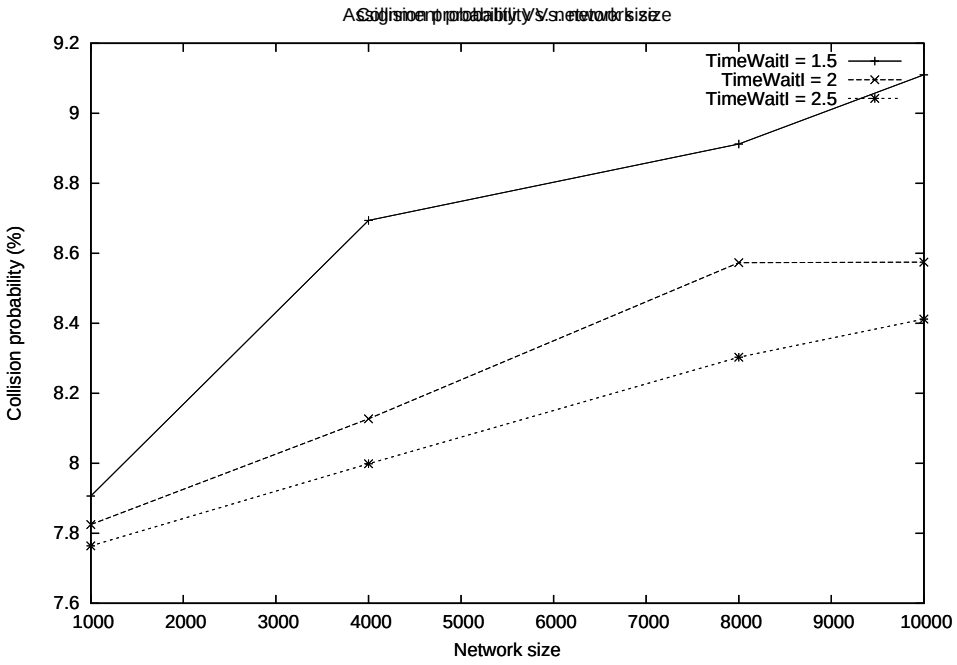

Figure 9: Collision probability Vs. network size

Figure 10 gives the collision probability as a function of network density for a network of 1,000 nodes. We can see that collision is more likely in a network with higher density. This is not surprising since more nodes are competing for each channel. We can also see that the collision probability can be controlled by increasing the value of $timeWait$.

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 collision probability while keeping the execution time under control.

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. Nodes are not assumed to all be awake at the beginning of the algorithm and remain alive until its termination. The initiator no longer knows the exact size of the network when it starts assigning final IDs since new nodes can wake-up later during the execution of the algorithm or after its termination. The initiator now estimates the final size of the network to be k times the size reported from the original nodes. The parameter k is fixed prior to the start of the algorithm and is a function of the wake-up procedure of the network. For example, if we estimated that half of the network is awake at the start of the algorithm, then k needs to be at least 2. There are k unique IDs for each node participating in the initial round of the algorithm. The additional IDs are kept at the initiator and assigned later to new nodes.

There are several modifications to the algorithm. Nodes no longer need to keep retransmitting lost messages indefinitely until their confirmation. In fact, a node considers its neighbor dead (or gone in sleep-mode) if it does not confirm a message

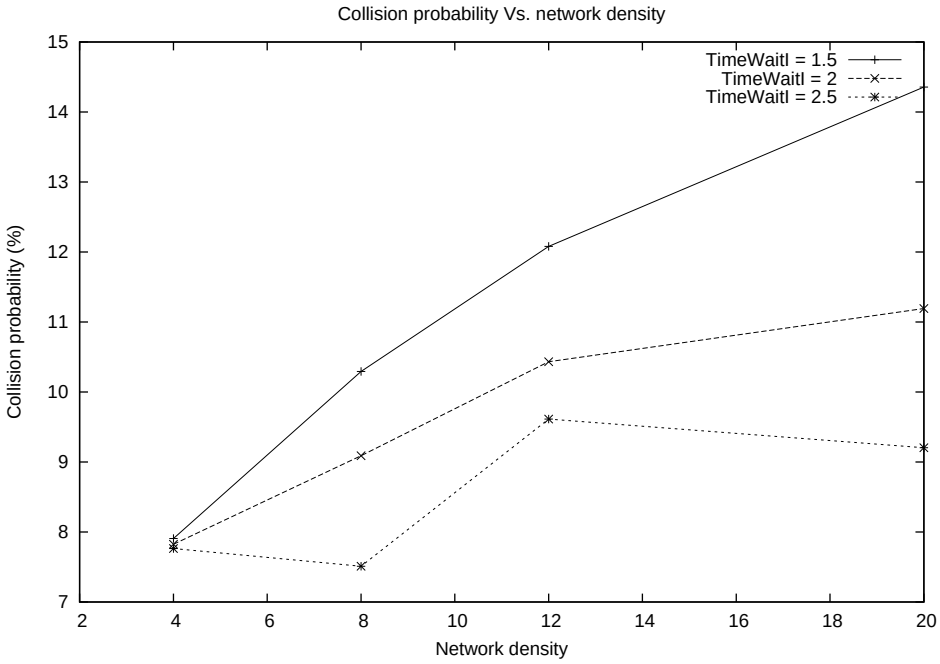

Figure 10: Collision probability Vs. network density

after two trials. If such a neighbor is the parent node, then the node tries to attach itself to a different node by acting as a node that just joined (woke-up) the network. If the dead node is a child, then the node deletes it from its children list. The child has to find a parent when it rejoins the network.

When a parent node dies, each one of its children sends a parent request (message of type 10). The first neighbor to respond with a message of type 11 is chosen as the new parent. A parent selection message (type 12) is then sent. The reception of this message is confirmed by the chosen parent through a message of type 13.

A node that wakes up after the start of the algorithm sends an initialization request (message of type 14). Nodes receiving this message resend their initialization message. Any other new node (waking-up at the same time) will respond as before with a message of type 2. Any node that obtains a new sub-tree requests to be allocated a set of IDs equal to the size of the additional sub-tree. This request is performed using a message of type 15 going to the original initiator node. Each node along the path confirms this message using a message of type 16. When the initiator receives this message, it sends a second allocation message of type 17. This message is confirmed at every intermediate node using a message of type 18.

7 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. 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 *MOBICOM*, 2000.

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

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

[11] 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.

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

[13] E. Ould-Ahmed-Vall, G. F. Riley, and B. S. Heck, "Gtsnets: the georgia tech sensor network simulator," in *International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM), Poster Paper*, 2004.

[14] G. F. Riley, "Large-sacle network simulation with gtnets," in *Winter Simulation Conference*, 2003.