# Distributed Algorithms for Sensor Networks

By Christoph Lenzen[1] and Roger Wattenhofer[2]

[1] *School of Engineering and Computer Science, Hebrew University of Jerusalem Edmond Safra Campus, Givat Ram, 91904 Jerusalem, Israel*

[2] *Computer Engineering and Networks Laboratory, ETH Zurich Gloriastrasse 35, 8092 Zurich, Switzerland*

Distributed algorithms are an established tool for designing protocols for sensor networks. In this article we discuss the relation between distributed computing theory and sensor network applications. Along the way, we present a few basic and illustrative distributed algorithms.

**Keywords: local algorithms, topology of sensor networks, interference models, maximal independent set, vertex colouring, clock synchronization**

## 1. Introduction

Many of today's computing and communication systems are *distributed*—systems that are composed of autonomous computational entities that communicate with each other, usually by passing messages. Distributed systems encompass a variety of applications. The most obvious distributed system is the Internet itself, and various of its applications such as the domain name system, peer-to-peer file sharing, or cloud computing. On the other hand distributed systems also exist on a smaller scale, such as multi-core processors or field-programmable gate arrays. Indeed, if we expand our horizon beyond technical systems, also the brain or society† are in some sense distributed systems, with neurons and human beings taking the roles of the autonomous entities, respectively. Despite their obvious differences, all these distributed systems have in common that many entities may concurrently be active in the system. These entities have certain degrees of freedom: they may have their own hardware, their own data or code, and sometimes their own independent task. Nevertheless, they share common resources and information and, in order to solve a problem that concerns all or a subset of the entities, coordination is necessary.

A sensor network is an ensemble of *nodes*, where each node by itself is a tiny computer that may gather input data from its sensors and is capable to communicate with other nodes by radio. As radio signal strength deteriorates with distance quickly, it is generally infeasible to have all nodes talk to each other (or to a base station) directly. Instead, nodes may have to use other nodes as intermediate relays in order to communicate. This situation can be modelled by a graph, where the node set is given by the sensor nodes (plus possibly the base station); an edge between two nodes indicates that they are capable to directly exchange messages by

† The relation between distributed systems and orthodox monolithic systems is sometimes compared to the relation between sociology and psychology.

radio communication. As such, a sensor network is an out-of-the-book distributed system (Peleg, 2000).

Distributed algorithms are at the theoretical foundations of distributed systems – we study how the nodes of a distributed system should orchestrate their communication and computation in order to solve a given task efficiently. Moreover, we are interested in impossibility results and lower bounds as they demonstrate what is not possible with a distributed algorithm. The area as a whole is known as *distributed computing.*

In a large distributed system, no node can have a *global* view of the entire system at any time. Instead, every node has a *local* view of the system only, and has to base its decisions on this local information. Many tasks can be solved completely locally,† for instance, a node can figure out the lowest measured temperature in its neighbourhood by simply communicating with its neighbours. Many other tasks are inherently global, for instance, identifying the minimum temperature in the whole sensor network. To solve such global problems, some information must traverse across the entire network graph. Among other things, researchers in distributed computing want to know what type of tasks are local, and what type of tasks are global. As we will see, many interesting network coordination tasks that are relevant in sensor networks are neither local nor global, but somehow in between.

There are a number of reasons why distributed algorithms are highly appealing for the design of sensor network protocols. First of all, we want to avoid that memory constrained sensor nodes have to store lots of information about far-away nodes. Thus, in fact it is an advantage if nodes compute the local part of a problem's solution only. More importantly, restricting the distance from which information is collected implies that (*i*) algorithms are fast, as communication is typically the most time consuming part, and (*ii*) changes in input or topology can be dealt with locally, i.e., merely a part of the nodes need to recompute their output. In other words, in a large system, faults or reconfigurations interfere with a small fraction of the system only. This brings us to another impressive capability of distributed systems: If well-crafted, they can function correctly from a global perspective even if some of the nodes are crashing, show erroneous behaviour, or are even maliciously trying to make the system as a whole fail. Distributed algorithms are crucial in order to exploit this potential, as any part of an algorithm that runs on a single node will fail if the respective node crashes. Even worse, if the node does not simply crash, it may corrupt the entire network with wrong information. Last but not least, as solutions are derived from as little knowledge on the overall state of the network as possible, many distributed algorithms turn out to be simple and elegant. While elegance is not necessarily a design issue for sensor networks, simplicity is, in particular considering the limited computational power, memory, and energy of sensor nodes.

Overall, we believe that distributed algorithms can offer a lot to designers of sensor networks. However, even though sensor networks *seem* to be out-of-the-book distributed systems in theory, in practice quite a few difficulties are between an abstract distributed algorithm and its implementation on a sensor node. In this article, we will highlight some of the main differences between a real-world sen-

† We recommend a survey by Suomela (2011) on strictly local problems, i.e., problems which can be solved by looking at the graph topology and inputs up to a constant number of hops.

sor network and an idealised distributed system. We will spotlight pitfalls, success stories, and open problems researchers encountered—or still do encounter—while trying to close these gaps between theory and practice. We will start our tour with a brief introduction to abstract distributed algorithms. Subsequently, we will gradually proceed to more realistic communication models. Along the way, we will briefly present some basic techniques that we believe to illuminate the way of thinking of distributed computing as well as the beauty of the resulting solutions.

At this point, we emphasise that this article does not make any attempt to survey the state-of-the-art in distributed computing. Also, we will bias our choice of the discussed topics towards the ones we are familiar with, i.e., we do not attempt to give a representative cross section of distributed computing.

## 2. Distributed Algorithms

For many decades the study of distributed network algorithms was treated like a pure research topic. Maybe too pure for the real world of messy distributed systems. As such the area fell asleep in the 1990's. The advent of sensor (and its close relatives such as ad hoc or mesh) networks in the current millennium reignited interest in the area again, as all of a sudden there were applications matching the local communication model well. To get a better understanding for this, let us first have a closer look at a standard model used in distributed algorithms.

As mentioned before, a distributed system is modelled as a graph $G = (V, E)$, whose set of nodes $V$ (with $n := |V|$) are the computing devices and whose set of edges $E$ define which nodes may communicate with each other directly. One typically focuses on the *locality* of a problem, that is, on the maximal (hop) distance from which nodes must receive information as function of the number of nodes $n$. Theory in its most pure form thus commonly studies the *local, synchronous message passing model.* More concretely, computation proceeds in rounds, where in each round, all nodes concurrently exchange messages. In other words, any synchronous distributed algorithm looks like this:

**Algorithm 1:** General Skeleton of a Distributed Algorithm

```
Each node v performs the following actions concurrently with all other nodes:
repeat
    send (possibly different) messages to neighbours
    receive neighbours' messages
    perform some local computations to prepare for the next round
until termination;
```

When studying distributed algorithms, we care mostly about running time; the running time of a synchronous distributed algorithm is measured by the number of synchronous communication rounds, i.e., how often the communication-computation loop is executed until the nodes terminate.

### *MIS, a Classic Problem in Distributed Computing*

For the sake of concreteness, let us study an example, the so-called maximal independent set (MIS) problem. As mentioned before, in a distributed system all nodes

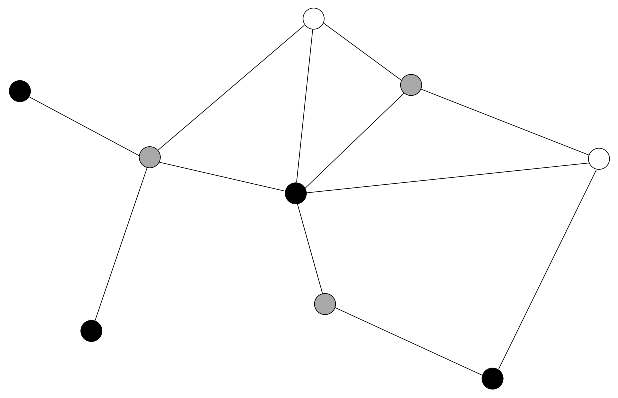
Figure 1. Partitioning of a graph's node set into independent sets. The sets of grey and black nodes are maximal independent sets.

are capable of acting simultaneously. However, even if tasks are local in the sense that it does not matter what distant nodes are doing, frequently one needs to avoid that two neighbours take action at the same time. For instance, if two neighbouring nodes try to transmit at the same time, it is likely that both messages cannot be received due to wireless interference. On the other hand, we want to achieve a large amount of concurrency. In particular, for each node at least *some* neighbour should be able to perform operations. These requirements can be formalised as follows.

**Definition 1** (Maximal Independent Set)**.** *Given a graph $G = (V, E)$, an* independent set *is a subset $S \subseteq V$ of the nodes such that no two adjacent nodes are in $S$, i.e., for any edge $\{v, w\} \in E$, we have that $v \notin S$ or $w \notin S$. A* maximal independent set (MIS) *$S$ is an independent set satisfying that $S \cup \{v\}$ is not independent for any $v \in V \setminus S$. See Figure 1 for an example.*

The MIS is a basic *symmetry breaking* problem that is trivial for a centralised algorithm. Starting with $S := \emptyset$, one just goes over the nodes one by one and adds any node without neighbour in $S$ to $S$. However, if nodes act concurrently, they can join $S$ in the same round in order to be much faster. The only thing we need to ensure is that no two adjacent nodes enter $S$ in the same round. Assuming that nodes have unique identifiers, the trivial sequential MIS solution translates to a simple distributed algorithm. For brevity, denote in the following by $\mathcal{N}_v := \{w \in V \mid \{v, w\} \in E\}$ the neighbourhood of $v \in V$.

When executing Algorithm 2, any still active node that has a locally maximal identifier will join the independent set in the next (even) round. Because there can never be two neighbours that have a locally maximal identifier, this can never cause a conflict. All neighbours of such nodes terminate in the subsequent round, making sure that we will also have no conflicts later on. Note that it is safe for these nodes to terminate, as they now have at least one neighbour in the independent set, i.e., they cannot be part of any independent set containing the nodes that already joined. The running time of this algorithm is given by the longest descending chain of identifiers. This is at most $D + 1$, where $D$ denotes the diameter of the graph, i.e., the maximal length of a shortest path between any pair of nodes.†

† Without loss of generality, we always assume $G$ to be connected.

**Algorithm 2:** Naive MIS

```
input  : each node $v \in V$ has a unique identifier $id(v)$
output: each node $v$ knows whether it is in the MIS
Each node v performs the following actions concurrently with all other nodes:
send $id(v)$ to neighbours
receive neighbours' identifiers
Memorise neighbours with higher priority as $A := \{w \in \mathcal{N}_v \mid id(w) > id(v)\}$
repeat
    if $A = \emptyset$ then
        send 'join' to neighbours $\mathcal{N}_v$
        join MIS and terminate
    if receive 'join' message then
        send 'not join' to neighbours $\mathcal{N}_v$
        do not join MIS and terminate
    receive 'not join' messages
    $A = A \setminus \{w \in \mathcal{N}_v \mid$ received 'not join' from $w\}$
until termination;
```

Unfortunately, $D$ might be large—a most simple network is the line (also known as linked list, or chain) network, which already has diameter $n-1$! This is not exciting, as in a bad case it is no better than a sequential solution—it seems that a simple locally definable problem such as finding an MIS should have a solution without a global causality chain.

### *Searching for a Fast Distributed MIS Algorithm*

Defying this intuition, until today no algorithms are known that are *always* fast. Even if one does not care about the size of messages, the fastest known solution has running time $2^{\mathcal{O}(\sqrt{\log n})}$ (Panconesi & Srinivasan, 1996). 'Interesting' graphs tend to have a diameter that is much smaller. While some algorithms are efficient if the graph satisfies certain properties, ideally we would like to have a solution that works nicely for *any* network. For distributed algorithms, in many cases this becomes difficult because for any fixed course of action, there is a particular example where it is bad. And because nodes initially are aware of a small part of the graph only, it may take a lot of time until they learn about this. Oftentimes, this obstacle can, at least partly, be overcome by means of *randomisation.* Randomised distributed algorithms follow the same rules as *deterministic* ones, with the single exception that nodes may base their decisions also on 'coin tosses'. Again, let us illustrate this concept by a plausible example.

Algorithm 3 is a variant of a long-known family of algorithms (Luby, 1986; Alon *et al.*, 1986; Israeli & Itai, 1986; Métivier *et al.*, 2009). Essentially, it is identical to Algorithm 2, with the decisive difference that in every iteration, the 'identifiers' are picked anew at random. If the random numbers have a logarithmic size, the probability that two numbers are equal is negligible. It can be shown by a fairly simple argument that in each iteration, the expected number of edges adjacent to terminating nodes is a constant fraction of the total number of remaining edges. Since this holds for *any* graph, in particular the subgraphs induced by the active nodes in each iteration, this implies an expected running time of $\mathcal{O}(\log n)$. One can

**Algorithm 3:** Randomised MIS

```
output: each node $v \in V$ knows whether is in the MIS
Each node v performs the following actions concurrently with all other nodes:
repeat
    let $r(v)$ be a randomly chosen number
    send $r(v)$ to neighbours
    receive random numbers from all (non-terminated) neighbours $\mathcal{N}_v$
    if $r(v) > r(w)$ for all $w \in \mathcal{N}_v$ then
        send 'join' to $\mathcal{N}_v$
        join MIS and terminate
    if receive 'join' message then
        send 'not join' to $\mathcal{N}_v$
        do not join MIS and terminate
until termination;
```

show that in fact the algorithm will terminate in $\mathcal{O}(c \log n)$ rounds also *with high probability (w.h.p.)*, that is, with probability $1 - 1/n^c$. This is a powerful result, as even for a moderate constant $c$ it is absurdly unlikely that the algorithm does not terminate within $\mathcal{O}(c \log n)$ rounds.†

## *MIS and Colouring*

Apart from the vague motivation given above, why should we be interested in an MIS algorithm in the first place? One possible answer to this question is that it is useful in computing different, perhaps more attractive structures. For instance, an MIS is always a *dominating set* (i.e., in each node's neighbourhood there is at least some node from the set). Typically one is interested in small dominating sets (as one could trivially take all nodes), but there are interesting graphs where an MIS is not much larger than a smallest, a so-called *minimum dominating set (MDS)*. Another striking example is colouring.

**Definition 2** (Colouring)**.** *Given a graph $G = (V, E)$, a $k$-*colouring *of $G$ is a function $c : V \to \{1, \ldots, k\}$ such that for each colour $i \in \{1, \ldots, k\}$, the set $c^{-1}(i)$ is independent. In other words, any two neighbouring nodes $v$ and $w$ do not have the same colour, that is, $c(v) \neq c(w)$. Figure 1 gives an example of a 3-colouring.*

Naturally, a goal is to minimise the number of colours: A $k$-colouring permits that each node acts once in $k$ rounds without neighbours interfering. This is of high interest in sensor networks, as it makes possible to set up a schedule where all nodes may transmit once in a while without causing conflicts. Regrettably, it is however extremely difficult to determine the minimum number of colours required to colour a graph: it is NP-hard to obtain even an estimate that is more precise than a factor $n^{1/7-\varepsilon}$, for any constant $\varepsilon > 0$ (Bellare *et al.*, 1998).

For this reason, it is common to settle for finding a colouring with $\Delta + 1$ or $\mathcal{O}(\Delta)$ colours, where $\delta(v)$ is the degree (the number of neighbours) of node $v$, and $\Delta$ is the maximum degree in the graph. Like computing an MIS, computing a $(\Delta + 1)$-colouring is trivial if approached sequentially, working node by node.

† For example, if $n = 30$ and $c = 10$ it is three times more likely to hit the lottery jackpot *twice in a row* than to see the algorithm not terminating within the stated number of rounds.

There is a close relation between colouring and MIS. From a colouring with $k$ colours, one can derive an MIS in $k$ rounds by incrementally adding nodes of the same colour to the independent set concurrently.

In turn, we can emulate an MIS algorithm on a virtual graph where each node $v$ with degree $\delta(v)$ emulates a cluster of $\delta(v)+1$ virtual nodes called $v_0, v_1, \ldots, v_{\delta(v)}$. All nodes within each cluster are connected by virtual edges. In addition, $v_i$ is connected to $w_i$ if $v$ and $w$ are neighbours in the original graph. By definition of the MIS problem, each cluster $v$ will have exactly one virtual node $v_i$ in the MIS, and the index $i$ of that node determines the colour, i.e., $c(v) := i$.

Despite utilising elaborate and complex techniques, the currently best known deterministic colouring algorithm (Barenboim & Elkin, 2010) remains unsatisfactorily slow. In contrast, plugging in Algorithm 3 into the above transformation yields a simple randomised algorithm with running time $\mathcal{O}(\log n)$ w.h.p. However, as a node $v$ needs to emulate $\delta(v)+1$ many nodes in the virtual graph, which each need to pick and compare random bit strings of size $\Omega(\log n)$ in every iteration, messages would have to contain $\Omega(\Delta \log n)$ bits in the underlying graph in order to emulate the MIS algorithm at full speed. Luckily, we can fix this by simply letting each node choose a free (i.e., not yet occupied by a neighbour) colour from the range $\{1, \ldots, 2\delta(v)\}$ at random, in each round. Since the neighbours may have or claim only one colour each, there are always at least $\delta(v)$ free colours. Therefore, every active node can successfully claim a colour differing from all its neighbours' with probability at least $1/2$. If successful, it can fix this colour as its output, inform its neighbours, and terminate. We again get an algorithm that terminates in time $\mathcal{O}(\log n)$ w.h.p.

## *Distributed Computing Models*

Of course, distributed computing theory has to offer more evolved algorithms. In general, when devising distributed algorithms, one tries to achieve (or balance between) multiple objectives. A main objective is always to guarantee a small running time. Apart from this, we usually want messages to be not too large. A size restriction of $\mathcal{O}(\log n)$ bits per message is the golden standard. Smaller messages often imply that algorithms become restricted, as one cannot encode a node identifier any more in a single message.† In addition, computations should be 'reasonable'. This in particular excludes just collecting the entire topology and inputs at a single (or each) node and solving the whole problem, or dealing with NP-hard (sub)problems. Luckily, small messages imply that nodes do not obtain much information, quite often resulting in computations remaining simple without special consideration. In addition, often the number of messages sent by each node should be small, and memory consumption should be small. Finally, the nodes should need as little knowledge on the network as possible. Ideally, nodes do not know the total number of nodes $n$, the maximum degree $\Delta$, the diameter $D$, or the largest identifier in the network.

As can be seen from this (non-exhaustive) list, distributed computing studies quite a few of the properties that are relevant to sensor networks. However, there is an aspect of distributed computing theory that is of equal, if not higher, significance

† Nevertheless, one can accomplish surprising things. For instance, with a clever trick Algorithm 3 can be implemented in a way such that messages contain merely a constant number of bits!

to practitioners. Apart from studying what is possible, distributed computing also deals with understanding what is *im*possible.

### *Lower Bounds*

For instance, is there a (randomised) MIS algorithm that is substantially better than Algorithm 3? The answer is 'no'. For the MIS problem, and for many related problems, a lower bound of $\Omega(\sqrt{\log n})$ (or alternatively $\Omega(\log \Delta)$) has been established (Kuhn *et al.*, 2010). In other words there is *no* (randomised) distributed algorithm that can compute an MIS in less than $\Omega(\sqrt{\log n})$ time, that is, in order to figure out whether to join an MIS, each node must collect at least the information in its $\Omega(\sqrt{\log n})$-hop neighbourhood. This is certainly surprising, as the MIS problem can be defined strictly locally, and one would not expect such a 'butterfly effect'.

However, again the picture is incomplete. Even if there is no MIS algorithm that can be faster than $\Omega(\sqrt{\log n})$ rounds in *general* graphs, this does not necessarily imply that the same is true for wireless networks. The MIS lower bound construction relies on a peculiar family of bipartite graphs, and one might argue that clearly wireless networks do not exhibit arbitrary connectivity.

## 3. Wireless Connectivity

As we discussed in the previous section, the MIS lower bound does not apply to the connectivity graphs one may encounter in a real wireless network. In this section, we study what kind of graph families are reasonable, and whether they allow for faster distributed algorithms, breaking the lower bound for general graphs.

As we know, the strength of a radio transmission decreases with distance. Thus, it is a key characteristic of wireless (sensor) networks that, unless the network is small, it is unlikely that every node can communicate directly with every other node. On the other hand, it is much more likely that two nearby nodes can communicate to each other directly than it is for two far-away nodes. That is, the connectivity between nodes is fundamentally governed by some kind of *geometry*.

In the early days of algorithmic wireless network research, this geometry was modelled by *unit disc graphs (UDG)*, e.g. Marathe *et al.* (1995); Breu & Kirkpatrick (1998). Let $V$ be a set of nodes, located in the two-dimensional Euclidean plane. In the UDG model we assume that two nodes are adjacent if and only if their Euclidean distance is at most one. If two nodes are farther away, they need to communicate through intermediate relay nodes. The UDG model is too idealistic; in reality, radio communication is never perfectly omnidirectional, and even small obstacles such as plants could change radio connectivity.

A natural generalisation of the UDG is the *quasi unit disc graph (QUDG)*, (Barriére *et al.*, 2003; Kuhn *et al.*, 2008). Again, in a QUDG, the nodes $V$ are located in the plane. All pairs of nodes with Euclidean distance at most $\rho$ for some given $\rho \in (0, 1]$ are adjacent. Pairs with a distance larger than one are never in each other's transmission range. Finally, pairs with a distance between $\rho$ and one may or may not be neighbouring. Indeed, for this 'grey' area in $(\rho, 1]$ there are several possible model options. For instance, one could imagine an adversary choosing for each node pair whether they are in each other's transmission range, or one could apply a connectivity probability distribution depending on the distance. Measurement studies (Ganesan *et al.*, 2002) suggest that in an unobstructed environment, $\rho$ can be modelled as a reasonable constant. In this case, the overhead to make a protocol work in the QUDG model is often bearable.

In contrast, if our sensor network is inside a building, obstructions usually drive the parameter $\rho$ towards zero, as there might be nodes that are physically close but blocked by a wall. And as soon as $\rho$ is small enough, it allows for arbitrary connectivity graphs. However, real-world connectivity graphs are not arbitrary. Although nodes which are close but on different sides of a wall may not be able to communicate, a node is typically highly connected to the nodes which are in the same room, and thus many neighbours of a node are adjacent. In other words, even if there are many obstacles, the total number of neighbours of a node which are not adjacent is likely to be small. This observation is hard to capture with purely geometric models, and motivates more advanced connectivity models such as the *bounded-growth graph (BGG)* model (Kuhn *et al.*, 2005). The BGG model can be defined abstractly, without referring to Euclidean geometry. Again we connect our set of nodes $V$ by edges $E$. As defined before, a set of nodes $S \subset V$ is called independent if no two nodes in $S$ are neighbours, i.e., there is no edge in $E$ connecting two nodes in $S$. Now we define the ball $B_r(v)$ with radius $r$ around a node $v \in V$ as all nodes that can be reached from $v$ by just following at most $r$ edges in $E$. And we ask the set of edges $E$ to be such that the size of any independent set $S$ in ball $B_r(v)$ is polynomially bounded in $r$, i.e. $|S(B_r(v))| \in \mathcal{O}(r^2)$ or $\mathcal{O}(r^3)$. Even though the BGG definition is fairly general, many algorithms that run on a BGG are asymptotically as efficient as algorithms for (Q)UDGs. As such the BGG has become a model of choice when we are interested in connectivity only (Schmid & Wattenhofer, 2006).

### *Wireless Connectivity Simplifies the MIS Problem*

So how much can we improve the running time of MIS or colouring algorithms if we restrict ourselves to (Q)UDGs or BGGs? As it turns out, dramatically! There are algorithms that are super-exponentially better than the lower bound for general graphs. In particular, there is a deterministic algorithm (Schneider & Wattenhofer, 2008) that can compute an MIS in $\mathcal{O}(\log^* n)$ rounds.† Using this MIS algorithm as a base, we can also compute many other interesting structures such as asymptotically optimum colourings or (connected) dominating sets in $\mathcal{O}(\log^* n)$ time.

Unfortunately, the algorithm achieving this surprising result is rather intricate. However, we can present a basic building block (Cole & Vishkin, 1986) if we restrict ourselves to 3-colouring a ring-topology network. In the following, we assume that our nodes are arranged in an oriented circle, i.e. each node knows its clockwise neighbour and counter-clockwise neighbour. The nodes need to pick a colour from $\{1, 2, 3\}$ not conflicting with their neighbours. Starting from the initial identifiers (which are a valid colouring), in each step they reduce the size of the bit string encoding their current colour exponentially. To this end, $v$ compares its colour $c(v)$

† The function $\log^* n$ describes the number of times one needs to apply the logarithm to $n$ until the result is at most one. This 'inverse tower function' grows extremely slowly; it is at most five for any $n$ up to $2^{65,536}$, a value compared to which the estimated number of atoms in the universe looks tiny.

**Algorithm 4:** Ring Colouring

```
input  : each node $v \in V$ has a unique identifier $id(v)$
output: each node $v$ knows its colour $c(v) \in \{1,2,3\}$
Each node $v$ performs the following actions concurrently with all other nodes:
$c(v) := id(v)$
for $\log^* n + \mathcal{O}(1)$ rounds do
    send $c(v)$ to counter-clockwise neighbour $u$
    receive $c(w)$ from clockwise neighbour $w$
    $i :=$ index of least significant differing bit of $c(v)$ and $c(w)$
    set the new colour to $c(v) := 2 \cdot i + c(v)[i]$
repeat
    send $c(v)$ to the two neighbours $\mathcal{N}_v = \{u, w\}$
    receive $c(u), c(w)$ from neighbours $\mathcal{N}_v$
    if $c(v) > \max\{c(u), c(w)\}$ then
        $c(v) := \min\{1,2,3\} \setminus \{c(u), c(w)\}$
        terminate and output $c(v)$
until termination;
```

| | | | |
|---|---|---|---|
| counter-clockwise neighbour $u$: | 1001011 | $\rightarrow$ | 1100 |
| node $v$: | 100101011 | $\rightarrow$ | 101 |
| clockwise neighbour $w$: | 10010101 | $\rightarrow$ | 100 |
| second clockwise neighbour: | 111 | $\rightarrow$ | ? |

Figure 2. An iteration of Algorithm 4. The least significant bit is to the right.

to $c(w)$, the colour of its clockwise neighbour $w$. Starting from the least significant bit, $v$ counts the number of bits until encountering the first difference. Its new colour is given by this—binary encoded—number, plus the differing bit. Since $w$ performs the same operation, they either $(i)$ encode a different index or $(ii)$ append differing bits to their strings. Thus, the resulting colours are not the same, as is the case for $v$ and its other neighbour by the same reasoning (see Figure 2. Since in each step an index in the binary encoded colour $c(v)$ is binary encoded and extended by one bit, after $\log^* n + \mathcal{O}(1)$ iterations the number of remaining colours is constant. At this point, it is sufficient if each node with a locally maximal colour larger than three replaces it in each round by a free colour from the range $\{1,2,3\}$. See Algorithm 4 for pseudo code.

It is interesting to note that both Algorithm 4 and its descendant (Schneider & Wattenhofer, 2008) are asymptotically optimal.† It was shown that computing a 3-colouring or an MIS on a ring takes at least $\Omega(\log^* n)$ time (Luby, 1986; Naor, 1991). Note that a ring is also a (Q)UDG and a BGG. Moreover, it has also been shown that finding an asymptotically optimum (connected) dominating set in a UDG requires $\Omega(\log^* n)$ rounds (Lenzen & Wattenhofer, 2008).

So, with these problems solved in an adequate family of connectivity graphs, are

† In contrast, despite all efforts the (randomised as well as deterministic) asymptotic time complexities for $(\Delta + 1)$-colouring in general graphs still elude us. For colouring, the strongest known lower bound is just the one for rings, a mere $\Omega(\log^* n)$ rounds! Closing this super-exponential gap is among the most prominent open problems in distributed computing.

we equipped with all the tools we need to deal with real networks? Unfortunately, there is one more problem! So far we assumed that nodes can always transmit concurrently. In wireless networks, this is usually not possible, as concurrent transmissions will interfere with each other. Instead, we need to enhance our distributed algorithms by a media access control (MAC) mechanism in order to handle this interference. This is the topic of the next section!

## 4. Wireless Interference

Usually, sensor networks communicate by radio. An immediately evident result is that nodes are *not* able to send a different message to each neighbour at the same time. On the other hand, directed antennas are uncommon, i.e., in principle it is feasible that all neighbours receive a sent message. In other words, we prefer distributed algorithms where nodes transmit the same uniform message to all their neighbours.

A more subtle consequence of the use of radio communication is that the transmission medium is shared: Transmissions are exposed to interference. Concretely, a node $u$ may not be able to correctly receive a message of an adjacent node $v$ because there is a concurrent transmission going on nearby. While for higher layer protocols it may be accurate enough to model interference by having random transmission failures, interference must be a first-class citizen for understanding the basic communication infrastructure.

In some sense, an interference model explains how concurrent transmissions block each other. A signal might for example interfere with itself due to multi-path propagation (e.g., an electromagnetic wave of a direct path cancelling with the wave on a longer path reflecting at an object). Fully understanding these self-interference effects is beyond the scope of this article, and maybe beyond the scope of research in distributed algorithms in general, for some years to come.

Interference because of two or more concurrent transmissions is captured by two classes of models, protocol and physical interference models (Gupta & Kumar, 2000; Schmid & Wattenhofer, 2006). In the *protocol* model, interference is defined using an interference graph $G' = (V, E')$ on the same node set as the connectivity graph $G = (V, E)$. A receiver node $v$ can receive a transmission from node $u$ if and only if $\{u, v\} \in E$ (connectivity) but there is no concurrent transmission by a node $w$ with $\{w, v\} \in E'$ (interference). There have been many proposals how the edge set $E'$ should be defined. Generally $E'$ includes all the edges of $E$, and more. For instance, if $G$ is a UDG, $G'$ may be a UDG as well, however with a larger unit radius. Or $E'$ may include all edges between node pairs that have at most distance $k$ in $G$, i.e., $\{u, v\} \in E'$ if $u \in B_k(v)$. Or $G'$ may be an arbitrary (Q)UDG or BGG.

### *Physical Interference*

All these protocol model definitions have in common that interference ends abruptly, as if there was an invisible wall, and that interference does not sum up, i.e., if there are several concurrent transmissions just outside the interference range, they will not affect a receiver. *Physical* interference models strive for capturing these effects. The relation between the nodes $V$ is given by a distance matrix, either derived from actual distances in a Euclidean space, or by a metric space, or by an arbitrary abstract gain matrix (without direct physical representation). In other words, between any two nodes $\{u, v\}$ we have a distance $d(u, v)$. A transmission of node $u$ with transmission power $P$ will give a signal at receiver $v$ with power $R = P \cdot d(u, v)^{-\alpha}$, where $\alpha$ is the so-called path-loss parameter, often a constant between 2 and 6. Node $v$ can successfully decode the transmission of $u$ if $R$ is at least a factor $\beta$ larger than the interference, defined as the sum of the signal strength produced by all concurrent transmissions. In addition there might also be background noise. All this gives the so-called Signal-to-Interference-plus-Noise-Ratio (SINR) model. Recently, studying physical interference in an algorithmic context has become a hot research topic, e.g., Moscibroda & Wattenhofer (2006); Goussevskaia *et al.* (2009); Fanghänel *et al.* (2009); Kowalski & Rokicki (2010); Kesselheim & Vöcking (2010); Kesselheim (2011); Halldórsson & Mitra (2011); Kantor *et al.* (2011). For first surveys, we refer to Lotker & Peleg (2010); Goussevskaia *et al.* (2010); Fanghänel & Vöcking (2011).

However, most distributed algorithms work in a protocol model. Apart from exceptions, media access is usually done probabilistically. A simple idea is to let nodes transmit randomly, with a probability inversely proportional to the 'competition'. In particular, if each node $v$ transmits with probability $p := 1/\delta_2(v)$, where $\delta_2(v)$ is the highest degree in the two-hop neighbourhood of the interference graph $G'$ of node $v$, we are sure that at least a constant fraction of the transmissions is successful (Abramson, 1970). A major problem is now how to figure out the competition $\delta_2(v)$, as we have a chicken-and-egg problem: In order to communicate successfully we need to know an approximation of $\delta_2(v)$. There has been a flurry of work how to approach the right density, by starting with some probability and eventually converging to the right one, depending on different models, e.g. whether nodes can or cannot tell if multiple nodes try to transmit concurrently (Willard, 1986; Gasieniec *et al.*, 2001; Chrobak *et al.*, 2004; Kuhn *et al.*, 2004; Jurdzinski & Stachowiak, 2005; Czumaj & Rytter, 2006; Cornejo & Kuhn, 2010; Afek *et al.*, 2011). A good benchmark for multi-hop networks is the so-called broadcast problem, where one source node needs to send a message to all the nodes in the network, as it combines the media access question cleverly with an important application (Chlamtac & Kutten, 1985; Bar-Yehuda *et al.*, 1987; Alon *et al.*, 1991; Kushilevitz & Mansour, 1993). For a recent media access theory survey, we refer to Kowalski (2011).

## 5. Clock Synchronisation

In the previous section we discussed the issue of avoiding collisions between the sensor nodes when using synchronous time slots. As each sensor node is equipped with an internal clock, it should be easy to divide time into slots, and make each node send (or listen, or sleep, respectively) in the appropriate slots according to the media access control (MAC) layer protocol used.†

However, as it turns out, this comes at a price. Synchronising the clocks in a network is not trivial, in particular in sensor networks where nodes often try to save

† Indeed, such a time division multiple access (TDMA) protocol may not be needed. In the example of the Aloha protocol it was shown that one can run slotted protocols also in systems where nodes do not have a common notion of time, at the price of a factor 2. This technique can be generalised to other TDMA MAC protocols. In theory, a factor of 2 does not sound prohibitive, in practice however, it matters!

energy by sleeping.‡ As nodes' internal clocks are not perfect, they will run at speeds that are time-dependent. For instance, variations in temperature or supply voltage will affect this *clock drift.* For standard clocks, the drift is in the order of parts per million, i.e., within a second, it will accumulate to a couple of microseconds. Wireless TDMA protocols account for this by introducing *guard times.* Whenever a node knows that it is about to receive a message from a neighbour, it powers up its radio a little bit earlier to make sure that it does not miss the message even though clocks are not perfectly synchronised. Also, if nodes are badly synchronised, message of different slots might collide.

## *Clock Synchronization in Theory...*

The analysis of guard times is a *clock synchronisation* problem. This kind of questions have been studied by the distributed computing community extensively. For instance, it is well-known that in a network of diameter $D$, the worst-case skew between two nodes is $\Omega(D)$ (Biaz & Welch, 2001). This bound holds even in absence of clock drift, as it is not possible to measure the time it takes to *transmit* the current clock value to another node precisely. The respective error might accumulate with distance, summing up to $\Omega(D)$ between two nodes in distance $D$.

However, as wireless interference is to some degree a geometric phenomenon, guard times need to account for the clock skew to nearby nodes only. Thus, it is sufficient if close-by nodes are well-synchronised. Or, rephrasing this observation, the system can run as if it was synchronous if we can keep the clock skew between neighbours, the *local skew*, small. Fan & Lynch (2004) formulated this so-called *gradient clock synchronisation* problem and proved the surprising result that the local skew must be at least $\Omega(\log D/\log\log D)$ in the worst case, no matter what algorithm is used. Thus, even though nodes may be perfectly aware of their neighbours' clocks being off, they may not be able to compensate for it consistently! Recently, it has been established that smallest possible local skew is $\Theta(\log_{1/\rho} D)$, where $\rho$ bounds the relative clock drift (Lenzen *et al.*, 2010). Recalling that typically $\rho < 10^{-5}$, this implies that in any practical sensor network, the local skew can be bounded by a constant.

## *...and in Practice*

Sadly, again there is a pitfall. The optimal algorithm assumes that neighbours exchange clock values every $\mathcal{O}(\rho\mathcal{J})$ time units, where $\mathcal{J}$ is the uncertainty in message transmission time. By means of MAC layer time stamping, $\mathcal{J}$ can be kept as small as $10^{-6}$ s. However, powering up the radio and transmitting a message can take up to about 0.1 s, i.e., nodes would have to exchange messages all the time!

On the upside, also lower bounds do not always have the final word. In practice, one can achieve much better synchronisation than the bound in Lenzen *et al.* (2010) states! The issue here is that the lower bound holds *in the worst case*, which is unlikely to occur in practice. Instead of clock drift and message delays being arranged in the most adverse manner, typically the speed of clocks changes

‡ Dozer (Burri *et al.*, 2007), for instance, is a sensor network protocol stack that enables long-term functionality of an unattended sensor network by powering down nodes' radios whenever possible.

at a comparatively slow pace and the message transmission times follow a benign probability distribution. Assuming that clock rates change only slightly within $kB$ time (for $k \in \mathbb{N}$ and $B$ being the average time between a node sending/receiving a message), this can be exploited in order to achieve a maximal clock skew of $\mathcal{O}(\mathcal{J}\sqrt{D \log n/(k \log\log n)})$ w.h.p. (Lenzen *et al.*, 2009).† This bound is asymptotically tight for the more optimistic model, and the claimed upper bound could be verified in a real-world network.

## 6. Conclusion

The study of the complexity of distributed algorithms leads to fascinating and deep research questions. In the context of wireless sensor networks, these research question remain theoretically interesting, but they obtain an additional practical dimension. In this work, we have focused on characterising the power of distributed algorithms vis-a-vis some fundamental network coordination tasks that are relevant in wireless sensor networks. We have shown that this power very much depends on the underlying network model.

This work was partially supported by the Swiss National Science Foundation (SNSF). We would like to thank Thomas Moscibroda for his valuable input in an early stage of this manuscript. We are grateful to Johannes Schneider and Jukka Suomela for helpful comments and suggestions on how to improve the article in a late stage.

## References

Abramson, N. 1970 The Aloha System: Another Alternative for Computer Communications. In *Proc. November 17-19, 1970, Fall Joint Computer Conference (AFIPS)*, pp. 281–285.

Afek, Y., Alon, N., Barad, O., Hornstein, E., Barkai, N. & Bar-Joseph, Z. 2011 A Biological Solution to a Fundamental Distributed Computing Problem. *Science*, **331**(6014), 183–185.

Alon, N., Babai, L. & Itai, A. 1986 A Fast and Simple Randomized Parallel Algorithm for the Maximal Independent Set Problem. *Journal of Algorithms*, **7**(4), 567–583.

Alon, N., Bar-Noy, A., Linial, N. & Peleg, D. 1991 A Lower Bound for Radio Broadcast. *Journal of Computer and System Sciences*, **43**, 290–298.

Bar-Yehuda, R., Goldreich, O. & Itai, A. 1987 On the Time-Complexity of Broadcast in Radio Networks: An Exponential Gap Between Determinism and Randomization. In *Proc. 6th Symposium on Principles of Distributed Computing (PODC)*, pp. 98–108.

Barenboim, L. & Elkin, M. 2010 Deterministic Distributed Vertex Coloring in Polylogarithmic Time. In *Proc. 29th Symposium on Principles of Distributed Computing (PODC)*, pp. 410–419.

† It is assumed that the considered time interval is no larger than $n^{\mathcal{O}(1)}$. Since the worst case has always a positive probability to occur, it is impossible to beat the worst-case lower bound of $\Omega(D)$ for all times $t \in [0, \infty)$ with positive probability.

Barriére, L., Fraigniaud, P., Narayanan, L. & Opatrny, J. 2003 Robust Position-Based Routing in Wireless ad hoc Networks with Irregular Transmission Ranges. *Wireless Communications and Mobile Computing*, **3**, 141–153.

Bellare, M., Goldreich, O. & Sudan, M. 1998 Free Bits, PCPs, and Nonapproximability—Towards Tight Results. *SIAM Journal of Computing*, **27**, 804–915.

Biaz, S. & Welch, J. L. 2001 Closed Form Bounds for Clock Synchronization Under Simple Uncertainty Assumptions. *Information Processing Letters*, **80**(3), 151–157.

Breu, H. & Kirkpatrick, D. G. 1998 Unit Disk Graph Recognition is NP-hard. *Computational Geometry Theory and Applications*, **9**, 3–24.

Burri, N., von Rickenbach, P. & Wattenhofer, R. 2007 Dozer: Ultra-low Power Data Gathering in Sensor Networks. In *Proc. 6th Conference on Information Processing in Sensor Networks (IPSN)*, pp. 450–459.

Chlamtac, I. & Kutten, S. 1985 On Broadcasting in Radio Networks–Problem Analysis and Protocol Design. *IEEE Transactions on Communications*, **33**(12), 1240–1246.

Chrobak, M., Gasieniec, L. & Kowalski, D. 2004 The Wake-up Problem in Multi-hop Radio Networks. In *Proc. 15th Symposium on Discrete Algorithms (SODA)*, pp. 992–1000.

Cole, R. & Vishkin, U. 1986 Deterministic Coin Tossing with Applications to Optimal Parallel List Ranking. *Information and Control*, **70**(1), 32–53.

Cornejo, A. & Kuhn, F. 2010 Deploying Wireless Networks with Beeps. In *Proc. 24th Conference on Distributed Computing (DISC)*, pp. 148–162.

Czumaj, A. & Rytter, W. 2006 Broadcasting Algorithms in Radio Networks with Unknown Topology. *Journal of Algorithms*, **60**, 115–143.

Fan, R. & Lynch, N. 2004 Gradient Clock Synchronization. In *Proc. 23rd Symposium on Principles of Distributed Computing (PODC)*, pp. 320–327.

Fanghänel, A., Kesselheim, T., Räcke, H. & Vöcking, B. 2009 Oblivious Interference Scheduling. In *Proc. 28th Symposium on Principles of Distributed Computing (PODC)*.

Fanghänel, A. & Vöcking, B. 2011 Scheduling and Power Assignments in the Physical Model. In *Theoretical Aspects of Distributed Computing in Sensor Networks*, pp. 31–57. Springer Berlin Heidelberg.

Ganesan, D., Estrin, D., Woo, A., Culler, D., Krishnamachari, B. & Wicker, S. 2002 Complex Behavior at Scale: An Experimental Study of Low-power Wireless Sensor Networks. Tech. rep., University of California. CS TR 02-0013.

Gasieniec, L., Pelc, A. & Peleg, D. 2001 The Wakeup Problem in Synchronous Broadcast Systems. *SIAM Journal on Discrete Mathematics*, **14**, 207–222.

Goussevskaia, O., Halldórsson, M., Wattenhofer, R. & Welzl, E. 2009 Capacity of Arbitrary Wireless Networks. In *Proc. 28th Conference on Computer Communications (INFOCOM).*

Goussevskaia, O., Pignolet, Y.-A. & Wattenhofer, R. 2010 Efficiency of Wireless Networks: Approximation Algorithms for the Physical Interference Model. *Foundations and Trends in Networking*, **4**(3), 313–420.

Gupta, P. & Kumar, P. R. 2000 The Capacity of Wireless Networks. *IEEE Transactions on Information Theory*, **46**(2), 388–404.

Halldórsson, M. & Mitra, P. 2011 Wireless Capacity with Oblivious Power in General Metrics. In *Proc. 12th Symposium on Discrete Algorithms (SODA).*

Israeli, A. & Itai, A. 1986 A Fast and Simple Randomized Parallel Algorithm for Maximal Matching. *Information Processing Letters*, **22**(2), 77–80.

Jurdzinski, T. & Stachowiak, G. 2005 Probabilistic Algorithms for the Wake-Up Problem in Single-Hop Radio Networks. *Theory of Computer Systems*, **38**, 347–367.

Kantor, E., Lotker, Z., Parter, M. & Peleg, D. 2011 The Topology of Wireless Communication. In *Proc. 43rd Symposium on Theory of Computing (STOC).*

Kesselheim, T. 2011 A Constant-Factor Approximation for Wireless Capacity Maximization with Power Control in the SINR Model. In *Proc. 12th Symposium on Discrete Algorithms (SODA).*

Kesselheim, T. & Vöcking, B. 2010 Distributed Contention Resolution in Wireless Networks. In *Proc. 24th Symposium on Distributed Computing (DISC)*, pp. 163–178.

Kowalski, D. R. 2011 Coordination Problems in Ad Hoc Radio Networks. In *Theoretical Aspects of Distributed Computing in Sensor Networks*, pp. 319–350. Springer Berlin Heidelberg.

Kowalski, D. R. & Rokicki, M. A. 2010 Connectivity Problem in Wireless Networks. In *Proc. 24th Symposium on Distributed Computing (DISC)*, pp. 344–358.

Kuhn, F., Moscibroda, T. & Wattenhofer, R. 2004 Initializing Newly Deployed ad hoc and Sensor Networks. In *Proc. 10th Conference on Mobile Computing and Networking (MobiCom)*, pp. 260–274.

Kuhn, F., Moscibroda, T. & Wattenhofer, R. 2010 Local Computation: Lower and Upper Bounds. *Computing Research Repository*, **abs/1011.5470**.

Kuhn, F., Nieberg, T., Moscibroda, T. & Wattenhofer, R. 2005 Local approximation schemes for ad hoc and sensor networks. In *Proc. 3rd Joint Workshop on Foundations of Mobile Computing (DialM-POMC)*, pp. 97–103.

Kuhn, F., Wattenhofer, R. & Zollinger, A. 2008 Ad hoc Networks Beyond Unit Disk Graphs. *Wireless Networks*, **14**, 715–729.

Kushilevitz, E. & Mansour, Y. 1993 An $\Omega(D \log(N/D))$ Lower Bound for Broadcast in Radio Networks. In *Proc. 12th Symposium on Principles of Distributed Computing (PODC)*, pp. 65–74.

Lenzen, C., Locher, T. & Wattenhofer, R. 2010 Tight Bounds for Clock Synchronization. *Journal of the ACM*, **57**(2).

Lenzen, C., Sommer, P. & Wattenhofer, R. 2009 Optimal Clock Synchronization in Networks. In *Proc. 7th Conference on Embedded Networked Sensor Systems (SenSys)*, pp. 225–238.

Lenzen, C. & Wattenhofer, R. 2008 Leveraging Linial's Locality Limit. In *Proc. 22nd Symposium on Distributed Computing (DISC)*, pp. 394–407.

Lotker, Z. & Peleg, D. 2010 Structure and Algorithms in the SINR Wireless Model. *SIGACT News*, **41**(2), 74–84.

Luby, M. 1986 A Simple Parallel Algorithm for the Maximal Independent Set Problem. *SIAM Journal on Computing*, **15**(4), 1036–1055.

Marathe, M., Breu, H., Hunt III, H. B., Ravi, S. S. & Rosenkrantz, D. J. 1995 Simple Heuristics for Unit Disk Graphs. *Journal of Networks*, **25**, 59–68.

Métivier, Y., Robson, J. M., Saheb Djahromi, N. & Zemmari, A. 2009 An optimal bit complexity randomised distributed MIS algorithm. In *Proc. 16th Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pp. 323–337.

Moscibroda, T. & Wattenhofer, R. 2006 The Complexity of Connectivity in Wireless Networks. In *Proc. 25th Conference on Computer Communications (INFOCOM)*.

Naor, M. 1991 A Lower Bound on Probabilistic Algorithms for Distributive Ring Coloring. *SIAM Journal on Discrete Mathematics*, **4**(3), 409–412.

Panconesi, A. & Srinivasan, A. 1996 On the Complexity of Distributed Network Decomposition. *Journal of Algorithms*, **20**(2), 356–374.

Peleg, D. 2000 *Distributed Computing: A Locality-Sensitive Approach.* Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.

Schmid, S. & Wattenhofer, R. 2006 Algorithmic Models for Sensor Networks. In *Proc. 14th Workshop on Parallel and Distributed Real-Time Systems (WP-DRTS).*

Schneider, J. & Wattenhofer, R. 2008 A log-star Distributed Maximal Independent Set Algorithm for Growth-Bounded Graphs. In *Proc. 27th Symposium on Principles of Distributed Computing (PODC)*, pp. 35–44.

Suomela, J. 2011 Survey of Local Algorithms. Manuscript, in submission to ACM Computing Surveys.

Willard, D. E. 1986 Log-logarithmic Selection Resolution Protocols in a Multiple Access Channel. *SIAM Journal of Computing*, **15**, 468–477.