Deterministic Recurrent Communication and Synchronization in Restricted Sensor Networks

Antonio Fernández Anta, Miguel Mosteiro, Christopher Thraves-Caro

Deterministic Recurrent Communication and Synchronization in Restricted Sensor Networks

Antonio Fernández Anta[1], Miguel A. Mosteiro[1], and Christopher Thraves[2]

[1] LADyR, GSyC, Universidad Rey Juan Carlos, 28933 Móstoles, Madrid, Spain
anto@gsyc.es, miguel.mosteiro@urjc.es
[2] ASAP Project team, IRISA/INRIA Rennes, Campus Universitaire de Beaulieu, 35043 Rennes Cedex, France.

Abstract. Monitoring physical phenomena in Sensor Networks requires guaranteeing permanent communication between nodes. Moreover, in an effective implementation of such infrastructure, the delay between any two consecutive communications should be minimized. The problem is challenging because, in a restricted Sensor Network, the communication is carried out through a single and shared radio channel without collision detection. Dealing with collisions is crucial to ensure effective communication between nodes. Additionally, minimizing them yields energy consumption minimization, given that sensing and computational costs in terms of energy are negligible with respect to radio communication. In this work, we present a deterministic recurrent-communication protocol for Sensor Networks. After an initial negotiation phase of the access pattern to the channel, each node running this protocol reaches a steady state, which is asymptotically optimal in terms of energy and time efficiency. As a by-product, a protocol for the synchronization of a Sensor Network is also proposed. Furthermore, the protocols are resilient to an arbitrary node power-up schedule and a general node failure model.

1 Introduction

A Sensor Network is an infrastructure deployed in a hostile or remote area for monitoring purposes. The basic entities of a Sensor Network are called *sensor nodes*, small devices provided with radio-communication, processing, and sensing capabilities. Upon being distributed at random in the area of interest, sensor nodes have to build a communication system from scratch. A strong shortcoming in Sensor Networks is the energy supply of sensor nodes. Consequently, one of the main challenges is the efficient administration of such resource, extending the usability of the network. In sensor nodes, sensing and computational costs in terms of energy consumption are negligible with respect to radio communication. Thus, it is crucial to optimize the communication schedule. In a harshly restricted Sensor Network, the communication is carried out by means of a single and shared radio channel where nodes may broadcast messages to all neighboring nodes but no collision detection mechanism is available. Therefore, special mechanisms to effectively transmit or receive a message are required. Indeed, a node b receives a message transmitted from a neighboring node a only if neither b nor the other neighbors of b transmit at the same time. Otherwise, a collision occurs and the messages are garbled. Furthermore, b is not able to recognize the difference between

This research was partially supported by the Spanish MICINN under grant no. TIN2008-06735-C02-01 and the Comunidad de Madrid under grant no. S2009TIC-1692. EU Marie Curie International Reintegration Grant IRG 210021. And, French ANR project Shaman.

this garbled message received and the background noise present in the channel if no transmission is produced.

The mechanism used by a node to decide to transmit or receive at any time is called the *transmission schedule*. Some transmission schedules use randomness to avoid collisions, but frequently involve a large number of redundant transmissions, consequently incurring in excessive energy consumption. On the other hand, deterministic transmission schedules, although efficient in terms of energy consumption, usually provide only large time guarantees for successful communication. Therefore, the problem addressed in this work, i.e., to find a deterministic transmission schedule with optimal time and energy guarantees of successful communication, is a fundamental question in Sensor Networks.

ROAD MAP: The rest of the document is organized as follow. In Sections 2, the model and problem definition are presented. In Section 3, our results are presented and contextualized with the previous results. Section 4 contains synchronization algorithms of independent interest. Finally, deterministic recurrent communication algorithms are introduced in Section 5.

2 Model and Problems Definition

Regarding network topology and connectivity, and node constraints, we use the restrictive model in [24, 25, 23] summarized here as follows.

NETWORK AND NODES: Let us denote with V the set of sensors. Each sensor node is assumed to have a unique identification number (ID) in $\{0, \ldots, n-1\}$. Sensors are expected to be deployed at random in the area of interest. Each sensor is provided with a radio system to communicate with the rest of the network, but each radio system has only a limited range for transmissions and receptions. It is assumed that the transmission range and the reception range are the same, and it is referred as the communication range. Consequently, each node is able to communicate with a restricted number of other sensors, the ones deployed within its communication range. In this work, we use an undirected graph $G = (V, E)$ to model the topology of the network. Each node in V represents a sensor node, and the link $(u, v) \in E$ represents that nodes u and v are in communication range. (*Geometric Graph.*) Let us denote with $N(v)$ the set of neighbors of node v. Let $n = |V|$ denote the number of nodes in the network, and let $k = \max_{v \in V} |N(v)|$ be the maximum degree of a node in G (i.e., the network). Finally, we use D to denote the diameter of the network. Unless otherwise stated, we assume that n, k and D are known by all the nodes in the system. (We assume the precise values are known for clarity, but limiting that knowledge to asymptotically tight upper bounds yield the same results asymptotically.) Regarding computational resources of sensor nodes, node-memory size, is restricted only to $O(k + \log n)$ bits. Were the deployment of nodes uniform (random geometric graph) as it is popularly assumed in the Sensor Networks literature [33, 1], our protocols would work even if the node-memory size is restricted to just $O(\log n)$ bits.

LOCAL SYNCHRONY: Time is assumed to be slotted in equal-length *time slots* or *steps*. It is assumed that the length of a slot is sufficient to transmit one message, i.e., each transmission occurs in a given slot. Without loss of generality [34], it is assumed that the slots of all nodes are in phase, i.e., they all start and finish at the same time instants. For convenience, we assume a *global time* that takes non-negative integer values and advances one unit per step. Note that this is a fictional device and that the nodes do not have access to its value. For convenience we assume that the global time is

the number of time steps since the first nodes in the system have been awakened. We assume the availability of a hardware clock mechanism at each node, denoted local-clock, such that, starting from 0 when the node is powered up, the clock is incremented by one automatically at the end of each time slot[3]. Then, for all $i \in V$ and $t \in \mathbb{Z}^+$, $\mathsf{local\text{-}clock}_i(t)$ denotes the value of local-clock of node i at time step t before being incremented. In the first step t executed by a node i, $\mathsf{local\text{-}clock}_i(t) = 0$.

Node Awakening and Types of Adversaries: Nodes are in two possible states, *sleeping* and *awake*. It is assumed that initially all nodes are sleeping. The nodes are assumed to be awakened by an adversary.[4] Without loss of generality, it is assumed that every node of the network is eventually awakened. In the rest of the paper x will be used to denote the first node awakened by the adversary, breaking ties arbitrarily. As x is always awake (see below), $\forall t \geq 0 : \mathsf{local\text{-}clock}_x(t) = t$. Regarding node reliability, as customary in the Sensor Networks literature, we assume that nodes may fail. I.e., a node may crash and stop working. The adversary decides when to crash and recover (awake again) nodes. However, if crashes and recoveries occur arbitrarily, due to determinism, there exist topologies for which the adversary may stop a node from receiving any message, even if connectivity is required.[5] Thus, limitations to the crash/recovery schedule are in order. In this work, we consider node failures as long as: (i) the network stays connected (one connected component) at all times, (ii) node x is always awake (in fact it would be enough if there is always some node that has the global time up and running), and (iii) each period when a node runs without failures lasts at least the length of the stabilization time (as defined in Section 2.1). In this work, we consider two types of adversaries.

Definition 1. *A* τ-adversary *is an adversary that awakens all the nodes of the network within a window time of size* τ*, i.e., no node is awakened at a time* $t \geq \tau$*. The parameter* τ *is assumed known by the nodes.*

Since the stabilization time under a τ-adversary is at least τ, according to the failure model assumed, a τ-adversary cannot recover crashed nodes.

Definition 2. *A* ∞-adversary *is an adversary that has no restriction on when nodes are awakened.*

Communication: Each radio system transmits and receives in a single and shared radio channel. Therefore, at each step, each node decides between transmission mode or reception mode. Moreover, node v *receives* from node u in a slot if and only if node u is the only neighbor of v transmitting in that slot, and v is in reception mode at that slot. In the case that two or more neighbors of node v transmit in the same slot a collision occurs at node v. A node v is not able to distinguish between silence (none of the nodes in $N(v)$ transmits) and collision. We denote the communication range as r. A customary assumption in Sensor Networks [23] is that nodes can adjust the power of transmission to a smaller level, introducing only a constant factor in the number of nodes that has to be deployed to maintain connectivity. Instead, for the sake of clarity, in this work we assume that nodes can duplicate their transmission range to $2r$. Likewise, such an assumption does not yield an extra asymptotic cost. Notice that, independently of this assumption, the maximum degree k and diameter D defined before correspond to the underlying graph G defined for range r.

[3] Observe that, if not readily available, the described mechanism can be implemented as a software counter.

[4] In contrast with the wake-up problem studied in the literature, we do not assume that sleeping nodes may additionally be awaken by the transmission of a neighboring node.

[5] For any time slot t, if none or more than one neighbor transmit, do nothing. Otherwise, put the transmitter to sleep during t.

2.1 Deterministic Recurrent Communication Problem

The problem solved in this paper is called *deterministic recurrent communication*. The goal in solving this problem is to provide a communication service that can be used by the components of a distributed application residing in different nodes to exchange *application messages*. Thus, the service must allow a component in a node to recurrently communicate with the components in neighboring nodes. For the sake of clarity, we assume that all nodes run application components that have an infinite supply of application messages to transmit.

Definition 3. *A distributed protocol solves the* deterministic recurrent communication *(DRC) problem if it guarantees that, for every step t and every pair $(u, v) \in E$, there is some step $t' \geq t$ such that, in step t', v receives an application message from u.*

The protocols proposed in this paper are adaptive, in the sense that when nodes are awakened, they run a *start-up phase*. During this phase, nodes use *control messages* to agree on a periodic transmission schedule. After the start-up phase, a *stable phase* starts in which they use the agreed transmission schedule to exchange application messages. For some of the protocols, control messages still have to be used in the stable phase. We use three goodness parameters to evaluate these protocols. The first one is the maximum number of steps of the start-up phase for any node, called the *stabilization time*. Then, we define the following metrics to evaluate energy and time efficiency in the stable phase. For any $(u, v) \in E$ and any $i > 1$, let $R_u^i(v)$ be the number of transmissions of u between the $(i-1)^{th}$ and the i^{th} receptions of application messages from u at v, and $R_u(v) = \max_i R_u^i(v)$. In order to measure time we denote $\Delta R_u^i(v)$ the time (number of time slots) that are between the $(i-1)^{th}$ and the i^{th} receptions of application messages from u at v, and $\Delta R_u(v) = \max_i \Delta R_u^i(v)$. We define the *message complexity* of a protocol for DRC as $\max_{(u,v)\in E} R_u(v)$. We define the *delay* of a protocol for DRC as $\max_{(u,v)\in E} \Delta R_u(v)$.

In this paper, the goal is to derive protocols that solve DRC with asymptotically optimal message complexity and delay, even if they incur in significant stabilization times. We design our protocols assuming the existence of an *oblivious deterministic recurrent communication protocol* that solves DRC without a start-up phase. In this protocol, whether a node u is in transmission or reception mode at step t is a function only of u's ID and $\mathsf{local\text{-}clock}_u(t)$ (the number of steps u has been awake). Such oblivious deterministic protocols exist. An example is the Primed Selection communication protocol proposed in [24]. In the rest of the paper, the oblivious deterministic recurrent communication protocol will be modeled as a binary function $\mathtt{ORC}$ on $V \times \mathbb{Z}^+$. Then, for all $u \in V$ and all $j \in \mathbb{Z}^+$ we have $\mathtt{ORC}(u, j) \in \{transmit, receive\}$. The delay of this protocol will be denoted by T. Since oblivious protocols have no start-up phase, this means that if nodes u and v, such that $(u, v) \in E$, are awake and run $\mathtt{ORC}$, in every interval of T steps they will receive from each other.

2.2 The Synchronization Problem

As a by-product of the protocols proposed in this paper for DRC, we propose also deterministic protocols that solve the *synchronization problem* under both classes of adversaries defined. In the synchronization problem it is assumed that each node has a slot counter $\mathsf{global\text{-}clock}$ (incremented in every step) and a Boolean variable synced indicating whether it is synchronized or not. The slot counters of all synchronized nodes must have the same value. For each node $i \in V$ and time slot t, let $\mathsf{global\text{-}clock}_i(t)$ and $\mathsf{synced}_i(t)$ be, respectively, the slot counter and the Boolean variable of node i at the beginning of time slot t. We say that a network is *synchronized* at a time

step$t \in \mathbb{Z}^+$ if, for all $i, j \in V$, such that $\mathsf{synced}_i(t) = \mathsf{synced}_j(t) = true$, it holds that $\mathsf{global\text{-}clock}_i(t) = \mathsf{global\text{-}clock}_j(t)$.

Definition 4. *We say that a protocol solves the* synchronization problem *if there exists a time t from which the protocol guarantees that the network is synchronized at all times after t, and every node that awakes eventually gets synchronized. The maximum time between a node awaking and getting synchronized is the* synchronization time *of the protocol.*

In the synchronization protocols proposed here each node initializes its counter global-clock to 0 and increments it by 1 every step. A node can also adopt a larger global-clock value from another node. Then, since x is the first node awake and it never fails, it will always have the largest global-clock counter, i.e., for each node $u \in V$ and each $t \geq 0$, if u is awake at time t then $\mathsf{global\text{-}clock}_u(t) \leq \mathsf{global\text{-}clock}_x(t)$. Moreover, $\forall t \geq 0 : \mathsf{local\text{-}clock}_x(t) = \mathsf{global\text{-}clock}_x(t) = t$.

3 Framing Our Results with Related Work.

To the best of our knowledge, deterministic recurrent communication under a restricted Sensor Network model was only studied in [24] and later improved in [25]. It was shown in the latter a message-complexity optimal oblivious protocol with delay at most $k(n+k)(\ln(n+k) + \ln\ln(n+k))$, which is shown to be optimal delay-wise for a subclass of non-adaptive protocols for most values of k. For adaptive protocols, it was shown in that work a delay of $O(k^2 \log k)$ relaxing memory size constraints and an asymptotically optimal $O(k)$ additionally limiting the adversarial node awakening schedule. In the present paper, a worst-case asymptotically optimal $O(k)$ delay bound is proven, even removing those restrictions.

The question of how to disseminate information in Radio Networks has led to different well-studied important problems such as *Broadcast* [5, 31], *Selection* [30], and *Gossiping* [32, 10]. These problems differ in the number of nodes that hold a possibly different message to disseminate to all nodes in the network. Although these are one-shot communication primitives, some of the results obtained could be used repeatedly to achieve recurrent communication.

Deterministic solutions [14, 11, 16, 8] for Broadcast and Gossiping include assumptions such as simultaneous startup or the availability of a global clock, which are not feasible in Sensor Networks. The selection problem, on the other hand, was studied by Kowalski [30] in a model where the node awakening schedule is adversarial, proving the existence of a $O(k^2 \log n)$ algorithm and showing constructively how to obtain an algorithm that achieves $O(k^2 \text{ polylog } n)$. These results are obtained for a model where nodes turn off upon successful transmission. Thus, they do not apply to our setting also.

In [2], Alon, Bar-Noy, Linial and Peleg gave a deterministic distributed protocol to simulate the message passing model in radio networks. Using this technique, each node receives a transmission of all its neighbors after $O(k^2 \log^2 n / \log(k \log n))$ steps. Again, simultaneous awakening of nodes is required, a feature that can not be assumed in restricted models of Sensor Networks. In the same paper, lower bounds for this problem are also proved by showing bipartite graphs that require $\Omega(k \log k)$ rounds. Bipartite graphs with maximum degree $\omega(1)$ are not embeddable in geometric graphs therefore these bounds do not apply to our setting.

Related lines of work from combinatorics include *selectors*, *selective-* and *strongly-selective families* [29, 15, 22, 7]. The application of any of these combinatorial objects to recurrent communication in Radio Networks would require simultaneous awakening of the participating nodes. Within the scope of the *wake-up* problem, the existence

of a combinatorial structure called *radio-synchronizer* was shown in [13], later explicited in [12]. The existence of an extension of radio-synchronizers, called *universal-synchronizers*, was also shown in the latter, and a constructive proof of universal-synchronizers was given in [9]. In Radio Networks terminology, a radio-synchronizer is an n-set of schedules of transmissions (one for each node) such that, for any node awakening schedule and for any subset of k nodes, there is a time step when exactly one of the k nodes transmits. Synchronizers (radio- or universal-) are of the utmost importance in Radio Networks because they tolerate arbitrary rotations of each schedule of transmissions. In other words, they can be used obliviously without assuming any specific node awakening schedule. Furthermore, due to the same reason, synchronizers could be used repeatedly to implement a recurrent communication primitive, as long as it is enough for each node to receive messages from *some* neighboring node infinitely many times. In the present paper, we study a recurrent communication primitive that requires each node to receive from *each* neighboring node infinitely many times. (See Definition 3.)

In order to compute a transmission schedule that solves DRC with asymptotically optimal delay bound, we include in the algorithm presented in this paper a synchronizing phase. Within the scope of Radio Networks, the problem of globally synchronizing the network has been recently studied in [21], but their model includes a single-hop network and many channels of communication.

The application of Radio Network wake-up protocols to global synchronization was studied in [13, 12, 9]. In their model, nodes may be awaken adversarially, but additionally they may be also awaken by the transmission of another node. The synchronization technique proposed takes advantage of the latter and works only after all nodes have been awaken. Thus, it can only be applied to our setting under a τ-adversary, adding an initial waiting phase to ensure that all nodes are awake before running that protocol. Extending the best running time obtained in [9] by the additional τ waiting steps gives $O(\tau + \min\{n, Dk\}k \text{ polylog } n)$. Whereas the synchronization algorithm of [26], suited here for a τ-adversary and using the `ORC` protocol of [24], yields a running time of $\tau + Dnk \log n$. Although which of these protocols is more efficient depends on the parameters instance, we propose the latter for clarity of the presentation towards the more general adversary.

Within the broader scope of distributed computing in general, the synchronization problem has been widely studied. The main motivation for these works has been multiprocessor systems. The problem is known there as *digital clock synchronization*: all clocks are activated from the beginning, clocks run at the same rate, however, they are initialized with arbitrary different values. The problem is to find a protocol that synchronizes the clocks in the same time, and the shown time is incremented by one in each step. Deterministic solutions are presented in [26, 3], but a much more stable framework is used as base of their analysis. For instance, the model used include a single-hop network, and possible collisions in communications are not considered. These two constraints are crucial drawbacks to apply the proposed algorithms in Sensor Networks. On the other hand, Herman and Ghosh in [27] presented a deterministic protocol to synchronize a network with a tree topology and a probabilistic protocol for a multi-hop network, restricted network and probabilistic solution collide with our purpose. The presence of Byzantine nodes is considered in [20, 19, 28, 17, 6]. But again in the restricted framework of single-hop networks, and without considering possible collisions in communication. In [18], Dolev considered the presence of crash and Byzantine faulty nodes in multi-hop networks. Dolev introduces an algorithm that works in the case of crash faults, unfortunately for us, the model includes the assumption of simultaneous awakening of nodes. On the other hand, he proves that it is not possible to synchronize the system in the presence of even one Byzantine node, the reason why we do not consider that case.

In the context of P2P networks, the synchronization problem in dynamics distributed systems was studied in [4]. The authors consider churn in their model, and they propose an algorithm based on epidemic protocols to solve the problem. The main difference with our context is the fact that the authors assume an overlay network, and a peer sampling service that continuously updates the topology of the network, an impossible assumption in Sensor Networks.

3.1 Our Results

In this work, we present an adaptive protocol that solves DRC asymptotically optimally for message complexity and delay efficiency measures. We model the arbitrary node awakening schedule, and node failures, with the two types of adversaries previously defined, τ-adversary and ∞-adversary. As a building block of our deterministic computation of an optimal transmission schedule, we include a synchronization algorithm for each of type of adversary. Once nodes are synchronized, we provide a $19(k+1)$ coloring of the network, where k is the maximum degree[6]. Thus, the transmission schedule guarantees, in the case of the first adversary, that for every time interval of length $19(k+1)$ slots, each node has at least one successful transmission to all its neighbors. In the case of the less restricted adversary, the transmission schedule has to be resilient to the awakening of new nodes. Thus, after synchronization, each time step is doubled extending the length of that interval to $38(k+1)$ slots. Due to the pigeonhole principle, these delays between reception of each neighboring node are asymptotically optimal.

Given that the efficient use of energy is crucial to extend the life-cycle of a sensor node, and that the radio-communication cost in terms of energy dominates other consumption factors, it is extremely important to minimize the number of transmissions produced that do not achieve effective application communication. The protocols presented in this paper are shown to have optimal message complexity of 0 for the restricted adversary, and a message complexity of $19(k+1)/n$ for the unrestricted adversary, which is asymptotically optimal if all nodes run application components that have an infinite supply of application messages to transmit.

4 Synchronization Protocols

In this section, we present two protocols that solve the synchronization problem. The protocols solve the problem under a τ-adversary and an ∞-adversary, respectively.

4.1 Synchronizing Against a τ-adversary

The MaxSpread Protocol presented here is a re-creation of the synchronization algorithm presented in [26]. Each node counts the number of steps since it was awaken, and transmit this counter following the `ORC` schedule of the oblivious communication protocol. If a node receives a count bigger than the value of its counter, then it updates its counter to the larger value. The details of MaxSpread Protocol are presented in Algorithm 2 in the Appendix.

The correctness of MaxSpread and its convergence time had been proven in [26]. Nevertheless, we briefly mention the key ideas to understand the behavior of the protocol. Notice that, when MaxSpread is executed, the two following statements hold:

[6] Recall that k is the maximum degree with communication range r. The proposed protocols use also a communication range of $2r$. The value $19(k+1)$ is an upper bound on the maximum degree with this range.

– If node i has the largest counter global-clock in the network at slot t, then i has the largest global-clock in the network at slot $t+1$.
– If $(i,j) \in E$ and i has the largest global-clock in the network at slot t, then j will also have the largest global-clock at some slot $t' < t+T$.

From these two statements is straightforward to prove that, no matter the τ-adversary, the system reaches synchronization. Also, from these two statements, it is straightforward to prove that the network is synchronized $D \cdot T$ slots after all the nodes are awake. Since it is known that all the nodes are awakened in τ slots, and nodes do not fail during stabilization, then the network is synchronized in at most $D \cdot T + \tau$ slots after the first node is awake. It is assumed that every node in the network knows the values T, τ, and D. Therefore, every node in the network knows the moment in which the whole network reaches synchronization, which is when its global-clock marks $D \cdot T + \tau$. Hence the following theorem.

Theorem 1. MaxSpread *Protocol solves the synchronization problem under any τ-adversary with synchronization time $D \cdot T + \tau$.*

4.2 Synchronizing Against an ∞-adversary

We present now the protocol ContMaxSpread, designed to solve the synchronization problem against an ∞-adversary. Observe that, due to the nature of an ∞-adversary, any synchronization protocol has to keep sending synchronization messages during all its execution, even after the network has been synchronized. In this way, any new node awakened after the network is synchronized recognizes this fact, and joins the network adopting the common value of the global clock.

Hence, the synchronization protocol ContMaxSpread has two phases, a *synchronization phase,* and an *application phase.* The synchronization phase follows similar principles as the MaxSpread protocol, i.e., the largest global-clock is spread through the network. However, as mentioned above, ContMaxSpread keeps sending synchronization messages in the application phase. The protocol ContMaxSpread sets up a synchronization flag synced to communicate the current synchronization state of the network (from a node's point of view). Roughly speaking, during the first $T_1 = 3n^2 + 2nT$ steps of the synchronization phase, a node listens for messages from the network. That listening part is devoted to provide the node with the current synchronization state of the network. If the network is synchronized when the node wakes up, it will know about it before the listening period is over and, without having to send any message will get synchronized. If that does not happen, during the next $T_2 = 2nT$ steps of the synchronization phase, the node transmits to its neighbors its value global-clock and its synchronization flag synced following `ORC` (as in the MaxSpread protocol). As will be shown, at the end of this subphase the network (and hence the node) has to be synchronized. During the application phase, a node transmits its value global-clock and its synchronization flag synced (perhaps, piggybacked in an application message), but this time, the transmission is done in a round robin fashion, i.e., if the identifier of the node is equal to the value global-clock modulo n, then the node transmits. More details of the ContMaxSpread protocol are presented in Algorithm 1.

Lemma 1. *The global clock of a node u awakened before $T_1 + T_2$ satisfies that either*

– $\mathit{global\text{-}clock}_u(T_1 + T_2) \leq T_2$, *or*
– $\mathit{global\text{-}clock}_u(T_1 + T_2) = \mathit{global\text{-}clock}_x(T_1 + T_2) = T_1 + T_2$.

Proof. Recall that the adversary is restricted so that the network is connected at all times and awake nodes are alive for at least the stabilization time. This means that,

Algorithm 1: ContMaxSpread pseudocode for node v. local-clock is v's hardware clock, $T_1 = 3n^2 + T_2$, $T_2 = 2nT$.

```
1  initialization
2      set global-clock to 0;
3      set synced to false;
4      start tasks 1 and 2 concurrently;
5  task 1
       // synch phase
6      while global-clock < T1 + T2 and synced = false once for each time slot do
7          if ORC(v, local-clock) = transmit and global-clock ≥ T1 then
8              transmit (global-clock,synced) with radius r;
9          increment global-clock;
10     set synced to true;
11     stop task 2;
       // application phase
12     foreach time slot do
13         if global-clock = v then
14             transmit (global-clock,synced) with radius r;
15         increment global-clock mod n;
16 task 2
17     upon reception of (global-clock', synced') from other node do
18         if synced = false then
19             if synced' = true then
20                 set synced to true;
21                 set global-clock to global-clock';
22             else
23                 set global-clock to max{global-clock, global-clock'};
```

up to time $T_1 + T_2$, if node $u \in V$ is awake, there exists some *time ordered path* $x = v_0, v_1, \ldots, v_l = u$ (recall that x is the first node awake) in the network connecting x to u such that $l < n$ and, for all $0 < i \leq l$, $\mathsf{local\text{-}clock}_{v_{i-1}}(t) \geq \mathsf{local\text{-}clock}_{v_i}(t)$. We call the *distance* from u to x as the smallest number of edges of any of these time ordered paths. Since all the time steps, and hence global clocks, considered in this proof are smaller than T_1+T_2, then no node fails, and all awake nodes are in the synchronization phase of the algorithm, have a time ordered path to x, and have $\mathsf{synced} = false$.

We show that a node u that at time $T_1 + T_2$ has a global clock different from x's must have $\mathsf{global\text{-}clock}_u(T_1 + T_2) \leq T_2$. Let us consider a node u awakened before T_1+T_2 and whose global clock at that time is $\mathsf{global\text{-}clock}_u(T_1+T_2) < T_1+T_2$. A node that is awakened before T_1 and whose distance to x is d has the same global clock as x by time $T_1 + d \cdot T$. (Broadcast time in a network of awakened nodes, see Lemma 2 in Appendix Section B for details.) Thus, given that the distance is always less than n, a node awakened before T_1 has the the same global clock as x by time $T_1+nT \leq T_1+T_2$. Therefore, to complete the proof, it remains to consider the case where u was awakened at some time within the global-time interval $[T_1, T_1 + T_2)$. To prove the claim in that case, it is enough to prove that $\mathsf{global\text{-}clock}_u(T_1 + T_2) = \mathsf{local\text{-}clock}_u(T_1 + T_2)$ because, given that u did not wake up before T_1, it holds that $\mathsf{local\text{-}clock}_u(T_1 + T_2) \leq T_2$.

Let us assume, by way of contradiction, that u has $\mathsf{global\text{-}clock}_u(T_1 + T_2) \neq \mathsf{local\text{-}clock}_u(T_1 + T_2)$. This means that u has received a message (Line 17) before time $T1 + T2$ with a field $\mathsf{global\text{-}clock}'$ larger than its own global clock, and has adopted it in Line 23. From Line 7, the value $\mathsf{global\text{-}clock}'$ is received because some node v had $\mathsf{global\text{-}clock}_v(t) = \mathsf{local\text{-}clock}_v(t) \geq T_1$ at some time $t < T_1 + T_2$, and transmitted this value (using schedule ORC). Let us denote the propagation path (not necessarily time ordered) of this value before reaching u as $v = q_0, q_1, ..., q_s = u$.

From $t < T_1 + T_2$, $\mathsf{local\text{-}clock}_v(t) \geq T_1$, and $T_1 > T_2$, it is derived that v was awakened before T_1. Let d be the distance from v to x. Then, node v has the same global clock as x by time $T_1 + d \cdot T$. (From Lemma 2 in Appendix Section B.) Then, starting at time $T_1 + d \cdot T$, v has transmitted using the ORC schedule the same global

clock as x. The condition of Line 6 guarantees that it does this for at least $(2n-d)T$ steps.

Returning to the path $v = q_0, q_1, ..., q_s = u$, we have that q_1 must have received (and hence was awake) some message from $v = q_0$ (in particular, the global clock that was later propagated as global-clock$'$) before $T_1 + d \cdot T$. Furthermore, q_1 has received from v a message with the global clock of x by $T_1 + (d+1)T$. Applying the same argument, we conclude that q_2 received (and hence was awake) from q_1 before $T_1 + (d+1)T$ and has received the global clock of x by time $T_1 + (d+2)T$. Inductively, $q_s = u$ has received the global clock of x by time $T_1 + (d+s)T$. Since $d + s < 2n$, this mean that u has the same global clock as x by time $T_1 + T_2$, which is a contradiction.

Theorem 2. ContMaxSpread *protocol solves synchronization problem under any* ∞*-adversary with synchronization time* $T_1 + T_2$*, where* $T_1 = 3n^2 + 2nT$ *and* $T_2 = 2nT$*.*

Proof (Proof sketch). To prove the theorem, it is enough to prove that, at any global time step $t \geq T_1 + T_2$, any node in the network is either synchronized with x, or it is still in the listening part of the synchronization phase. The proof of the following claim is left to Section A in the Appendix for brevity.

Claim. For any node $v \in V$ and any time step $t \geq T_1 + T_2$, it takes at most $3n^2$ time steps for v to have the global time (be synchronized), even under failures (as defined in Section 2), unless v goes back to sleep before.

Lemma 1 shows that at global time $T_1 + T_2$ a global-clock in the network is either synchronized with x's global clock, or its value is smaller than T_2, which is $3n^2$ time steps smaller than T_1. Consequently, at global time $T_1 + T_2$, every node who is transmitting messages does it with x's global clock. Then, any node with global clock smaller than T_2 receive x's global clock before its own global clock reaches the value T_1. Finally, due to the same reason, if a node is awakened at global time $t \geq T_1 + T_2$, before its local clock reaches the value $3n^2$, it receives a message with x's global clock. Then, that node is synchronized without transmitting itself in the synchronization phase.

5 Communication Scheme

In this section, we show how to solve DRC for τ- and ∞- adversaries. Both protocols are algorithmically similar and can be broadly described for each node $v \in V$ as follows. Upon waking up, v runs three phases: synchronization, coloring, and application. During the first phase, v synchronizes itself (as defined in Section 2.2) with a node that woke up first in the network. During the second phase, v chooses a color that has not been chosen by any neighboring node. Finally, by mapping colors to time slots (thanks to the global synchronization achieved), the application phase of i corresponds to its stable phase. Given that the color chosen by v is unique within radius $2r$ of v, but the application messages are transmitted with radius r, all nodes within distance r of v receive v's application messages.

Moving to how do we implement each phase, synchronization is implemented using the protocols of Section 4 for the τ- and ∞- adversaries respectively. The coloring phase, on the other hand, is implemented by each node announcing the color chosen so that, by appropriate bookkeeping of the available colors at each node, nodes within distance $2r$ do not choose the same color (avoiding the hidden-terminal problem). To avoid collision of transmissions and simultaneous choice of the same color, taking advantage of the global synchronization achieved in the previous phase, each colored node chooses an available color and announces its choice in a time slot selected in Round-robin fashion according to ID. For the application phase, again thanks to the global synchronization

achieved, each node transmits its application messages in Round-robin fashion, but now according to its color.

For the τ-adversary (see Algorithm 3 in the Appendix), thanks to the inclusion of a τ-long waiting period at the beginning of the synchronization phase, the above described phases are executed synchronously by all nodes in the network. In other words, all nodes in the network finish the synchronization (resp. coloring) phase and begin the coloring (resp. application) phase at the same time. For the ∞-adversary (see Algorithm 4 in the Appendix) on the other hand, new nodes may be woken up while others are already in the coloring or application phases. Thus, control messages have to be sent always to handle these late arrivals. The transmissions corresponding to those control messages are produced in Round-robin fashion according to ID. The coexistence of both types of messages during the application phase is handled by devoting even slots (w.r.t. global time) to control messages and odd slots (w.r.t. global time) to application messages, at the cost of duplicating the time delay.

Regarding the space complexity of these protocols, a node needs to store its own ID ($O(\log n)$ bits) and after the coloring phase one of $O(k)$ colors ($O(\log k)$ bits). Additionally, each node has to keep track of the colors still available ($O(k)$ bits), and maintains a counter that reach a maximum count in $O(kn)$ ($O(\log n)$ bits). Thus, the overall space complexity for each node is $O(k + \log n)$ bits.

The stabilization time, delay, and message complexity of the protocols described can be proved applying the results of Section 4 and standard analysis of the Round-robin algorithms used. We establish those bounds in the following theorems. Further details about the protocols and the analysis can be found in the Appendix.

Theorem 3. *Given a Sensor Network of n nodes, the protocol presented in Section 5 solves the DRC problem under a τ-adversary with stabilization time at most $D{\cdot}T{+}\tau{+}n$, where T is the delay of the* `ORC` *protocol. The delay of this DRC protocol is $19(k{+}1){-}1$ which is asymptotically optimal, and the message complexity is 0 which is optimal.*

Theorem 4. *Given a Sensor Network of n nodes, upon being woken up by a ∞-adversary, the protocol presented in Section 5 solves the DRC problem under an ∞-adversary with stabilization time at most $6n^2 + 4nT + 4n$, where T is the delay of the* `ORC` *protocol. The delay of this DRC protocol is $38(k{+}1){-}1$ and the message complexity is $19(k + 1)/n$, which are both asymptotically optimal.*

References

1. I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cyirci. Wireless sensor networks: A survey. *Computer Networks*, 38(4):393–422, 2002.
2. N. Alon, A. Bar-Noy, N. Linial, and D. Peleg. Single round simulation in radio networks. *Journal of Algorithms*, 13:188–210, 1992.
3. A. Arora, S. Dolev, and M. Gouda. Maintaining digital clocks in step. *Parallel Processing Letters*, 1:11–18, 1991.
4. R. Baldoni, A. Corsaro, L. Querzoni, S. Scipioni, and S. Tucci Piergiovanni. Coupling-based internal clock synchronization for large-scale dynamic distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 99(RapidPosts), 2009.
5. R. Bar-Yehuda, O. Goldreich, and A. Itai. On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization. *Journal of Computer and System Sciences*, 45:104–126, 1992.
6. M. Ben-Or, D. Dolev, and E. Hoch. Fast self-stabilizing byzantine tolerant digital clock synchronization. In *Proc. 27th Ann. ACM Symp. on Principles of Distributed Computing*, 2008.
7. A. De Bonis, L. Gąsieniec, and U. Vaccaro. Generalized framework for selectors with applications in optimal group testing. In *Proc. of 30th Intl. Colloquium on Automata Languages and Programming*, pages 81–96, 2003.

8. B. Chlebus, L. Gąsieniec, A. Gibbons, A. Pelc, and W. Rytter. Deterministic broadcasting in unknown radio networks. *Distributed Computing*, 15:27–38, 2002.
9. B. Chlebus, L. Gąsieniec, D. Kowalski, and T. Radzik. On the wake-up problem in radio networks. In *Proc. of 32nd Intl. Colloquium on Automata Languages and Programming*, volume 3580 of *Lecture Notes in Computer Science*, page 347359. Springer-Verlag, Berlin, 2005.
10. B. Chlebus, L. Gąsieniec, A. Lingas, and A. Pagourtzis. Oblivious gossiping in ad-hoc radio networks. In *Proc. of 5th Intl. Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 44–51, 2001.
11. B. Chlebus, L. Gąsieniec, A. Östlin, and J. M. Robson. Deterministic radio broadcasting. In *Proc. of 27th Intl. Colloquium on Automata Languages and Programming*, 2000.
12. B. Chlebus and D. Kowalski. A better wake-up in radio networks. In *Proc. 23rd Ann. ACM Symp. on Principles of Distributed Computing*, 2004.
13. M. Chrobak, L. Gąsieniec, and D. Kowalski. The wake-up problem in multi-hop radio networks. In *Proc. of the 15th Ann. ACM-SIAM Symp. on Discrete Algorithms*, 2004.
14. M. Chrobak, L. Gąsieniec, and W. Rytter. Fast broadcasting and gossiping in radio networks. In *Proc. of the 41st IEEE Ann. Symp. on Foundations of Computer Science*, 2000.
15. A. Clementi, A. Monti, and R. Silvestri. Selective families, superimposed codes, and broadcasting on unknown radio networks. In *Proc. of the 12th Ann. ACM-SIAM Symp. on Discrete Algorithms*, 2001.
16. A. Czumaj and W. Rytter. Broadcasting algorithms in radio networks with unknown topology. In *Proc. of the 44th IEEE Ann. Symp. on Foundations of Computer Science*, 2003.
17. D. Dolev and E. Hoch. On self-stabilizing synchronous actions despite byzantine attacks. In *Proc. of the 21st International Symposium on Distributed Computing*, pages 193–207, 2007.
18. S. Dolev. Possible and impossible self-stabilizing digital clock synchronization in general graphs. *Real-Time Systems*, 12(1):95–107, 1997.
19. S. Dolev and J. Welch. Self-stabilizing clock synchronization in the presence of byzantine faults (abstract). In *Proc. 14th Ann. ACM Symp. on Principles of Distributed Computing*, page 256, 1995.
20. S. Dolev and J. Welch. Wait-free clock synchronization. *Algorithmica*, 18:486–511, 1997.
21. Shlomi Dolev, Seth Gilbert, Rachid Guerraoui, Fabian Kuhn, and Calvin Newport. The wireless synchronization problem. *Proc. 28th Ann. ACM Symp. on Principles of Distributed Computing*, Aug 2009.
22. A. Dyachkov and V. Rykov. A survey of superimposed code theory. *Problems of Control and Information Theory*, 12(4), 1983.
23. M. Farach-Colton, R. J. Fernandes, and M. A. Mosteiro. Bootstrapping a hop-optimal network in the weak sensor model. *ACM Transactions on Algorithms*, 5(4):1–30, 2009.
24. A. Fernández Anta, M. A. Mosteiro, and C. Thraves. Deterministic communication in the weak sensor model. In *Proc. 11th Conf. On Principles Of Distributed Systems*, pages 119–131, 2007.
25. A. Fernández Anta, M. A. Mosteiro, and C. Thraves. Deterministic recurrent communication in restricted sensor networks. manuscript, 2009.
26. M. G. Gouda and T. Herman. Stabilizing unison. *Information Processing Letters*, 35(4):171–175, 1990.
27. T. Herman and S. Ghosh. Stabilizing phase-clocks. *Information Processing Letters*, 54(5):259–265, 1995.
28. E. Hoch, D. Dolev, and A. Daliot. Self-stabilizing byzantine digital clock synchronization. In *Proc. of the 8th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 350–362, 2006.
29. P. Indyk. Explicit constructions of selectors and related combinatorial structures, with applications. In *Proc. of the 13th Ann. ACM-SIAM Symp. on Discrete Algorithms*, 2002.
30. D. R. Kowalski. On selection problem in radio networks. In *Proc. 24th Ann. ACM Symp. on Principles of Distributed Computing*, pages 158–166, 2005.
31. E. Kushilevitz and Y. Mansour. An $\Omega(D \log(N/D))$ lower bound for broadcast in radio networks. *SIAM Journal on Computing*, 27(3):702–712, 1998.

32. D. Liu and M. Prabhakaran. On randomized broadcasting and gossiping in radio networks. *Lecture Notes in Computer Science*, 2387:340–349, 2002.
33. P. Rentala, R. Musumuri, U. Saxena, and S. Gandham. Survey on sensor networks. http://citeseer.nj.nec.com/479874.html.
34. L. G. Roberts. Aloha packet system with and without slots and capture. *Computer Communication Review*, 5(2):28–42, 1975.

A Time of Synchronization after $T_1 + T_2$

Proof. Consider a node v that is awake in $[t, t+3n^2)$ but does not have the global time at time $t \geq T_1 + T_2$. Due to the connectivity conditions, until v becomes synchronized, within each consecutive $3n$ steps a new node must be synchronized: at most n steps to synchronize some non-synchronized node, at most n steps when those newly synchronized nodes may maintain connectivity without synchronizing other nodes, and, if all the synchronized nodes that are neighbors of non-synchronized nodes fail before synchronizing any of them, at most n further steps to synchronize at least one new node that has to awake to maintain connectivity, which is possible because at least one synchronized (x) node must stay awake. Due to failures, the same node may be re-synchronized, but not more than twice during the period $[t, t + T_1 + T_2)$ due to the stable-time requirement, and the above cost already accounts for both synchronizations. Thus, the claim follows.

B Time of Broadcast

Lemma 2. *A node u that is awakened before T_1 and whose distance to x is d has the same global clock as x by time $T_1+d\cdot T$, i.e., $\mathsf{global\text{-}clock}_u(T_1+d\cdot T) = \mathsf{global\text{-}clock}_x(T_1+d\cdot T)$.*

Proof. From the condition of Line 7, at time T_1 node x starts transmitting with the schedule defined by `ORC`, and will do so for at least $T_2 = 2nT$ steps (from the condition of Line 6). Consider a time ordered path $x = v_0, v_1, \ldots, v_d = u$ from x to u. Since the delay of `ORC` is T, node v_1 will receive from x before step T_1+T. When that happens, v_1 updates its global-clock variable as shown in Lines 19 to 23. Hence, $\mathsf{global\text{-}clock}_{v_1}(T_1 + T) = \mathsf{global\text{-}clock}_x(T_1 + T)$. Then, from the condition of Line 7, at time $T_1 + T$ node v_1 starts transmitting with the schedule defined by `ORC`, and will do so for at least $(2n - 1)T$ steps (from the condition of Line 6). A simple induction shows that node v_i, $1 < i \leq d$, will satisfy $\mathsf{global\text{-}clock}_{v_i}(T_1 + iT) = \mathsf{global\text{-}clock}_x(T_1 + iT)$. The claim follows since $v_d = u$.

C Proof of Theorem 3

Proof. First, we show correctness and time efficiency of the stabilization phase. (Refer to Algorithm 3.) According to the failure model assumed (Section 2), nodes do not fail during the stabilization phase. Hence, correctness and time efficiency of the stabilization phase are studied in a scenario without failures. For any node, the synchronization phase lasts at most $D \cdot T + \tau$ steps. Furthermore, it was proved in Theorem 1 that within $D\cdot T+\tau$ steps after the first node is woken up, all nodes have been synchronized. I.e., all nodes have the same global-clock and synced values. Thus, all nodes finish the synchronization phase and start the coloring phase at the same time (when the condition in Line 8 of Algorithm 3 is false). After that, the coloring phase takes exactly n further steps (see Algorithm 3 Line 13) yielding a total stabilization time of $D\cdot T+\tau+n$ as claimed.

In order to prove that the coloring obtained is correct for the underlying graph G (where only edges of length at most r are included), consider a disk D_v of radius $2r$ centered in any node $v \in V$, and notice that 19 disks of diameter r are sufficient to completely cover D_v. Then, given that k is the maximum degree of G, there are at most $19(k + 1)$ nodes in D_v. So, $19(k + 1)$ colors are enough to ensure that all nodes in D_v can choose a different color. Given that nodes choose a color in Round-robin

fashion according to ID (thanks to the global synchronization achieved), after the n steps of the coloring phase all nodes have chosen a color.

Defining some ordering among the $19(k+1)$ colors, the coloring provided defines a transmission schedule such that every node receives from each of its neighboring nodes every $19(k+1)$ slots. Thus, the DRC problem is solved with delay $19(k+1)-1$, which is asymptotically optimal given that, in the worst case, all nodes have k neighbors and must receive from each of them. Given that all transmissions of every node are received by all of its neighbors within radius r, the message complexity is 0, which is optimal. Finally, given that a τ-adversary cannot recover crashed nodes, then failures do not affect the recurrence on the communication protocol.

D Proof of Theorem 4

Proof. Again, due to the failure model assumed (Section 2), nodes do not fail during the stabilization phase, which in this case lasts until the coloring is finished. Hence in the first part of the proof, we consider a scenario without failures. For the sake of clarity, assume first that all nodes are woken up simultaneously. Then, the claim can be proved along the same lines of Theorem 3. (Refer to Algorithm 4.) For any node, the synchronization phase lasts at most $6n^2 + 4nT$ steps. Furthermore, it was proved in Theorem 2 that within $3n^2+4nT$ steps after a node is woken up (in this case $6n^2 + 4nT$ due to the duplication of slots after the synchronization phase), it becomes synchronized (w.r.t. global time). I.e. all nodes have the same global-clock and synced values. Thus, all nodes finish the synchronization phase and start the coloring phase at the same time (when the condition in Line 8 of Algorithm 4 is false). After that, the coloring phase takes at most $4n$ further steps (see Algorithm 4 Line 14), a waiting period of $2n$ steps (the reason for this will be clear soon) followed by a period of $2n$ steps when nodes choose colors in a Round-robin fashion using only even steps (w.r.t. global time). Nodes leave the coloring phase as soon as they get colored, but the whole $2n$ steps are needed for the node with ID n. Thus, the total stabilization time is at most $6n^2 + 4nT + 4n$ as claimed.

The correctness of the coloring obtained can be proved as in Theorem 3. Defining some ordering among the $19(k+1)$ colors, the coloring provided defines a transmission schedule. Given that only odd steps are used for application messages, every node receives from each of its neighboring nodes every $38(k + 1)$ slots. Thus, the delay of this protocol is $38(k + 1) - 1$ as claimed. Regarding the message complexity, all transmissions of every node are received by all of its neighbors, but now only the transmissions produced in the odd steps correspond to application messages. For any node $v \in V$ running the application phase, the rate of transmissions of v due to control messages is $1/2n$ (see Algorithm 4). Whereas the rate of transmissions of v due to application messages is $1/(38(k + 1))$. Then, the message complexity is $19(k + 1)/n$, which is asymptotically optimal because $k < n$.

To complete the proof, we consider now the impact of a node $v \in V$ that is woken up "late" with respect to the first node woken up in the network (global time). As shown in the proof of Theorem 2, v is globally-synchronized seamlessly (before v produces any transmission) thanks to the waiting period of $6n^2 + 2nT$ steps, as long as the already synchronized nodes keep transmitting the global time forever as done in Lines 10, 16, and 21 in Algorithm 4. Furthermore, it was proved in the same theorem that within at most $6n^2 + 4nT$ steps after v is woken up, it becomes synchronized (w.r.t. global time). After becoming synchronized, v will produce transmissions in Round-robin fashion according to the protocol avoiding any collisions. After entering the coloring phase, thanks to the initial waiting period of $2n$ steps, v updates properly its set of available colors after receiving transmissions of all synchronized neighbors within

distance $2r$. Thus, after at most $4n$ steps v has been colored properly and has announced its color choice. Thus, the claimed stabilization time, delay, and message complexity also holds for v. Finally, notice that a node that fails and then rejoins the network can be considered as a node that is woken up "late". Consequently, failures do not affect the behavior of the communication protocol.

Algorithm 2: MaxSpread pseudocode for node v.

```
1  initialization
2      set global-clock to 0
3      set synced to false
4      start tasks 1 and 2 concurrently
5  task 1
       // synch phase
6      while global-clock < D · T + τ once for each time slot do
7          if ORC(v, local-clock) = transmit and global-clock ≥ τ then
8              transmit (global-clock) with radius r
9          increment global-clock
10     set synced to true
11     stop task 2
       // application phase
12     foreach time slot do
13         increment global-clock mod n
14 task 2
15     upon reception of (global-clock′) from other node do
16         set global-clock to max{global-clock, global-clock′}
```

Algorithm 3: DRC pseudocode for node v under τ-adversary. local-clock is the hardware node clock. T is the delay of the ORC protocol.

```
1  initialization
2      set global-clock to 0
3      set color to null
4      set synced to false
5      set available-colors to {0, 1, ..., 19(k + 1) − 1}
6      start tasks 1 and 2 concurrently
7  task 1
       // synchronization phase
8      while global-clock < D · T + τ and synced = false once for each time slot do
9          if ORC(v, local-clock) = transmit and global-clock ≥ τ then
10             transmit (global-clock,color,synced) with radius 2r
11         increment global-clock
12     set synced to true
       // coloring phase
13     while global-clock < D · T + τ + n once for each time slot do
14         if global-clock ≡ v(mod n) then
15             set color to one of the available-colors
16             transmit (global-clock,color,synced) with radius 2r
17         increment global-clock
18     stop task 2
       // application phase
19     foreach time slot do
20         if global-clock = color then
21             transmit (application-message) with radius r
22         increment global-clock mod 19(k + 1)
23 task 2
24     upon reception of (global-clock′, color′, synced′) from other node do
25         set available-colors to available-colors − {color′}
26         if synced = false then
27             if synced' = true then
28                 set synced to true
29                 set global-clock to global-clock'
30             else
31                 set global-clock to max{global-clock, global-clock′}
```

Algorithm 4: DRC pseudocode for node v under ∞-adversary. local-clock is the hardware node clock. T is the delay of the ORC protocol.

```
1  initialization
2      set global-clock to 0
3      set synced to false
4      set color to null
5      set available-colors to {0, 1, ..., 19(k + 1) − 1}
6      start tasks 1 and 2 concurrently
7  task 1
       // synchronization phase
8      while global-clock < 6n^2 + 4nT and synced = false once for each time slot do
9          if ORC(v, local-clock) = transmit and global-clock ≥ 6n^2 + 2nT then
10             transmit (global-clock,color,synced) with radius 2r
11         increment global-clock
12     set synced to true
       // coloring phase
13     while color = null once for each time slot do
14         if global-clock ≡ 2v(mod 2n) and global-clock ≥ 6n^2 + 4nT + 2n then
15             set color to one of the available-colors
16             transmit (global-clock,color,synced) with radius 2r
17         increment global-clock
18     stop task 2
       // application phase
19     foreach time slot do
20         if global-clock ≡ 2v(mod 2n) then
21             transmit (global-clock,color,synced) with radius 2r
22         if global-clock ≡ (2color + 1)(mod 38(k + 1)) then
23             transmit (application-message) with radius r
24         increment global-clock mod 76(k + 1)n
25 task 2
26     upon reception of (global-clock', color', synced') from other node do
27         set available-colors to available-colors − {color'}
28         if synced = false then
29             if synced' = true then
30                 set synced to true
31                 set global-clock to global-clock'
32             else
33                 set global-clock to max{global-clock, global-clock'}
```