SUMMARY This paper presents an efficient time slot assignment algorithm for a wireless sensor and actor network (WSAN), which consists of stationary sensors for detecting events and mobile actors for performing tasks. TDMA protocols are suitable for WSAN due to time-critical tasks, which are assigned to actors. In order to improve the performance of TDMA protocol, a time slot assignment algorithm should generate not only efficient TDMA scheduling but also reduce periodic run-time overhead. The proposed algorithm offers $O(\delta^2)$ run-time in the worst case, where $\delta$ is the maximum number of one-hop and two-hop neighbors in the network. The average run-time in simulation results is far less than $O(\delta^2)$; however, while the maximum number of assigned slots is bounded by $O(\delta)$. In order to reduce the run-time further, we introduce two fundamental processes in the distributed slot assignment and design the algorithm to optimize these processes. We also present an analysis and verify it using ns-2 simulations. Although the algorithm requires time synchronization and prior knowledge of two-hop neighbors, simulation results show that it reduces the run-time significantly and has good scalability in dense networks.

key words: wireless sensor and actor networks, wireless sensor networks, medium access control, TDMA protocols

1. Introduction

A wireless sensor and actor network (WSAN) [1] consists of stationary sensors and mobile actors connected by wireless communication. Sensors gather environmental information by observing the target region and detecting given events; the information is aggregated at actors (or sink nodes). Actors process the collected information and perform appropriate actions. This configuration involving the cooperation of sensors and actors has lead to the realization of automatic systems for a broad range of applications [2], including intelligent military systems, environmental monitoring and control systems, and intrusion detection and tracking systems.

In a WSAN, it is critical that the sensors inform each detected event to the actors as soon as possible; otherwise, the actors may face an uncontrollable situation and, thereby, fail to perform the effective action (i.e. fire extinguishment). In order to address such time-critical tasks properly, the WSAN requires a MAC protocol to provide real-time service within a delay bound. In a network where multiple sensors detect the event, a contention-based protocol is unable to guarantee the delay bound due to high contentions and severe collisions between nodes. Some contention-based MAC protocols can reduce the average delay bound by lessening collisions. For example, CSMA/p [3] presents an optimal CSMA solution with a nonuniform probability distribution $p^*$ that minimizes collisions between contending nodes. However, CSMA/p [3] assumes that all contending nodes are tightly synchronized for channel access with accuracy of a single contention window slot; in practice, CSMA/p [3] is sub-optimal. On the other hand, a TDMA protocol is a practically good solution for guaranteeing the delay bound. However, in order to maximize the performance of TDMA, two obstacles must be addressed: maintaining the time synchrony between nodes and assigning time slots to nodes.

Time synchronization is divided into global time synchronization and local time synchronization. Global time synchronization synchronizes all nodes to the reference node and requires network-wide flooding. Local time synchronization allows pair-wise nodes to synchronize to each other in a distributed manner with exchange of time synchronization messages. A TDMA protocol requires that each node synchronizes to other nodes within two hops. Maintaining time synchrony can be achieved by a combination of global and local time synchronization. TPSN [4] presented some experimental results for global time synchronization in a multi-hop network. TPSN solves global time synchrony by performing time synchronization periodically according to a bound. Given that the clock error between nodes in two hops should be bounded by 5 ms, the re-synchronization period $t_{\text{resync}}$ [4] is computed as follows:

$$5 \text{ms} = 60 \mu s + 4.75 \mu s/s \cdot v_{\text{resync}}, \ t_{\text{resync}} = 17 \text{min}.$$  

That is, periodic global time synchronization can satisfy the requirement of TDMA protocols with small overhead and local time synchronization can prolong the re-synchronization period.

Next, consider the TSA (Time Slot Assignment) problem. This problem entails finding the time slot for each node so that no two nodes that are in one hop or two hops share the same time slot. The performance of an TSA algorithm is determined by two metrics: the number of assigned slots and the run-time. The number of assigned slots is the largest slot number among assigned slots and the run-time is time taken for a node to reserve its slot. The problem of assigning time slots to each node with the minimum number of slots is equivalent to the distance-2 coloring problem, which

Copyright © 2008 The Institute of Electronics, Information and Communication Engineers
is known to be NP-complete [5]. However, the number of assigned slots should be kept as small as possible so that the algorithm provides a more efficient schedule to the TDMA protocol. Because topology change requires reassignment of time slots, the run-time taken by an algorithm is also an important metric for WSAN.

While the centralized solutions in early works are not suitable for WSAN, the distributed schemes in recent works provide us new insight into designing an efficient TSA. Although it is not directly applicable to WSAN, RAND [6] guarantees that the performance in the number of assigned slots is bounded by $\delta + 1$, where $\delta$ is the maximum size of two-hop neighborhood in the network. DRAND [7] proposes a distributed version of RAND and shows good performance with respect to the number of assigned slots, which is bounded by $\delta + 1$. However, DRAND is not optimized for the run-time and, thereby, is not efficient in WSANs. TRAMA [8] achieves good performance in the run-time by using a distributed hashing technique, which is proposed in NAMA [9]. However, the performance for the number of assigned slots is $O(n)$ in the worst case, where $n$ is the number of nodes in the network. Therefore, we develop a new distributed TSA scheme, called HRAND (Hashing-based RAND). HRAND is designed to generate efficient scheduling and reduce periodic run-time overhead for WSANs. In order to reduce the run-time, two fundamental processes should be considered in distributed TSA schemes: node election process and time slot reservation process; these are fully explained in Sect. 3. We propose efficient and scalable methods for these processes. Although HRAND requires time synchronization and prior knowledge of two-hop neighbors, it shows good performance in terms of the run-time and the number of assigned slots.

2. Related Work

The recent schemes for TSA can be categorized into random slot-based and minimum slot-based schemes, based on the strategy of slot selection. Random slot-based schemes [8]–[13], where each node selects a slot randomly, focus mainly on improving the run-time. TRAMA [8] and NAMA [9] use a distributed hash function to determine which node accesses the wireless channel in each time slot. TRAMA and NAMA perform TSA without exchange of control messages, but provide poor performance in the number of assigned slots. Given that $n$ nodes in a tandem network have a priority chaining such as $prior(1) < prior(2) < \ldots < prior(n)$, only node $n$ accesses the medium. Hence, the performance is $O(n)$. Moreover, it is possible that a node does not transmit a packet within a delay bound due to randomness in node election. SEEDEX [10] uses a similar hashing function to compute the states of neighbors and allows a node to transmit only if the collision probability is lower than a threshold. In contrast with TRAMA and NAMA, SEEDEX does not guarantee the collision-free access, and nodes may consequently suffer from collisions. In Parthasarathy’s scheme [11], nodes select time slots randomly among $[1, c \delta]$ through the four-phase protocol ($c$: a constant). The random selection reduces the time used in determining conflict-free slots, but leads to an increase in the average number of assigned slots. In FPRP [12] and E-TDMA [13], a node contends with neighbors for a slot and selects a slot randomly using the five-phase protocol. It is possible that two or more nodes share the same slot in the worst case.

The main concern of minimum slot-based schemes [6], [7], [14], [15] is performance in the number of assigned slots. In this regard, Ramanathan [6] proposes three heuristics: RAND, MNF, and PMNF. These heuristics guarantee that performance in the number of assigned slot is bounded by $O(\delta)$. Because the heuristics require a global topology, they are not directly applicable to WSANs. In TRAMA [7], a node requests available slots from its neighbors and selects the smallest conflict-free slot. The performance in the number of assigned slots is $O(\delta)$, but DRAND is not optimized for run-time. In MASA [14], a mobile agent randomly moves across uncolored nodes, gathers neighbor’s information, and assigns the smallest conflict-free slot to each node. Because there is only one mobile agent in the network, MASA may show poor run-time performance and have lack of scalability to the network size.

3. Overview of Distributed TSA

In this section, we describe two fundamental processes that constitute the implementation of distributed minimum slot-based TSA. Each node in the network repeatedly performs two processes in sequence until it is assigned a slot: node election process and time slot reservation process.

The node election process determines whether a node can access the channel during a given round. A straightforward way to implement this process is that each node accesses the channel with probability $p$. Given that each of the $N$ nodes within two hops is willing to access the channel with probability $p$, the probability that there is exactly one elected node is $Np(1 - p)^{N-1}$, which is maximized when $p = \frac{1}{N}$. This method is simple and low-cost in terms of implementation, but there occur collisions between elected nodes. Another way to implement the election process is to use distributed hashing [9]. Distributed hashing provides collision-free channel accesses to nodes under time synchronization. Because the node with the highest priority is allowed to access the channel, the probability for a channel access is equal to $\frac{1}{N}$.

We consider that a node with an assigned slot no longer needs to access the channel. As the number of completed nodes increases, the initial value of $p$ becomes further from the optimal value. It is necessary to set the value of $p$ accordingly. Maintaining the proper value of $p$ requires restricted flooding with two hops. Whenever a node is assigned a slot, it broadcasts the slot number and its neighbors then rebroadcast it. In a dense network, a packet tends to be congested while flooding incurs overhead. A low-cost method for maintaining the probability is presented in Sect. 4.4.
The elected node performs a time slot reservation process to select the smallest conflict-free slot. A direct way to implement the reservation process is that the node broadcasts a request message and waits until all responses are gathered from its neighbors [7], [11]. The response contains time slots assigned to the responding node and its one-hop neighbors. The requesting node now has knowledge of all assigned slots and can reserve the smallest conflict-free slot. This method requires an efficient MAC protocol. As the number of neighbors becomes larger, more responses tend to be lost, without efficient collision avoidance. Moreover, whenever a response is lost, the responding node is required to retransmit it later. Without efficient collision avoidance and retransmission, this method shows poor run-time performance. This paper proposes an efficient method for the slot reservation process that does not require collision avoidance or retransmission. The method is fully explained in the following section.

4. HRAND (Hashing-based distributed RAND)

HRAND runs periodically to reassign time slots to nodes due to the node mobility. The period for dealing with the topology change is called the random access period [8], [16]. Before running HRAND, a node processes neighbor discovery and global time synchronization. Thereafter, time is divided into rounds of a constant length and nodes within two hops are synchronized on the round boundary within a guard time (i.e., 5 ms). In this section, we describe the terminology and the prior processes, introduce the basic scheme (HRAND-basic), and then explain two enhanced schemes (HRAND-p and HRAND-pi).

4.1 Terminology

Initially, all nodes do not have time slots and a node without a slot is termed an uncolored node. In contrast, when completing a slot assignment, a node reserves its own slot and is called a colored node. After all nodes are deployed or during the random access period, a node obtains neighbor information through neighbor discovery: let \( N_1(i) \) stand for the set of one-hop neighbors of node \( i \) and let \( N_2(i) \) stand for the set of two-hop neighbors of node \( i \). Note that two-hop neighbors of node \( i \) include the nodes that are connected to node \( i \) in exactly two hops \( (N_1(i) \cap N_2(i) = \emptyset) \). Thus, the set of nodes within two hops for node \( i \) is \( |i| \cup N_1(i) \cup N_2(i) \), where the number of nodes is denoted by \( N \). Let \( C_1(i) \) \( (C_2(i)) \) be the set of colored nodes in \( N_1(i) \) \( (N_2(i)) \) at a given time. Only when \( i \) is in clear context, we use \( N_1, N_2, C_1, \) and \( C_2 \) instead. Let \( n_1(n_2) \) be \( |N_1| \cup |N_2| \), where \(|S|\) is the cardinality of set \( S \).

4.2 Neighbor Discovery and Global Time Synchronization

Whenever the random access period begins, the reference node initiates global time synchronization by using TPSN [4]. Meanwhile, each node performs neighbor discovery to maintain the link connectivity; a node periodically broadcasts a hello message. Upon first receiving the global time message, a node synchronizes to the global time and then attaches the time and sequence number to a hello message in the subsequent transmissions. The hello message piggybacked by the global time fulfills two aims at once. However, the message may be lost due to collisions. The pertinent questions then are about how many retransmissions of the message are required and what is a suitable length of interval between retransmissions. According to Eq. (5) in [16], given that a node has \( N \) nodes within two hops, 7 retransmissions with an interval of 1.44\( N \) guarantee 99% packet delivery.

4.3 HRAND-Base

HRAND assigns slots to nodes in rounds. Each round consists of three phases, as shown in Fig. 1: REQUEST, RESPONSE, and ANNOUNCEMENT phases. Each phase consists of a single frame. The details of these phases are as follows.

**REQUEST phase:** At the beginning of each round, an uncolored node \( i \) computes priorities of nodes, defined as \( \text{pr} (j) = \text{hash}(j \oplus k) \oplus j \), where \( j \in |i| \cup N_1(i) \cup N_2(i) \) and \( k \) is the \( k \)th round. If node \( i \) has the highest priority, then the node is the owner of the \( k \)th round and records a test slot \( t_i \) in a request message. Note that the initial value of \( t_i \) is 0; and the value of \( t_i \) is increased by 1 at the end of the round when node \( i \) fails to obtain a slot. This round is termed one iteration of node \( i \).

Node \( i \) then broadcasts a request message that contains the source id, \( t_i \), and time stamp. The time stamp is used for local time synchronization, as in ZMAC [17], where receiving nodes update their time by taking a weighted average of the current time and received time. Because the node with the highest priority accesses the channel, all its neighbors receive the request message without collisions.

**RESPONSE phase:** Upon receiving the request message from node \( i \), neighbor \( j \) checks whether \( t_i \) conflicts with slots assigned to the node itself or nodes in \( C_1(j) \). If there is no conflict, node \( j \) remains silent during this phase. The silence corresponds to implicit consent for node \( i \) to use \( t_i \). This method for acknowledgement is adopted from FPRP [12]. In Fig. 2, all nodes remain silent during a round and requesting node 3 determines its slot with \( t_i \).

On the other hand, if there is a conflict, node \( j \) transmits a reject message in response to the request. In Fig. 3, node 1 transmits a reject and the requesting node fails to reserve a slot. In the case of multiple rejects, the messages

![Fig. 1](image-url)
can be congested. The requesting node discriminates between a grant and a reject based on the state of the wireless medium. If the medium is busy, the response is assumed to be a reject; otherwise, it is assumed to be a grant.

**ANNOUNCEMENT phase:** During the RESPONSE phase, the requesting node \(i\) receives 1) nothing, or 2) a single reject, or 3) a congested message. In the case of 1), the node reserves its test slot \(t_i\) as its slot and broadcasts a success message, as shown in Fig. 2. The success message contains the source id and the reserved slot. Neighbors receiving the message store the slot number in the assigned slot list \(AS(i)\), which is used to adjust a test slot. The node learned that slots of \([0, t_i - 1]\) were not available in the previous iterations. These slots are similarly not available in the current iteration. Therefore, the reserved slot is the smallest conflict-free slot among the available slots. In the cases of 2) and 3), the node fails to reserve a slot and broadcasts a failure message, as shown in Fig. 3. Node \(i\) then adjusts \(t_i\) to \(\min\{t | t \in AS(i^\prime), t > t_i\}\). Given that node \(i\) failed with \(t_i = 1\) and \(AS(i) = [0, 2, 3]\), \(t_i\) is set to 4 at the end of the phase.

### 4.4 Improvements: HRAND-p and HRAND-pi

In order to improve the run-time of HRAND-basic, it is necessary to examine the election probability and the number of iterations. Because the probability is equal to \(1/((n_1 + n_2 + 1)\), it takes \((n_1 + n_2 + 1)\) rounds on average for a node to become a round owner. As neighbors reserve slots one by one, one of the colored nodes may come to own a round unnecessarily. Assume that node 4 succeeded to reserve a slot in the 4th round, as shown in Fig. 4. Unfortunately, node 4 wins other uncolored nodes again in the \((i + 1)\)th round. As a result, the \((i + 1)\)th round is wasted. The node election using distributed hashing is close to optimal at the beginning of the process, but becomes worse later. To avoid wasting rounds, it is necessary to increase the election probability of uncolored nodes.

Next, it takes at most \((n_2 + 1)\) iterations for a node to complete the slot assignment. Whenever a one-hop neighbor reserves a slot, a node learns a newly assigned slot from a success. Because the node skips the slot in the subsequent iterations, \(n_1\) is irrelevant to the number of iterations. On the other hand, when one of two-hop neighbors reserves a slot, a node is not informed of the newly assigned slot, because the node is out of transmission range. Therefore, the node needs to send a request in at most \((n_2 + 1)\) times. Assume that node 1 has five colored two-hop neighbors, as shown in Fig. 5. Obviously, the node fails to reserve slot 0 in the first iteration. After performing five failed iterations, the node succeeds to reserve slot 5 in the 6th iteration. If its one-hop neighbor was assigned before, node 1 can reserve slot 6 in the same 6th iteration.

We present the details of HRAND-p and HRAND-pi. HRAND-p aims to increase the election probability of uncolored nodes. HRAND-p and AEA (Adaptive Election Algorithm) in TRAMA [8] have some similarities in the sense that unnecessary nodes are removed from the node election. AEA chooses a node \(u\) as an owner if \(u\) satisfies the following two conditions: 1) \(\text{prio}(u) > \text{prio}(z)\) \(\forall z \in N_2(u)\) and 2) \(\text{prio}(u) > \text{prio}(y)\) \(\forall y, y \in (N_1(u) - \text{set of unnecessary nodes})\) and \(x \in N_2(y) \cap (N_1(u) \cup N_2(u))\). In contrast, HRAND-p chooses \(u\) as a winner if \(u\) satisfies the condition: \(\text{prio}(u) > \text{prio}(w)\) \(\forall w, w \in N_1(u) \cup N_2(u) - C_1(u)\). Whereas AEA requires a node to check if its neighbor \(y\) has the highest priority among \(N_2(y)\), a node in HRAND-p simply compares its priority with priorities of uncolored nodes within two hops. The number of comparisons in AEA is bounded by \(n_2(n_1 + 1)\), but our scheme requires at most \((n_1 + n_2)\) comparisons. Note that every uncolored node performs these comparisons in every round. AEA slightly improves the election probability, but the low-cost method is...
suitable for TSA because the round duration is much shorter than the TDMA slot duration.

Figure 6(a) shows that HRAND-p increases the election probability of uncolored nodes. Although the priority of node 5(80) is lower than that of node 4(100), node 5 has the highest priority among [2,3,6,7] and steals the current round from node 4. The election probability of node 5 is increased from \( \frac{1}{(N_{iter} + |C_1(j)| - 1)} \) to \( \frac{1}{(N_{iter} + |C_1(j)| - 2)} \) in the current round and rises to \( \frac{1}{(N_{iter} + |C_1(j)| - 1)} \) if all of the one-hop neighbors are assigned. Although HRAND-p increases the election probability, the maximum probability \( p_{max} \) is limited to \( \frac{1}{n_2 + 1} \). That is, a round is wasted with probability \( (1 - p_{max}) \) in the worst case. When \( n_2 \) becomes larger, the performance gain in run-time is decreased. The problem of wasted rounds is handled in HRAND-pi.

HRAND-pi aims to reduce the number of iterations while it utilizes the increased probability of HRAND-p. Whenever there is no uncolored node that steals a round from a colored owner \( i \), the node broadcasts an advertisement. The message contains the slots of \( [i] \cup C_1(i) \) and let its neighbor \( j \) know a portion of slots assigned to \( N_2(j) \). Node \( j \) then stores the slots in \( AS(j) \) and adjusts its test slot.

Consider the topology in Fig. 6(b). Let node 4 be a colored node with the highest priority (100) and let node 3 and node 5 be uncolored nodes with priority 95 and 80, respectively. Neither node 3 nor node 5 owns the current round because node 3 assumes that node 4 is the round owner and node 5 does not own the round due to node 4. In the case of HRAND-p, the current round is simply wasted. On the other hand, HRAND-pi allows node 4 to broadcast an advertisement only when the medium remains silent during a time threshold (i.e., 2*guard time). Because node 4 is not certain whether there is an uncolored winner for the current round, node 4 should check the medium state before broadcasting an advertisement. Upon receiving the advertisement from node 4, nodes 2 and 5 acquire two slots of nodes 6 and 7. That is, nodes 2 and 5 save up to two iterations.

It is important that nodes are informed of two-hop neighbors’ slots without additional overhead. HRAND-pi successfully reduces the number of iterations by up to the number of time slots \( (s) \) obtained through an advertisement. Given that a node is elected with probability \( p' \), it takes \( \frac{1}{p'} \) rounds on average for a node to perform one iteration. Therefore, HRAND-pi reduces the average number of rounds by up to \( \frac{1}{p'} \).

5. Analysis

In this section, we analyze the average number of rounds \( E[T] \) taken for a node to reserve a time slot. \( E[T] = E[\sum_{i=1}^{N_{iter}} R_i] = E[N_{iter}] \cdot E[R] \), where \( N_{iter} \) is the number of iterations taken for a node to complete TSA and \( R \) is the number of rounds taken for a node to be elected. As we described, \( N_{iter} \leq n_2 + 1 \); hence, \( E[N_{iter}] \leq n_2 + 1 \). The election probability \( p \) of HRAND-basic is equal to \( \frac{1}{n_1} \) and \( E[R] = n_1 + n_2 + 1 \); hence, \( E[T] \leq (n_2 + 1)(n_1 + n_2 + 1) \). The run-time performance of HRAND is \( O(\delta^2) \) in the worst case. To obtain a better upper-bound of \( E[T] \), we apply the Markov-chain model to HRAND-basic and HRAND-pi. The topology for our analysis is a two-hop network, shown in Fig. 7. Nodes except node 0 form a complete graph of \( (N-1) \) size. The number of edges connected to node 0 is equal to \( n_1 \). We present \( E[T] \) of node 0 on the topology.

The Markov chain for HRAND-basic is described as follows: a set of states \( S \) is defined as \( [(c,i)]c = |C_2(0)|, 0 \leq i \leq c + 1 \), where \( i \) is the number of iterations performed by node 0. The process starts in \( (0,0) \), which means that there are no colored two-hop neighbors and there is no iteration performed by node 0. If the chain moves to \( (0,1) \), node 0 is the first node that performs an iteration and the process is finished. In general, \( (c,c+1) \) is an absorbing state and \( (c,i) \) for \( i \leq c \) is a transient state. If the chain moves from \( (0,0) \), one of two-hop neighbors reserves a slot and, therefore, node 0 needs two iterations. In general, if the chain is currently in a transient state \( (c,i) \), the chain requires at least \( (c-i+1) \) steps before being absorbed. Note that one step in the Markov chain corresponds to one round in HRAND.

In one step, the chain can move from \( (c,i) \) to one of three states: \( (c+1,i), (c,i+1) \), and \( (c,i) \). The transition from \( (c,i) \) to \( (c+1,i) \) indicates that a new node in \( N_2 \) will be elected and assigned in the next round. Because node 0 is not yet assigned, two-hop neighbors obtain a slot with one iteration \( (N_{iter} \leq 2) \). The transition probability \( p_{(c,i),(c+1,i)} \) is equal to the probability that one of the uncolored nodes in \( N_2 \) is selected among \( N \), as shown in Eq. (1).

The transition from \( (c,i) \) to \( (c,i+1) \) indicates that node 0 will perform an iteration in the next round. If \( i = c \), the chain will move to an absorbing state and node 0 will be
assigned. In the case of \( i < c \), the chain will move to a transient state and node 0 will not be assigned. The transition probability \( p_{(c,i),(c+1,i)} \) is given in Eq. (2).

The transition from \((c,i)\) to \((c,i)\) corresponds to either of the two cases below. Case 1) if \( i \leq c \), either of one of the nodes in \( N_1 \) or one of the colored nodes in \( N_2 \) will be elected in the next round. Nodes in \( N_1 \) and colored nodes in \( N_2 \) do not affect on the number of iterations taken by node 0. Hence, the state is not changed. Case 2) if \( i = c + 1 \), node 0 is already assigned and the chain is in an absorbing state. The transition probability \( p_{(c,i),(c+1,i)} \) is given in Eq. (3).

\[
p_{(c,i),(c+1,i)} = Pr[X_{m+1} = (c+1,i) | X_m = (c,i)]
\]

\[
p_{(c,i),(c+1,i)} = \frac{a_{c+c}}{N}, \quad \text{for } 0 \leq c < n_2
\]

\[
p_{(c,i),(c+1,i)} = \frac{1}{N}, \quad \text{for } 0 \leq i \leq c
\]

\[
p_{(c,i),(c+1,i)} = \frac{1}{c+1}, \quad \text{for } i = c + 1
\]

Given \( n_1 = 1 \) and \( n_2 = 2 \) in a two-hop network, Fig. 8 shows a Markov diagram for HRAND-basic. Let \( s(k) \) be a state denoted by \((c_k,i_k)\). The Markov chain has six transient states \( s(k) \) for \( 1 \leq k \leq 6 \) and three absorbing states \( s(j) \) for \( 7 \leq j \leq 9 \). The chain starts in \( s(0) \) and is absorbed in one of \( s(7), s(8) \) and \( s(9) \). For example, one of transitions from \( X_0 = s(1) \) to an absorbing state in three steps is \( s(1) \rightarrow s(2) \rightarrow s(4) \rightarrow s(7) \). The expected number of steps before the chain is absorbed is equal to the expected number of rounds \( E[T] \) taken for node 0 to be assigned. The fundamental matrix is used to solve the absorption time.

The transition matrix \( P \) of the absorbing Markov chain having \( r = (n_2+1)(n_2+2)/2 \) transient states and \( r = (n_2+1) \) absorbing states is represented in the canonical form:

\[
P = \begin{bmatrix} \mathbf{Q} & \mathbf{R} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}
\]

where \( \mathbf{Q} \) is a \( t \times t \) matrix, \( \mathbf{R} \) is a \( t \times r \) matrix, \( \mathbf{0} \) is an \( r \times t \) zero matrix, and \( \mathbf{I} \) is an \( r \times r \) identity matrix. The fundamental matrix \( \mathbf{N} \) has an inverse of the matrix \( (I - \mathbf{Q}) \) and the entry \( n_{i,j} \) of \( \mathbf{N} \) is the expected number of times that a chain visits state \( j \), given that it starts in state \( i \). Because the chain for HRAND starts in state \( s(1) = (0,0) \), \( E[T] \) is equal to the sum of the entries in the first row of \( \mathbf{N} \): \( E[T] = \sum_{j=1}^{6} n_{s(1),s(j)} \). Given the Makov chain in Fig. 8, the matrix \( \mathbf{N} \) is computed as follows and \( E[T] = \sum_{j=1}^{6} n_{s(1),s(j)} = 8.67 \) rounds.

\[
\mathbf{N} = (I - \mathbf{Q})^{-1}
\]

\[
s(1) \begin{pmatrix} 1.33 & 1.33 & 1.33 & 0.67 & 2 & 2 \\ 0 & 0 & 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 & 4 \end{pmatrix}
\]

Next, the Markov chain for HRAND-p is built in a similar fashion. The set of states is defined as \( S = \{(c_2, i_2) | c_2 = |C_2|, i_2 = |C_1|, 0 \leq i \leq a + 1\} \). If \( i = c_2 + 1 \), a state \((c_2, i_2)\) is an absorbing state. If \( i \leq c_2 \), a state \((c_2, i_2)\) is a transient state. The equations of the transition probability are given below. Given \( n_1 = 1 \) and \( n_2 \) in the two-hop network, the average number of rounds required for HRAND-p is equal to 7.12 rounds and HRAND-p reduces the average number of rounds by 17.4% compared to HRAND-basic.

\[
p_{(c_2,i_2),(c_2+1,i_2)} = \frac{n_2 - c_2}{N - c_1}, \quad \text{for } c_2 < n_2
\]

\[
p_{(c_2,i_2),(c_2,i_2+1)} = \frac{1}{N - c_1}, \quad \text{for } i \leq c_2
\]

\[
p_{(c_2,i_2),(c_2,i_2+1)} = \frac{n_1 - c_1}{N - c_1}, \quad \text{for } c_1 < n_1
\]

\[
p_{(c_2,i_2),(c_2,i_2+1)} = \frac{1}{N - c_1}, \quad \text{for } i = c_2 + 1
\]

6. Simulation Result

In this section, we verify the analysis through comparison with simulation results in two-hop networks and evaluate the run-time of HRAND in multi-hop networks for comparison with DRAND. We also show the overhead of neighbor discovery and global time synchronization. We implement HRAND using the ns-2 simulator [18]. Sensor nodes in our simulation are based on a Mica2 platform with a transmission range of 20 m and a data rate of 40 kbps. During TSA, nodes are listening without sleeping. The frame format in transmission is based on BMAC [19]. A round consists of one 5 ms guard time and either three frames or one advertisement message: its duration is set to 17 ms.
In order to validate the analysis, we perform simulations on the two-hop network topology shown in Fig. 7. Given that the number of nodes is fixed, the number of two-hop neighbors affects the run-time of node 0, because the number of iterations is bounded by \( n_2 + 1 \) and the election probability in HRAND-p and HRAND-pi is bounded by \( \frac{1}{2n_2} \). The ratio of \( n_1 \) to \( n_2 \) is an important parameter in evaluating the run-time of HRAND schemes. We make four types of topologies having different ratios: 1:3, 1:4, 1:5, and 1:6. We evaluate the average run-time of node 0 for HRAND-basic, HRAND-p, and HRAND-pi using each topology, where the number of nodes is varied from 5 to 25.

Figure 9 shows the simulation results and the calculated values. First, the simulation results in HRAND-basic and HRAND-p are nearly the same as the calculated values of the analysis. Second, the average run-time in HRAND-basic and HRAND-p is quadratically proportional to the number of nodes. Because \( E[T] \leq N \cdot (n_2 + 1) = N \cdot (N - n_1) \), \( E[T] \) is quadratically proportional to \( N \). HRAND-p reduces the run-time of HRAND-p by up to 23%. However, the runtime improvement of HRAND-p decreases, while the ratio grows.

Third, HRAND-pi is scalable to the number of nodes and outperforms HRAND-basic and HRAND-p. The average run-time in HRAND-pi grows almost linearly with the number of nodes, although the run-time complexity in the worst case is \( O(\delta^2) \). HRAND-pi reduces the run-time of HRAND-p by 27% up to 88%. Although HRAND-pi uses the same election probability as HRAND-p, HRAND-pi alleviates the performance degradation significantly by using an advertisement message. The simulation results demonstrate the effectiveness of the advertisement message.

### 6.2 Multi-Hop Network

In Sect. 3, we described node election and time slot reservation processes. The existing schemes take different approaches to two processes:

- **DRAND** uses probability-based election in the election process. Each node determines whether it is a winner based on independent probability calculation. DRAND uses minimum slot-based reservation in the reservation process. The winner gathers all assigned slots from its neighbors and, in turn, chooses the minimum conflict-free slot. If there are multiple winners within two hops, the node tries again in the subsequent rounds.

- **FPRP** [12] uses probability-based election and random slot-based reservation. The protocol assigns time slots to nodes during a series of reservation slots, which are divided into reservation cycles. In every reservation cycle, each node is elected as a winner with election probability. The winner has a chance to reserve a time slot through five phase algorithm. Because a reservation slot is mapped to a time slot, the node only needs to check the existence of other winners; gathering the assigned slots is not required.

- **NAMA** [9] uses distributed hashing and random slot-based reservation. Because a node reserves a time slot with probability of \( 1/(1 + N_1 + N_2) \), the interval length between consecutive slots is a geometric distribution; the maximum length is not bounded. That is, NAMA does not guarantee the delay in worst case.

HRAND uses distributed hashing and minimum slot-based reservation. Although HRAND borrows distributed hashing and negative acknowledgement from NAMA and FPRP, our scheme employs new efficient time slot reservation. The major shortcoming of minimum slot-based scheme requires a node to gather the assigned slot from all neighbors; this task is quite time-consuming. In order to reduce the overhead and keep the number of assigned slot small, our scheme continues to check if the test slot is available, while increasing the test slot in each iteration. In addition, colored neighbors help uncolored nodes to reduce the number of iterations by informing the assigned slots.

We compare the performance of HRAND, DRAND, and FPRP in terms of the average run-time and the maximum number of assigned slots. As the network size is varied from 50 to 250, network topologies are generated randomly on a surface of 100 m x 100 m. DRAND uses the default setting of BMAC [19] (random backoff is on and acknowledgement is disabled). FPRP is repeated during 10 reservation cycles per one reservation slot, which are close to the recommended value in [12].
Figure 10 shows the average run-time taken for a node to reserve a slot in random generated network topologies.

Figure 10(a) shows the average run-time of HRAND schemes according to the number of nodes within two hops. The average run-time of HRAND-pi is almost linearly proportional to the number of nodes within two hops. As expected, HRAND-pi outperforms HRAND-basic and HRAND-p in random generated topologies. The performance gap between HRAND-p and HRAND-pi shows the effectiveness of the advertisement message.

Figure 10(b) shows the average run-time of HRAND, DRAND and FPRP. In simulation results, HRAND-pi outperforms DRAND by one or two orders of magnitude. For 20 nodes, DRAND requires about 15 seconds on average, but HRAND-pi requires a mere 0.58 seconds on average. As two-hop neighborhood size grows, the performance gap between DRAND and HRAND-pi becomes much larger. The results are mainly due to performance in slot reservation process. As the number of one-hop neighbors grows, the node in DRAND spends more time gathering the assigned slots from neighbors. In addition, collisions between responses make performance worse (random backoffs and retransmissions). On the other hand, the node in HRAND immediately receives responses after sending the test slot. Furthermore, advertisement messages significantly reduce the number of trials on the test slots. It is also encouraging that HRAND-pi is comparable to FPRP, which is tuned to the fast run-time (random slot-based scheme). Note that FPRP does not completely eliminate the slot conflicts, which occurs at nodes in network boundary.

Figure 11 shows the maximum number of assigned slots in HRAND, DRAND, and FPRP. Since both HRAND and DRAND are based on RAND, the number of assigned slots is theoretically bounded by $\delta + 1$. In FPRP, however, the maximum number of slots is not explicitly bounded due to randomness in slot reservation. In Fig. 11, the slot number of HRAND and DRAND is far less than $\delta + 1$, but the slot number of FPRP exceeds $\delta + 1$. Because the maximum number of slots represents the size of TDMA frame, the reduction in the slot number implies improvement of the overall performance in channel utilization.

TDMA using DRAND periodically suffers from the large run-time overhead and, therefore, DRAND is suitable for a relatively static network topology. On the other hand, FPRP provides fairly fast TSA solution, but the maximum number of assigned slots is not bounded theoretically. Our schemes perform TSA very rapidly while requiring the maximum number of assigned slots that is far less than $\delta + 1$.

6.3 Overhead Cost

Before running HRAND, each node performs neighbor discovery and global time synchronization. We implement one-hop and two-hop neighbor discoveries using the neighbor protocol in [16] and pseudo-bayesian broadcast in [20]. In neighbor discovery, the number of nodes within two hops ($\hat{N}$) is estimated to $4\pi R^2 \rho$, where $R$ is the transmission range and $\rho$ is the node density. Each node broadcasts 14 hello messages redundantly for the link stability check during $2T$ ($T = 1.44\hat{N}$) and subsequently broadcasts 14 two-hop messages during $2T$.

Figure 12 shows the overhead incurred by neighbor discoveries and global time synchronization in the same multi-hop networks as given in Sect. 6.2. Global time synchronization requires one packet flooding and incurs small overhead. The delay in both the neighbor discoveries is linearly propositional to the number of nodes within two-hops. However, the cost of the two-hop neighbor discovery is quite large compared to the run-time of the HRAND schemes. In future work, we plan to study methods to minimizing the overhead of two-hop neighbor discovery.
We have presented HRAND, an efficient distributed TDMA slot assignment scheme. HRAND is designed to generate efficient scheduling and reduce the periodic run-time overhead for wireless sensor and actor networks. In order to reduce the run-time, we have developed three HRAND schemes: HRAND-basic, HRAND-p, and HRAND-pi. HRAND-pi has a runtime complexity of $O(\delta^2)$ in the worst case, but simulation results show that its average run-time is almost linearly proportional to the two-hop neighborhood size. HRAND-basic and HRAND-p also show good performance in randomly generated network topologies. Compared to DRAND and FPRP, HRAND reduces run-time significantly and preserves the good channel utilization.

7. Conclusions

We have presented HRAND, an efficient distributed TDMA slot assignment scheme. HRAND is designed to generate efficient scheduling and reduce the periodic run-time overhead for wireless sensor and actor networks. In order to reduce the run-time, we have developed three HRAND schemes: HRAND-basic, HRAND-p, and HRAND-pi. HRAND-pi has a runtime complexity of $O(\delta^2)$ in the worst case, but simulation results show that its average run-time is almost linearly proportional to the two-hop neighborhood size. HRAND-basic and HRAND-p also show good performance in randomly generated network topologies. Compared to DRAND and FPRP, HRAND reduces run-time significantly and preserves the good channel utilization.

References

Hyunsoo Yoon received the B.S. degree in electronics engineering from the Seoul National University, Korea, in 1979, the M.S. degree in computer science from the KAIST, in 1981, and the Ph.D. degree in computer and information science from the Ohio State University, Columbus, Ohio, in 1988. From 1988 to 1989, with the AT&T Bell Labs, as a Member of Technical Staff. Since 1989 he has been a Faculty Member of the Division of Computer Science, KAIST. His research interests include parallel computer architecture, mobile communication, ad hoc networks, and information security.