Distributed Alarming in the On-Duty and Off-Duty Models

Marcin Bienkowski[1], Leszek Gąsieniec[2], Marek Klonowski[3],

Miroslaw Korzeniowski[3], Bernard Mans[4], Stefan Schmid[5], Roger Wattenhofer[6]

[1] Institute of Computer Science, University of Wroclaw, Poland
[2] Department of Computer Science, University of Liverpool, UK
[3] Institute of Mathematics and Computer Science, Wroclaw University of Technology, Poland
[4] Department of Computing, Macquarie University, Sydney, Australia
[5] Telekom Innovation Laboratories & TU Berlin, Berlin, Germany
[6] Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland

***Abstract*—Decentralized monitoring and alarming systems can be an attractive alternative to centralized architectures. Distributed sensor nodes (e.g., in the smart grid's distribution network) are closer to an observed event than a global and remote observer or controller. This improves the visibility and response time of the system. Moreover, in a distributed system, local problems may also be handled locally and without overloading the communication network.**

This article studies alarming from a distributed computing perspective and for two fundamentally different scenarios: *on-duty* and *off-duty*. We model the alarming system as a sensor network consisting of a set of distributed nodes performing local measurements to sense events. In order to avoid false alarms, the sensor nodes cooperate and only escalate an event (i.e., raise an alarm) if the number of sensor nodes sensing an event exceeds a certain threshold. In the on-duty scenario, nodes not affected by the event can actively help in the communication process, while in the off-duty scenario non-event nodes are inactive.

This article presents and analyzes algorithms that minimize the reaction time of the monitoring system while avoiding unnecessary message transmissions. We investigate time and message complexity tradeoffs in different settings, and also shed light on the optimality of our algorithms by deriving cost lower bounds for distributed alarming systems.

I. Introduction

Distributed monitoring and alarming is an important paradigm to detect harmful events early and robustly. In contrast to centralized monitoring systems, distributed solutions do not rely on the presence of (or even the presence of a functioning path to) a global and possibly remote controller. There are several interesting applications.

1) The Californian earthquake system *Quake Catcher* [8], started in 2008 and still active in 2013, is a distributed monitoring network which benefits from decentralization. The network consists of normal Mac laptops that work as seismic monitors. The laptops aggregate a wealth of information on major quakes in order to alarm in a few seconds' notice of a devastating quake.
2) While traditionally, electricity grids were relatively static, there is a trend [23] towards the integration of renewable energies. The "smart" operation of such networks poses several new monitoring and control challenges to prevent instabilities or blackouts due to dynamical patterns in energy demand and supply. Especially in the low-voltage distribution networks consisting of many sensors and smart meters, communication overhead should be kept low. If network devices manage to detect and locally resolve such problems (e.g., by powerline communications), faults can be contained locally and with good response times.
3) Distributed monitoring systems can also be used to detect fast propagating Internet worms. [14]

This article attends to the distributed alarming problem from an algorithmic point of view, and explores the possibilities and limitations of distributed solutions. We model the problem as a sensor network where (not necessarily wireless) sensor nodes are distributed in space and perform local measurements (e.g., of the temperature or humidity). When these devices sense an event, an *alarm* must be raised as soon as possible (e.g., to inform helpers in the local community). However, as the measurement of a single sensor may be unreliable and as a situation should only be escalated if the event is of a certain magnitude, the nodes must *cooperate* in order to avoid false alarms: nodes sensing an event should make sure that there are other nodes in their vicinity that have sensed the same event. Moreover, in order to save energy and avoid interference or congestion, the number of messages transmitted by a distributed alarming protocol should be minimized.

A. The Model

The *distributed alarming problem* studied in this article can be formalized as follows. We are given a sensor network in the form of an arbitrary undirected, unweighted graph $G = (V, E)$ where the n nodes form a set V with unique identifiers and are connected via $m = |E|$ communication edges E. We assume that at time t_0, arbitrary subset

of nodes $S \subseteq V$ senses an *event*. The nodes in S are called *event nodes*, and the nodes in $V \setminus S$ are called *non-event nodes*. For ease of presentation, in this paper, we will focus on a single connected component S. In this case, we will often denote the subgraph induced by S by $G(S)$ and refer to this graph as the *event component*. In Section V, we describe how our algorithms can be modified to work for multiple event components (in most cases the corresponding components can simply explored in parallel).

We assume that after an event hits the subset of nodes S at time t_0, at least one node from S is required to determine the *size* of S, which is denoted by n_S throughout the article. (In fact, all our algorithms can be easily modified, without changing their asymptotic performance, so that *all* nodes from S learn the value of n_S.) For convenience, we also define m_S as the number of edges in the component $G(S)$ induced by S. This article studies distributed algorithms that jointly minimize the message and time complexities.

Our model divides time into two main *phases*. The time before t_0 is called the *Preprocessing Phase*: During this phase, mechanisms can be set in place to be able to handle events occurring at t_0 more efficiently; the set of event nodes is not known at this time. The time at and after t_0 is called the *Runtime Phase*: The set S is revealed and nodes need to coordinate quickly to explore the component.

The preprocessing can be performed in a centralized fashion, with a global knowledge of the network, and its resulting structure can be reused for all future events. Hence, in this article, the complexity of the preprocessing phase plays a minor role; we are rather interested in a fast and efficient reaction during runtime. The complexities of all preprocessing phases proposed in this article are moderate, though.

Definition I.1 (The Distributed Alarming Problem)**.** *After preprocessing the network, when an event hits the event component $G(S)$ at time t_0, how can the nodes coordinate the exploration of $G(S)$, so that the time and message complexities are minimized? The time complexity is measured until at least one event node learns the size of S (and is aware of this fact), and the message complexity is measured until no more new messages are sent.*

We distinguish between two main scenarios:

- In the *on-duty* scenario, we allow non-event nodes to participate in the runtime phase of the algorithm, for example to serve as relay points for message transmissions of the event nodes.
- In the *off-duty* scenario, only event nodes may participate in the runtime phase.

The first scenario is suited for larger sensor nodes that are attached to an energy supply, whereas the second one addresses the situation of a (wireless) network where (battery-powered) nodes are in a parsimonious sleeping mode until woken up by the event.

This article assumes a *synchronous* environment, where algorithms operate in *communication rounds*: we assume that events are sensed by all nodes simultaneously at time t_0 and there is an upper bound (known by all nodes) on the time needed to transmit a message between two nodes. In particular, the classic $\mathcal{LOCAL}$ model [25] is considered: in each round, a node can send a message to each of its neighbors, receive messages from its neighbors, and depending on the received information, perform local computations. No specific constraints on the allowed message size are assumed.

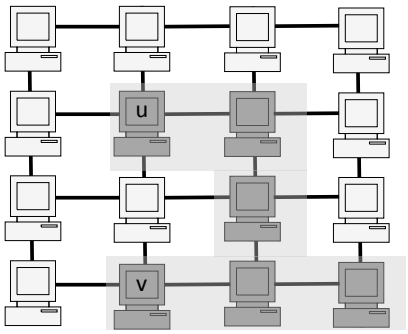

Fig. 1. After hitting the event component (*shaded*), the event nodes S (*dark*) need to determine the component size (*here*: 6). In the on-duty model, node u may reach node v in two hops (via a non-event node); in the off-duty model, these two nodes are at distance four.

While this model is simple and does not capture interference and congestion aspects, it allows us to focus on the fundamental problem of distributed local event detection. Figure 1 illustrates our model.

B. Preliminaries and Terminology

This article uses the following graph-theoretical concepts. For any two nodes u, v from a connected component $G(S)$, we define $\mathrm{dist}_S(u,v)$ as the length of a shortest path between u and v that uses only nodes from S as intermediate nodes. In these terms, $\mathrm{dist}(u,v) := \mathrm{dist}_V(u,v)$ is the standard distance between u and v in the graph. (As our graph is unweighted, the distance is simply the number of hops.) Furthermore, the t-neighborhood of a node v is the set $N_t(v) = \{u \,|\, \mathrm{dist}(u,v) \leq t\}$. We also define two different types of diameters of node sets: the *weak* and the *strong* one.

Definition I.2 (Weak and Strong Diameters)**.** *For a set $S \subseteq V$ of nodes of a graph $G = (V,E)$, the weak diameter of $G(S)$ is $diam(S) := \max_{u,v\in S}\{dist(u,v)\}$ and the strong diameter is $Diam(S) := \max_{u,v\in S}\{dist_S(u,v)\}$.*

In other words, the strong diameter is a diameter of a subgraph induced by a given subset of nodes. Clearly, for any set S it holds that $\mathrm{diam}(S) \leq \mathrm{Diam}(S)$. Henceforth, when the set S is clear from the context, we will just write d for $\mathrm{diam}(S)$ and D for $\mathrm{Diam}(S)$.

In the preprocessing phase of our algorithms, we will often use *(k,t)-neighborhood covers* [25]. In a nutshell, a neighborhood cover covers all nodes by a collection of possibly overlapping sets called *clusters*. In such a cover, the diameter of each cluster is bounded by $O(kt)$ and for

each node there exists a cluster containing the entire t-neighborhood of this node. It turns out that such covers can be computed quite efficiently.

Definition I.3 (Neighborhood Cover [3]). *A (k,t)-neighborhood cover of a graph $G = (V, E)$ is a collection of clusters $C_1, ..., C_r \subseteq V$ with the following properties:*

1) *for each cluster C_i, $diam(C_i) \in O(kt)$, and*
2) *for any node v, there exists a cluster C_i, such that $N_t(v) \subseteq C_i$.*

The node with the largest ID in a given cluster C_i is called the cluster head *h of C_i. A (k,t)-neighborhood cover is* sparse *(and denoted (k,t)-NC) if each node is in at most $O(kn^{1/k})$ clusters.*

The following lemma is due to [3].

Lemma I.4 ([3]). *Given a graph $G = (V, E)$ and integers k, $t \geq 1$, there exists a deterministic distributed algorithm that constructs a (k, t)-NC.*

In this paper, we show that the complexity of local event detection sometimes depends on the graph arboricity [10], [24], which is defined as follows.

Definition I.5 (Arboricity $\alpha(G)$). *The* arboricity $\alpha(G)$ *of an arbitrary graph G is defined as the minimum number of forests that are required to cover all edges in G.*

For any graph G, it holds that $1 \leq \alpha(G) \leq n$.

C. Algorithmic Challenges

The distributed alarming problem considered in this paper poses two main algorithmic challenges, and most of our algorithms consist of two building blocks accordingly.

The first building block addresses an issue called *neighborhood problem*: after a node has sensed an event, it does not know which of its neighbors (if any) are also in S. Distributed algorithms where event nodes simply ask all their neighbors are costly: if G is a star network and the only node in S is the star center, the message complexity is $\Theta(n)$ while the size of the event component is one. The standard trick to let nodes only ask the neighbors of higher degree does not work either: while it would be a solution for the star graph, it fails for dense graphs such as the clique graph. In fact, at first glance it may seem that $\Omega(n)$ is a lower bound for the message complexity of *any* algorithm for the clique as an event node has no information about its neighbors. We will show, however, that this intuition is incorrect in the on-duty scenario.

The second building block deals with the coordination of the nodes during the exploration of the component. In a distributed algorithm where all nodes start exploring the component independently, much redundant information is collected, resulting in a high number of messages. Clearly, the time required to compute the event component size is at least linear in the diameter of $G(S)$, and the number of messages needed by any distributed algorithm is at least linear in n_S. We are hence striving for distributed algorithms which are not very far from these lower bounds.

D. Contribution and Novelty

Despite the fundamental nature of the distributed alarming problem, to the best of our knowledge, this is the first paper to study local algorithms whose time and message complexities depend on the actual event component size, i.e., they are *output-sensitive*. Accordingly, we cover the problem broadly, and investigate different settings as well as different algorithms, both deterministic and randomized ones, and shed light on inherent lower bounds: what cannot be achieved efficiently in a distributed setting.

We introduce the on-duty and off-duty models and show that they are very different, both with respect to the applicable algorithmic techniques as well as the achievable performance: while in the on-duty scenario, structures (such as shortest paths to coordinators) computed during the preprocessing phase are also useful when the event occurs, the off-duty scenario can hardly rely on the existence of such structures and needs to coordinate almost from scratch.

Our model is also novel in the explicit distinction between preprocessing and runtime phase. Classic algorithms designed for ad-hoc and sensor networks typically need to start exploring the topology from scratch. We argue that in the context of event detection, we can often distinguish between two very different time scales: (1) setting up a communication infrastructure (before t_0), and (2) handling the actual event. The question of how to set up such infrastructures has hardly been studied in the literature so far. Moreover, it turns out that given the preprocessing phase (1), the event detection (2) can often be done much more efficiently.

Our technical contribution is twofold. First, we show which existing algorithmic techniques can be used and adapted for the distributed alarming problem. Second, we also derive new algorithmic techniques; for instance, we introduce a hierarchical sparse neighborhood algorithm which may be of independent interest.

E. Overview of Results

This article presents deterministic and randomized algorithms for different settings, both in the on-duty and the off-duty models, studies their optimality, and derives impossibility results.

1) *Clique Bounds: On-Duty vs Off-Duty.* We first show that there is a gap between on-duty and off-duty models, already in the simple complete network (the clique). While in the on-duty model, the distributed alarming problem can be solved in two rounds using $O(n_S)$ messages, where n_S is the event component size (Theorem II.1), for any deterministic algorithm ALG using TIME rounds and MSG messages, it holds that $\text{TIME} \cdot \text{MSG} = \Omega(n \cdot \log \log n)$, where n is the size of the whole network. (Theorem II.4)
2) *Clique Off-Duty Algorithms.* We present a class of deterministic algorithms GROUP that solves the problem in the off-duty clique model. GROUP is parametrized with a number k, and comes in two

variants: PAR and SEQ. The PAR variant of the algorithm GROUP uses $O(\log_k n)$ rounds and transmits $O(\min\{k, n_S\} \cdot n \cdot \log_k n)$ messages. The SEQ variant uses $O(k \cdot \log_k n)$ rounds and $O(n \cdot \log_k n)$ messages. (Theorem II.5) We also present a randomized Las Vegas algorithm RAND. The algorithm terminates in $\log(n/n_S) + O(1)$ rounds and uses $O(n)$ messages on expectation. (Theorem II.6)

3) *General On-Duty Algorithms.* We present the randomized on-duty algorithm DECOMP that solves the alarming problem on general graph instances. It is based on pre-computed sparse neighborhood covers where information about the event is passed on via independent sets, in order to reduce the message complexity. The runtime of DECOMP is $O(d \cdot \log^2 n)$ rounds, using $O(n_S \cdot \log^3 n \cdot \log d)$ messages; it computes the correct solution with high probability. Here, d is the weak diameter of the event node set S. (Theorem IV.1) DECOMP is optimal up to polylogarithmic factors.
4) *General Off-Duty Algorithms.* We describe the deterministic algorithm MINID that solves the alarming problem on general graphs. For a graph G with arboricity α, the MINID finishes in $O(n_S \log n_S)$ rounds using $O(m_S \log n_S + \alpha \cdot n_S)$ messages, where m_S is the number of edges in the component $G(S)$ induced by S. (Theorem IV.3)
5) *Search-based Algorithms.* Finally, we present the search-based algorithms PARDFS, PARBFS and ONPARBFS. These algorithms solve the problem, respectively, in time $O(n_S)$, $O(D)$ (where D is the strong diameter of the event component), and $O(d)$ (the weak diameter), using respectively $O(n_S \cdot (\alpha + n_S))$, $O(n_S \cdot (\alpha + m_S))$, and $O(n_S \cdot m))$ messages, where m is the total number of edges. The first two algorithms can be used in both scenarios, the third one in the on-duty scenario only. (Theorem IV.6) We generalize the search algorithms to perform a graded start of the DFS procedures. Our algorithm k-NWDFS, for any integer $k \in \{1, \ldots, n\}$, solves the problem in $O(n^2/k)$ rounds. In the off-duty scenario, it uses $O((\alpha + \min\{k, n_S\}) \cdot n_S)$ messages while in the on-duty scenario it uses $O((\min\{\alpha, \log^2 n\} + \min\{k, n_S\}) \cdot n_S)$ messages. (Theorem IV.8)

F. Related Work

Local algorithms have been explored for many years already, especially in the context of ad-hoc and sensor networks. Researchers in this field are particularly interested in the question of *what can and what cannot be computed locally* [20], i.e., given the knowledge of only a subset of the graph. For a good overview of the field in general and the $\mathcal{LOCAL}$ in particular, we refer the reader to the introductory book by Peleg [25] and the survey by Suomela[28].

Pre-Processing and Output Sensitive Local Algorithms. However, the distributed alarming model considered in this paper is different from the classic ad-hoc network problems, in that it distinguishes between two very different time scales: (1) the network and infrastructure construction (in this paper referred to as the pre-processing phase), and (2) the actual distributed event exploration (referred to as the runtime phase). As we will see, precomputations can often help to significantly speed up the execution of a local algorithm during runtime, determining the event component at lower time and message complexity. In this sense, our article also assumes an interesting new position between local and global distributed computations, as our algorithms aim at being *as local as possible* and *as global as necessary*.

In our model, we require that the time and message complexities depend on the actual problem instance size: the algorithms should be *output-sensitive*. There already exist some local solutions for other problems whose runtime depends on the concrete problem input, for example [5], [9]: rather than considering the worst-case over all possible inputs, if in a special instance of a problem the input has certain properties, a solution can be computed quickly. In this respect, our work is also related to local algorithms which rely on proof labeling schemes [12], [16], [17] and allow to locally and efficiently verify global properties.

"Output-sensitive" algorithms have also been studied outside the field of distributed computing, e.g., for sorting algorithms [15] (where the complexity of insertion sort or bubble sort depends on the number of *inversions*). Our article is a new incarnation of this philosophy as performance mostly depends on n_S, the event component size instead of n, the size of the whole network.

Finally, we note that a new $\mathcal{LOCAL}$ model which explicitly distinguishes between pre-processing and runtime has recently also been introduced in the context of Software-Defined Networks (SDNs) with distributed control planes. [27]

Algorithmic Techniques. In general, distributed alarming algorithms can be built on the basis of several classic distributed coordination primitives such as clustering [1] or spanning trees [11] for aggregation [19].

In this paper, we show that maximal independent set algorithms [22] can be used for the efficient graph exploration, and also build upon sparse neighborhood covers [4], [13] to solve the neighborhood problem in the on-duty model. For our off-duty algorithms, the arboricity of the network plays a central role, and there exist several results on the usefulness of Nash-Williams decompositions, e.g., for computing distributed minimal dominating sets [21], matchings [7], or coloring [18].

However, we not only adapt existing concepts to the new problem, but also introduce new algorithmic techniques. In particular, we introduce a hierarchical sparse neighborhood cover to achieve the desired output-sensitive time and message complexities.

Bibliographic Note. This article is based on two conference papers which appeared as "Distributed Disaster Disclosure" (SWAT 2008) and as "Event Extent Estimation"

(SIROCCO 2010). This journal version extends the conference papers, gives full proofs as well as pseudocodes and examples, and also introduces a new DECOMP algorithm. Finally, we also correct some mistakes.

G. Organization

The article is organized as follows. We first present algorithms for the simple case of complete graphs (Section II). We discuss on-duty and off-duty approaches as well as lower bounds, and find that the distributed alarming problem already exhibits some interesting time and message complexity tradeoffs on the clique. Subsequently, we attend to the fundamental problem of neighborhood discovery (Section III). Section IV then tackles general graphs. We show how to extend our algorithms to scenarios with multiple components in Section V, and finally conclude our work in Section VI.

II. Algorithms for Cliques

To acquaint the reader with the problem definition and — more importantly — to show the fundamental differences between the on-duty and the off-duty scenarios, we consider one of the simplest classes of graphs, namely *cliques*, i.e., complete graphs of n nodes.

A. On-Duty: The Optimal Solution

First, observe that solving the problem in the on-duty scenario on cliques is quite trivial. We need just one coordinating node v, chosen arbitrarily in the preprocessing phase. Then, when the event occurs, each event node sends a message to the coordinator v and v may reply with the number of event nodes. This immediately yields the following result.

Theorem II.1. *For the on-duty scenario the distributed alarming problem on a clique can be solved in two rounds using $O(n_S)$ messages.*

Clearly, the message and time complexities are asymptotically optimal. In contrast to this result, in the following section, we will show that for the off-duty scenario, the necessary number of messages in a clique is $\Omega(n)$. Furthermore, we show a tradeoff between the time and message complexity proving that their product is always $\Omega(n \log\log n)$ in the off-duty model.

B. Off-Duty: Lower Bounds

We observe that if there is a set S of n_S event nodes, the necessary condition for termination is that at least one event node sends a message to any other event node. Therefore, we show that for any deterministic algorithm, we may choose the set of event nodes, so that before there is any contact between a pair of event nodes, the number of messages sent (to non-event nodes) is $\Omega(n)$.

To this end, we introduce a concept of *primary schedules*. We fix any deterministic algorithm ALG, and assume for a while that only node i belongs to S. Then, node i transmits messages in some particular order, which we call a *primary schedule for i*. Note that for any starting set of event nodes, ALG uses the primary schedule for i as long as i does not receive a message from another node. For succinctness, we say that ALG *p-sends* a message in round ℓ, meaning that the primary schedule of ALG assumes sending a message in round ℓ. Note that the choice of ALG determines the choices of primary schedules of all nodes.

We say that two nodes *p-contact* each other if one of them p-sends a message to the other. Using a counting argument, we can find a pair of nodes that p-contact after transmitting many messages.

Lemma II.2. *For any deterministic algorithm for the clique and for any subset of k nodes A, there exists a pair of nodes $v, v' \in A$ that contact only after one of them p-sends at least $k/2 - 1$ messages.*

Proof: First, we observe that the total number of messages in all primary schedules is at least $\binom{k}{2}$. Otherwise, there would exist a pair of nodes that never p-contact. In effect, if the algorithm is run on an instance where only these two nodes belong to event component, it cannot solve the problem, as neither of these nodes can distinguish between instances where the second node is in S or not.

For simplicity of the description, we assume that messages are p-sent sequentially. The j-th message of node i receives label j. An edge between node i and i' receives the label which is the minimum of the labels of messages sent from i to i' and from i' to i. To show the lemma, it is sufficient to show the existence of an edge whose label is at least $k/2$. Assume the contrary, i.e., all edges have labels $k/2 - 1$ or smaller. Then, the label of any message would be at most $k/2 - 1$, which would imply that the total number of p-sent messages is $k \cdot (\frac{k}{2} - 1) < \binom{k}{2}$. ∎

Corollary II.3. *The number of messages sent by any deterministic algorithm in a clique is at least $\Omega(n)$.*

Proof: We apply Lemma II.2 with set A containing all n nodes of the clique and we choose the two nodes returned by Lemma II.2 to be in S. Before they contact, they together transmit $\Omega(n)$ messages. ∎

Theorem II.4. *Fix any deterministic algorithm* ALG *that solves the distributed alarming problem in a clique using* TIME *rounds and* MSG *messages. Then* TIME $\cdot$ MSG $= \Omega(n \cdot \log\log n)$.

Proof: We assume that $\log\log n \geq 4$. We consider the first t rounds of the nodes' primary schedules, where $t = \log(\log n / \log\log\log n) = \Omega(\log\log n)$.

First, assume that there exists a subset A of $n/2$ nodes, each p-sending fewer than $n/4$ messages in the first t steps. By Lemma II.2, there exists a pair of nodes $v, v' \in A$ that first p-contact after one of them p-sends at least $|A|/2 - 1 = n/4 - 1$ messages. Thus, if we start ALG on a graph where only v and v' belong to S, it takes at least time t and uses at least $n/4$ messages, implying the lemma.

Hence, in the remaining part of the proof, we assume that there exists a set B_0 containing at least $n/2$ nodes,

each p-sending at least $n/4$ messages within the first t steps. We create a sequence of sets $\{B_i\}_{i=0}^{t}$, where B_i is a maximum subset of B_{i-1} with the property that no two nodes of B_i p-send a message to each other in round i. By induction, no node from B_i p-sends a message to another node from B_i within the first i steps. Let $h = \frac{1}{2} \cdot \log\log n$. We consider two cases.

(a) There exists a round $i \leq t$, in which nodes of B_i p-send in total at least $hn/4$ messages. We run ALG on a graph where only nodes of B_i are event nodes. The event nodes do not contact each other in the first $i-1$ rounds and in round i they transmit $hn/4 = \Omega(n \log\log n)$ messages, implying the theorem.

(b) In each round $i \leq t$, nodes of B_i p-send in total at most $hn/4$ messages. In this case, we show that B_t contains at least 2 nodes. Thus, if we run ALG on a graph where only nodes of B_t are event nodes, they do not contact in the first $t-1$ rounds and, as $B_t \subseteq B_0$, they transmit at least $n/4$ messages in the first t rounds, which would imply the theorem. To prove the bound $|B_t| \geq 2$, it is sufficient to show that for any $i \leq t$, it holds that

$$|B_i| \geq \frac{n}{2 \cdot (2h)^{2^i - 1}} \ .$$

We show this relation inductively. The initial case of $i = 0$ holds trivially. Assume that the bound holds for $i-1$; we show it for i. Consider a graph on nodes from B_{i-1} with an edge connecting a pair of nodes if they p-contact in round $i-1$; the number of edges in such a graph is at most $hn/4$. By Turán's theorem [2], there exists an independent set $B_i \subseteq B_{i-1}$ of size

$$\begin{aligned} |B_i| &\geq \frac{|B_{i-1}|}{1 + \frac{h \cdot n}{2 \cdot |B_{i-1}|}} \geq \frac{|B_{i-1}|^2}{h \cdot n} \\ &\geq \frac{n^2}{4 \cdot (2h)^{2^i - 2} \cdot h \cdot n} = \frac{n}{2 \cdot (2h)^{2^i - 1}} \end{aligned}$$

In our context, independence means that the nodes of B_i do not p-contact each other in round i.

■

C. Off-Duty: A Deterministic Algorithm

Let us now investigate deterministic algorithms for the clique problem in the off-duty scenario. Clearly, a broadcast performed by all event nodes would be time-optimal, but it requires $n_S \cdot (n-1)$ messages. We therefore propose a natural class of algorithms called GROUP where nodes organize themselves recursively into groups.

The algorithm GROUP uses an integer parameter $k \in \{2, \ldots, n\}$. For simplicity of description, we assume that $\log_k n$ is an integer as well. We further assume that node identifiers are written as $(\log_k n)$-digit strings, where each digit is an integer between 0 and $k-1$. This implicitly creates the following hierarchical partitioning of all nodes into clusters. The topmost cluster (on level $\log_k n$) contains all nodes and is divided into k clusters, each consisting of n/k nodes, where cluster i contains all the nodes whose first digit is equal to i. Each of these clusters is also partitioned into k clusters on the basis of the second digit of identifiers. This partitioning proceeds to leaves, which are 0^{th} level clusters, each containing a single node. We call a cluster *active* if it contains at least one event node.

GROUP works in $\log_k n$ epochs; we assume that there is an empty 0^{th} epoch before the algorithm starts. We inductively require that at the end of the i^{th} epoch, there is a *leader* in each i^{th} level active cluster, the leader knows all the event nodes within its cluster and all these nodes know the leader. Note that this property holds trivially at the end of epoch 0. Furthermore, this property implies that at the end of epoch $\log_k n$, all nodes constitute a single cluster and its leader knows the set of all event nodes.

To study what happens in the i^{th} epoch, we concentrate on a single i^{th} level cluster A. (The procedure is performed in all such clusters independently in parallel.) A consists of k $(i-1)^{th}$ level clusters, denoted $A_1, A_2, \ldots, A_k$ that will be merged in this epoch. The leader of A will be chosen as the node with the smallest ID amongst leaders of active clusters A_i. The merging procedure comes in two flavors: parallel (PAR) and sequential (SEQ).

In the PAR variant, an epoch lasts two rounds. In the first round, the leaders of clusters A_j broadcast a *hello* message to all nodes from these clusters. All the event nodes among them answer with a message to a leader with the smallest identifier, and this node becomes a leader of the i^{th} level cluster A.

In the SEQ variant, the epoch lasts for $k+1$ rounds. For $j \leq k$, in the j^{th} round, the leader of cluster A_j broadcasts a hello message to all nodes from A, provided such a message was not sent already. The nodes that hear the message answer in the next round, and the leader that transmitted the broadcast becomes a leader of A.

Theorem II.5. *The* PAR *variant of the algorithm* GROUP *uses* $O(\log_k n)$ *rounds and transmits* $O(\min\{k, n_S\} \cdot n \cdot \log_k n)$ *messages. The* SEQ *variant uses* $O(k \cdot \log_k n)$ *rounds and* $O(n \cdot \log_k n)$ *messages.*

Proof: The time complexities for both variants are straightforward, and thus we concentrate on bounding the number of messages. In a single epoch of the PAR variant, each node receives a *hello* message from at most $\min\{k, n_S\}$ leaders and sends a single reply. This implies the total number of $2 \cdot \min\{k, n_S\} \cdot n \cdot \log_k n$ messages. In a single epoch of the SEQ variant, each node gets at most one *hello* message and answers at most once. Thus, the total number of transmitted messages is at most $2 \cdot n \cdot \log_k n$. ■

We observe that the best time-message product is achieved for $k = 2$, in which case both variants of GROUP solve the problem in time $O(\log n)$ using $O(n \log n)$ messages. Note that GROUP can be regarded as a generalization of two graph search techniques: the extreme cases require either one round or n messages and correspond to the parallel or sequential flooding of the graph by event nodes.

D. Off-Duty: A Las Vegas Algorithm

In this section, we extend our discussion to randomized approaches. The idea behind our algorithm RAND is to approximately "guess" the number of event nodes. For succinctness of the description, we assume that n is a power of 2. RAND proceeds in $\log n+1$ epochs, numbered from 0 to $\log n$, each consisting of two rounds. In the first round of the i^{th} epoch, each node — with probability $p_i = 2^i/n$ — broadcasts a *hello* message to all other nodes. In the second round event nodes reply. After an epoch with a broadcast, the algorithm terminates. The algorithm RAND eventually always solves the problem, as in epoch $\log n$ each node performs a broadcast with probability 1, i.e., RAND is a *Las Vegas* type of an algorithm.

Theorem II.6. *In expectation,* RAND *terminates in* $\log(n/n_S) + O(1)$ *rounds and uses* $O(n)$ *messages.*

Proof: Let $k = \lceil \log(n/n_S) \rceil$, i.e., $2^{k-1} < n/n_S \leq 2^k$. Then, epoch k is the first epoch in which the broadcast probability of each node reaches $1/n_S$, i.e., $p_k \in [1/n_S, 2/n_S)$. It is sufficient to show that the algorithm makes its first broadcast around epoch k and, in expectation, it makes a constant number of broadcasts.

Let $\mathcal{E}_i$ denote an event that RAND does not finish till epoch i (inclusive), i.e., there was no broadcast in epochs $1, 2, \ldots, i$. Let τ be a random variable denoting the number of epochs of RAND. Then, $\mathbf{E}[\tau] = \sum_{i=1}^{\log n} \Pr[\tau \geq i] = \sum_{i=0}^{\log n - 1} \Pr[\mathcal{E}_i] \leq \sum_{i=0}^{k-1} 1 + \sum_{j=0}^{\log n - k - 1} \Pr[\mathcal{E}_{k+j}]$.

To bound the last term, we first observe that the necessary condition for $\mathcal{E}_i$ is that no node transmits in epoch i. Hence, $\Pr[\mathcal{E}_i] \leq (1-p_i)^{n_S}$, and thus for $0 \leq j \leq \log n - k - 1$,

$$\Pr[\mathcal{E}_{k+j}] = \left(1 - \frac{2^{k+j}}{n}\right)^{n_S} \leq \left(\frac{1}{\mathrm{e}}\right)^{\frac{2^{k+j}}{n} \cdot n_S} \leq \mathrm{e}^{-2^j} \ .$$

Therefore, $\mathbf{E}[\tau] \leq k + O(1)$.

Now, we upper-bound the number of transmitted messages. Let X_i be a random variable denoting the number of nodes transmitting in epoch i. Clearly, $\mathbf{E}[X_0] = n_S \cdot p_0$ and for $i \geq 1$ it holds that $\mathbf{E}[X_i|\mathcal{E}_{i-1}] = n_S \cdot p_i$ and $\mathbf{E}[X_i|\neg\mathcal{E}_{i-1}] = 0$. The expected total number of broadcasts is then

$$\begin{aligned}
&\mathbf{E}\Big[\sum_{i=0}^{\log n} X_i\Big] \\
&\quad = \mathbf{E}[X_0] + \sum_{i=1}^{\log n} \mathbf{E}[X_i|\mathcal{E}_{i-1}] \cdot \Pr[\mathcal{E}_{i-1}] \\
&\quad = n_S \cdot p_0 + \sum_{i=1}^{k} n_S \cdot p_i \cdot \Pr[\mathcal{E}_{i-1}] \\
&\qquad + \sum_{i=k+1}^{\log n} n_S \cdot p_i \cdot \Pr[\mathcal{E}_{i-1}] \\
&\quad \leq \sum_{i=0}^{k} n_S \cdot p_i + \sum_{j=0}^{\log n - k - 1} n_S \cdot p_{k+j+1} \cdot \Pr[\mathcal{E}_{k+j}] \\
&\quad \leq 2 \cdot n_S \cdot p_k + \sum_{j=0}^{\log n - k - 1} n_S \cdot p_k \cdot 2^{j+1} \cdot \mathrm{e}^{-2^j} \\
&\quad = O(n_S \cdot p_k) = O(1) \ .
\end{aligned}$$

As the expected number of broadcasts is constant, the expected number of messages is $O(n)$. ∎

III. The Neighborhood Problem

Most of our algorithms for general graphs solve the neighborhood problem in their first few rounds. In these rounds, each node learns which of its immediate neighbors are event nodes.

While for special classes of graphs, e.g., trees, there are straightforward approaches, the situation for arbitrary graphs is less obvious. In this section, we present two approaches for the neighborhood problem. The first one, a *network decomposition approach* [3] requires the cooperation of non-event nodes, and hence works only in the on-duty scenario. The second one, employing the concept of arboricity [10], [24] does not have such a requirement, and thus can be used in both scenarios.

A. Sparse Neighborhood Covers

The first possible solution for the neighborhood problem is based on network decomposition approach, concretely on the (k,t)-*neighborhood cover* [25]. For solving the neighborhood problem, in the preprocessing phase, we compute a $(\log n, 1)$-NC. Additionally, each node computes and stores the shortest paths to all corresponding cluster heads.

Lemma III.1. *Given the precomputed* $(\log n, 1)$*-NC, it is possible to solve the neighborhood problem in the on-duty scenario, in time* $O(\log n)$ *and using* $O(n_S \cdot \log^2 n)$ *messages.*

Proof: In the runtime phase, the cluster heads serve as local coordination points, where event nodes can learn which of their neighbors sensed the event. This is executed in two stages. In the first one, each event node v sends a message to all cluster heads of the clusters it belongs to. All these cluster heads reply in the second stage with the set of v's neighbors that contacted them.

The time complexity is due to the fact that messages have to be routed to the cluster heads and back, and the diameter of any cluster is at most $O(\log n)$.

For bounding the message complexity, observe that by Definition I.3, each of the n_S nodes in the event component belongs to at most $O(\log n \cdot n^{1/\log n}) = O(\log n)$ clusters, and hence contacts this number of cluster heads. This entails $O(\log n)$ messages (the replies from cluster heads double this amount) and each message is forwarded for $O(\log n)$ hops (as the diameter of each cluster is at most $O(\log n)$). Hence, the total number of message transmissions is $O(n_S \cdot \log^2 n)$. ∎

B. Arboricity Based Discovery

In this section we show a neighborhood discovery algorithm that works even in the restricted off-duty scenario. We show how nodes can pre-compute a list of neighbors they will contact if they get activated by the event.

During the preprocessing phase, we compute respective rooted spanning forests $\mathcal{F} = \{F_1, F_2, .., F_\alpha\}$. Such a decomposition can be computed in polynomial time [10], [24]. For any node v, we define a set $P_v = \{w \,|\, w \text{ is a parent of } v \text{ in some } F_j\}$.

Lemma III.2. *Given the precomputed set $\mathcal{F}$ of α forests covering G, it is possible to solve the neighborhood problem in the off-duty scenario, in 2 rounds using $O(n_S \cdot \alpha)$ messages.*

Proof: In the first round, each event node v sends a *hello* message to all its neighbors from P_v. At the same time it receives similar messages from some of its event neighbors. Those event nodes that receive *hello* messages reply in the second round. We observe that each event node receives a *hello* message or a reply from all neighbors that are event nodes, and thus may effectively learn its neighborhood. As each event node v sends $|P_v| \leq \alpha$ messages in the first round and they are followed by the same number of replies, the total communication complexity is $O(n_S \cdot \alpha)$. ∎

IV. Algorithms for Arbitrary Graphs

For general graphs, we present two solutions, one for the on-duty and one for the off-duty scenario; both try to strike a balance between the time and message complexities. Further, we study algorithms that are based on the breadth/depth first search routines; these algorithms are useful primarily when we want to optimize time or communication complexity alone.

A. On-Duty: Algorithm Decomp

We start with a description of a randomized distributed algorithm Decomp that requires on-duty scenario capabilities. Its running time is linear in the weak diameter of the event component S, and the message complexity is linear in the component's size, both up to polylogarithmic factors. This is asymptotically optimal up to polylogarithmic factors since the exploration of a graph requires at least time $d = \mathrm{diam}(S)$ and n_S messages.

Algorithm Decomp builds on the knowledge passing paradigm. Its execution consists of epochs in which only some event nodes are in *active* state and the knowledge about event components is stored only on them. In the beginning of this process, all nodes are active and each of them is aware only of itself, and in the end a single event node remains active and it knows the entire subgraph $G(S)$.

We emphasize that in our construction, even when executed on a single event component, it can happen that the set of the nodes active in epoch $i+1$ is not a subset of nodes active in epoch i. Not even the cardinalities of active sets are required to monotonically decrease during consecutive epochs.

The challenge is to organize the knowledge passing process so that — on the one hand — no knowledge about S is lost (i.e., the active nodes altogether always know all event nodes) and — on the other hand — the number of active nodes drops quickly (so that the number of transmitted messages is minimized). In the following, we will first present the preprocessing phase and subsequently discuss the runtime phase of Decomp.

Decomp *Preprocessing Phase.* To use the neighborhood discovery routines described in Section III-A, Decomp has to compute sparse covers $(\log n, 1)$-NC in the preprocessing phase. In addition, it also computes a hierarchy of sparse neighborhood covers for exponentially increasing cover diameters, i.e., the decompositions $\mathcal{D}_i := (\log n, 2^i)$-NC for $i \in \{0, \ldots, \lceil \log(\mathrm{diam}(V)) \rceil + 1\}$. Each node also computes the shortest paths to the cluster heads of all clusters it belongs to (e.g., using Dijkstra's single-source shortest path algorithm [6]). These paths may contain nodes outside the clusters. Additionally, each node stores shortest path distances between all pairs of nodes.

Maximal Independent Sets. Besides pre-computed sparse neighborhood covers, Decomp relies on the concept of *Maximal Independent Sets (MIS)*. Decomp computes multiple MIS during the runtime phase in a distributed manner. In our application, we will compute maximal independent sets for the subgraph induced by event nodes. Furthermore, we generalize this notion: a k-MIS is a subset L of event nodes, such that for each node $v \in L$ its k-neighborhood does not contain any other node from L. We require maximality: each event node belongs to a k-neighborhood of some node from L. In these terms, 0-MIS is the set of all event nodes, whereas 1-MIS is a regular MIS.

Alternatively, one could define k-MIS in the following way. Fix an unweighted graph G. Let $G^{\leq k}$ be a graph on the same set of vertices, where two nodes are connected if there is a path between them of length at most k in G. Then L is a k-MIS of a given node set in graph G if and only if it is a MIS of this set in graph $G^{\leq k}$.

For any set A, we define $\Gamma(A) = \{w' \notin A : w \in A \wedge (w, w') \in E\}$ as the set of A's neighbors not in A. Furthermore, we define $\delta(A) = (A, \Gamma(A))$, i.e., $\delta(A)$ contains information about A and its immediate neighbors.

Decomp *Runtime Phase: High-Level Description.* The execution of the algorithm Decomp is split into epochs, numbered from 1. With high probability, we will maintain the following invariant at the end of each epoch i: the set of active event nodes, denoted L_i, constitutes a t_i-MIS of event nodes, where $t_i = 2^i - 1$. More precisely, at the end of epoch i, each event node v will know its *responsibility area* $R_i(v)$ (a subset of event nodes), such that all areas are disjoint, their union is the set of event nodes, for an inactive node v, $R_i(v) = \emptyset$, and for an active node v, $\{v\} \subseteq R_i(v) \subseteq N_{t_i}(v)$. Furthermore, any active node v knows which of the immediate neighbors of $R_i(v)$ are event nodes.

Algorithm 1 Runtime phase of DECOMP

1: perform neighborhood discovery
2: **for all** $v \in S$ **do**
3: v.active := **true**
4: $R_0(v) = \{v\}$
5: $i \leftarrow 1$
6: **while** $\exists$ active v s.t. $\Gamma(R_{i-1}(v))$ contains an event node
7: **each** active v computes $\delta(R_i(w))$ for all $w \in R_{i-1}(v)$
8: *(by simulating of Luby's MIS algorithm)*
9: **for all** active v **in parallel**
10: v sends $\delta(R_i(w))$ to each $w \in R_{i-1}(v)$
11: **for all** $v \in S$
12: v.active $\leftarrow (R_i(v) \neq \emptyset)$
13: $i \leftarrow i + 1$

Event node $w \in R_{i-1}(v)$	**Node** $v \in L_{i-1}$ **simulating** w
w performs local computation (e.g., chooses random variables)	v performs this local computation on behalf of w.
w sends message to $w' \in R_i(v')$	v sends a message to all cluster heads from $\mathcal{D}_{i+1}$; each cluster head forwards this message to all nodes from L_i contained in its cluster.
w becomes selected to be in t_i-MIS and each of its $G^{\leq t_i}$-neighbors (say, constituting set W') become selected not to be in t_i-MIS.	v sets $R_i(w) = W'$. For any node $w' \in W'$, the node v' responsible for w' sets $R_i(w') = \emptyset$.

TABLE I
EPOCH i OF DECOMP: NODES FROM L_{i-1} SIMULATE THE EXECUTION OF LUBY'S ALGORITHM RUN ON ALL EVENT NODES FOR COMPUTING THEIR t_i-MIS.

Algorithm 1 shows how this invariant can be preserved over the consecutive epochs. At the beginning, we define L_0 as the set of all event nodes, all nodes are active, and $R_0(v) = \{v\}$ for each event node v. Such an L_0 is clearly (the unique) 0-MIS, and thus the invariant holds just before the first epoch.

In epoch i, for each node w, its new responsibility area $R_i(w)$ is computed along with its neighborhood $\Gamma(R_i(w))$. These computations are however not performed by a node w itself, but by the active node v responsible for w, i.e., the node v such that $w \in R_{i-1}(v)$. The details of the corresponding lines 7–8 of Algorithm 1 are postponed to the next subsection. Afterwards the knowledge about $\delta(R_i(w))$ is propagated from v to w. Thus, when this process ends, each node w learns its new responsibility area $R_i(w)$ and w is active if and only if this area is nonempty.

We end this process when one active node recognizes that it knows the entire event component (i.e., its immediate neighborhood does not contain other event nodes).

Computing MIS in a single epoch. Now we explain in detail what happens in the lines 7–8 of Algorithm 1. What we want to compute is a t_i-MIS of set S in graph G or, alternatively speaking, a MIS of S in graph $G^{\leq t_i}$. To this end, we will simulate the standard, randomized *Luby's algorithm* [22] for finding a MIS in $G^{\leq t_i}$. By the analysis of [22], there exists a constant γ, such that after executing $c \cdot \gamma \cdot \log n$ rounds of the algorithm, the algorithm correctly computes a maximal independent set with probability at least $1 - n^{-c}$ (for any integer c). We emphasize that in each epoch a new MIS is computed from scratch not taking into account the current MIS, L_{i-1}.

It remains to show how to simulate a single step of the Luby's algorithm. In such step, nodes perform local computations and communicate with their neighbors (note that these are neighbors in $G^{\leq t_i}$, i.e., nodes whose distance in G is at most t_i). Note that each node knows these neighbors, as all-pairs distances were computed in the preprocessing phase. Furthermore, in each round of the *Luby's algorithm*, some nodes may become selected to be in MIS, and their neighbors (in $G^{\leq t_i}$) become selected to be outside of MIS.

All these actions that in Luby's algorithm are performed by a node w can be simulated by an active node $v \in L_{i-1}$ responsible for w (the one for which it holds $w \in R_{i-1}(v)$), as demonstrated in Table I. The only non-trivial part of the simulation is the transmission of messages sent by w to its $G^{\leq t_i}$-neighbor w'. To simulate this, each node $v \in L_{i-1}$ sends gathered messages of all nodes from its responsibility area $R_{i-1}(v)$ to all the cluster heads from $\mathcal{D}_{i+1}$ it belongs to. If there is no message transmission to be simulated, v sends an empty message. In either case, each cluster head learns all the active nodes belonging to its cluster. Afterwards all cluster heads transmit all the received messages to all nodes from L_{i-1} they are responsible for. If a node $v' \in L_{i-1}$ receives a message and the intended recipient is not in its rensposibility area, it simply discards that message.

Thus, it remains to show that if a node w transmits message to a node w' and $d(w, w') \leq t_i$, then this message is successfully delivered. Let v and v' be the nodes from L_{i-1} responsible for w and w', respectively. Observe that $d(v, v') \leq d(v, w) + d(w, w') + d(w', v') \leq t_{i-1} + t_i + t_{i-1} \leq 2^{i+1} - 3$. Then, by the property of neighborhood covers, there exists a cluster $C \in \mathcal{D}_{i+1}$, such that both v and v' belong to C, and thus the message is successfully delivered to v'.

It is straightforward to guarantee that all nodes simulate the same round of Luby's algorithm, as there is an upper bound on the number of rounds necessary for communicating with cluster heads from $\mathcal{D}_{i+1}$. In the presented scheme, the simulating nodes correctly compute the set $R_i(w)$ for each simulated node w. To additionally compute $\Gamma(R_i(w))$ it is sufficient to always include the neighborhood of a transmitting node in the transmitted message. Figure 2 illustrates the algorithm.

Theorem IV.1. DECOMP *terminates with a correct solution with high probability, and requires* $O(d \cdot \log^2 n)$ *rounds and* $O(n_S \cdot \log^3 n \cdot \log d)$ *messages;* $d = diam(S)$ *is the weak diameter of the event set* S.

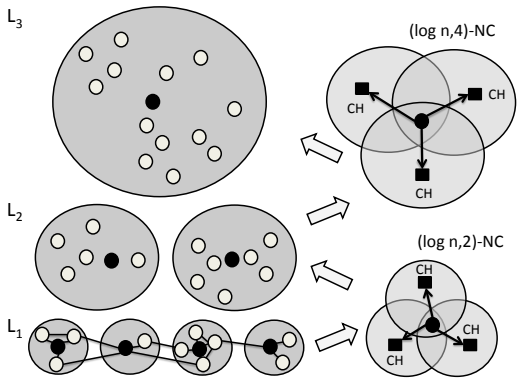

Fig. 2. DECOMP constructs independent sets of larger and larger range (*left*) with fewer active nodes (*dark*). Knowledge is transferred via the neighborhood covers (*right*).

Proof: First, we bound the number of nodes in L_i. Nodes from L_i form a t_i-MIS, and the responsibility area for any node contains at least $\lfloor t_i/2 \rfloor$ event nodes. Thus, the number of nodes in L_i is $O(n_S/t_i) = O(n_S/2^i)$.

Termination and runtime: At latest after epoch $\lceil \log d \rceil$, we get a 1-node active set. Its only node recognizes this situation and terminates. Epoch i takes $O(\log n)$ simulation steps, where each step requires sending messages to cluster heads (at the distance of at most $O(2^i \cdot \log n)$) and receiving responses. Knowledge passing at the end of an epoch requires an additional time of $O(2^i)$. Altogether, the total number of rounds is $O(\sum_{i=1}^{\lceil \log d \rceil} 2^i \cdot \log^2 n) = O(d \cdot \log^2 n)$.

Correctness: The correctness of the algorithm follows directly from the information passing mechanism: no information about event nodes is lost at any epoch. Given that maximal independent sets are computed in each epoch, DECOMP finds the entire event component. Set L_i is computed in epoch i with probability $1 - n^{-c}$ provided that set L_{i-1} was computed correctly in epoch $i-1$. As the algorithm terminates after $\lceil \log d \rceil \leq n$ many rounds, it manages to compute a correct maximal independent set in *all* epochs (and thus compute a correct solution) with probability at least $1 - n^{-c+1}$.

Message complexity: For a single round of the MIS simulation, each node from L_{i-1} communicates with $O(\log n)$ cluster heads, each at distance $O(2^i \cdot \log n)$. The responses from cluster heads can be bounded analogously. The knowledge passing complexity (at the end of an epoch) requires sending $|L_i|$ messages along at most t_{i-1} hops. Thus, the whole communication in epoch i, which has $O(\log n)$ rounds, takes $O(\log n \cdot (n_S/2^i) \cdot 2^i \cdot \log n \cdot \log n) = O(n_S \cdot \log^3 n)$ messages. As there are at most $\lceil \log d \rceil$ epochs, the total number of messages is $O(n_S \cdot \log^3 n \cdot \log d)$. ∎

Note that in the unlikely event that DECOMP does not compute a correct solution, at least one node can locally detect that the independent set is not maximal, and raise an alarm to trigger a recomputation. Using fast and slow transmissions, the algorithm can hence guarantee the correctness with probability 1 at the expense of increased runtime, which is then bound *with high probability*.

B. The Off-Duty Scenario

Let us now turn our attention to the off-duty scenario where only event nodes can participate in the distributed alarming. In the remaining part of this section we use the neighborhood discovery technique from Section III in an algorithm MINID. This algorithm's performance depends, besides n_S, only on the arboricity of the graph.

In the preprocessing phase of MINID, the algorithm computes α trees $\{F_j\}_{j=1}^{\alpha}$ covering the whole graph as described in Section III-B. Then, at the beginning of the runtime phase, MINID runs an arboricity-based neighborhood discovery (cf Section III-B).

First, we present the algorithm under the assumption that n_S is known; later we show that this assumption is not critical for our analysis.

The discovery of the connected set of event nodes is performed by leader election, where the node with the smallest index distributes its index to everyone else in the event set. The algorithm proceeds in $2 \log n_S$ epochs. Initially, each event node v constitutes its own cluster $C_v = \{v\}$, with v acting as the leader. In the course of an epoch, the number of clusters is reduced, so that after at most $2 \log n_S$ epochs a single cluster containing all event nodes in the set is formed. At any time two clusters C_i and C_j are neighbors if there exists an edge (v, w) connecting two event nodes $v \in C_i$ and $w \in C_j$.

We also assume that before entering a new epoch each cluster is supported by a spanning tree rooted at the leader. Note that all nodes in the cluster can be visited in time at most $2n_S$, e.g., by a DFS routine emulated with the help of a token released by the leader. Each cluster C_i is visited by the token three times. During the first visit at each node $v \in C_i$, the token distributes the index of C_i's leader to the entire C_i and to all event neighbors of C_i in different clusters. During the second visit, the token collects information about indices of neighboring clusters and it picks the C_j with the smallest index j. If $j < i$, during the third consecutive visit, the token distributes j to all nodes in C_i to inform them that they are now destined for C_j.

Let G_C be a digraph in which the set of nodes is formed of clusters C_i and where there is an arc from C_j to C_i iff nodes of C_i are destined for C_j. A node C_w with in-degree 0 in G_C corresponds to a cluster that during this epoch spreads its index to all other clusters reachable from C_w according to the directed connections in G_C. Note also that since the maximum in-degree of nodes in G_C is 1, each cluster with in-degree 1 will receive a new index from exactly one cluster. The process of reindexing is performed by a DFS procedure initiated by the leader in each cluster C_w with in-degree 0 in G_C and it is extended to the nodes of all (not only directly neighboring) clusters reachable from C_w (according to the connections in G_C).

Recall that the procedure presented above works under the assumption that the value of n_S is known in advance.

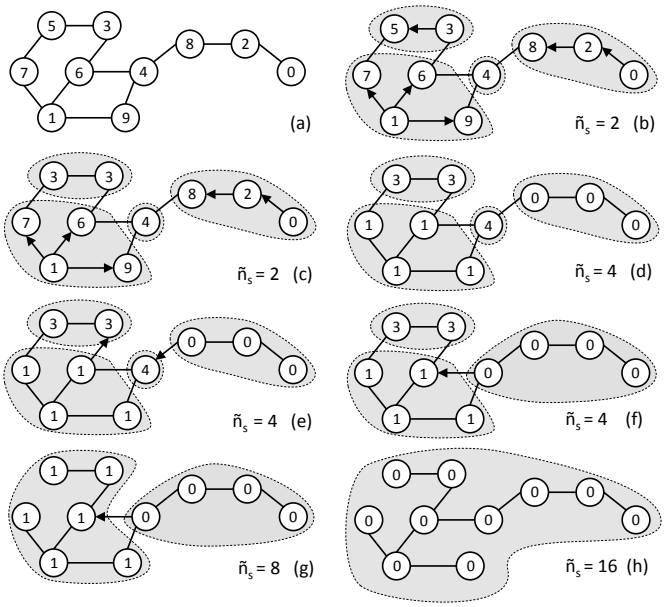

Fig. 3. An illustration of algorithm MinID. An example graph with node IDs is shown in (a). In (b), it is shown how nodes choose to which neighbors they want to be destined, and we can also see what clusters are formed after one loop if $\tilde{n}_s$ is sufficiently large. In (c), we see that only node 3 succeeds to spread its ID in the whole cluster, and the situation until $\tilde{n}_s$ increases is depicted. In (d), we see how the situation develops from (b) for $\tilde{n}_s = 4$: nodes 0, 1, 3 manage to spread their IDs. Afterwards, the whole cluster 3 is destined for cluster 1, and cluster 4 is destined for cluster 0 which is shown in (e), however, only node 0 manages to spread its ID. The final situation for this value of $\tilde{n}_s$ is shown in (f). In (g), we can see the situation after (e) for $\tilde{n}_s = 8$, finishing the execution. In (h), we can see how the situation develops if $\tilde{n}_s = 16$, i.e., for the first value that is actually larger than the number of nodes.

Since this is not the case, we take an exponentially increasing sequence of upper bounds $2, 4, .., 2^i, .., 2^{\lceil \log n \rceil}$ on n_S, and run our algorithm assuming these consecutive powers of two, until the correct bound on n_S is found. Note that when the algorithm runs with a wrong assumption on the size of the event set, the nodes eventually learn that the set is larger than expected. The nodes in clusters that are about to expand too much are informed by their leaders, and the nodes destined for other clusters, if not contacted by the new leader on time, also conclude that the bound on n_S is inappropriate. Thus, the process continues until the appropriate bound on n_S is found and then it is stopped.

An example execution of the algorithm MinID is given in Figure 3.

If we could show for a single epoch that each cluster either delegates its index to at least one other cluster, or assumes the index of another cluster, this would prove that the number of clusters is reduced to at most half of their number from the previous epoch.

However, there may exist clusters whose indices are local minima, i.e., they have the smallest index in their neighborhood in G_C, but each of their neighbors has another neighbor with yet a smaller index and chooses to accept that neighbor's index. Each such local minimum will have a neighbor with a smaller index in the next epoch, as all its neighbors will accept smaller indices. Thus within two epochs each cluster either grows or is removed, from which follows the following lemma:

Lemma IV.2. *During two consecutive epochs of* MinID *the number of clusters is reduced by half as long as it is larger than* 1.

Theorem IV.3. *In a graph G with arboricity α, the deterministic algorithm* MinID *finishes in $O(n_S \log n_S)$ rounds using $O(m_S \log n_S + \alpha \cdot n_S)$ messages.*

Proof: For an assumed bound n_S on the size of the component and for a single epoch, the three visits along an Euler tour followed by reindexing take time $O(n_S)$ and incur total communication complexity of $O(m_S)$, since each edge is traversed a constant number of times. According to Lemma IV.2, after at most $2 \log n_S$ epochs there is exactly one cluster. Hence, if the bound n_S is correct, the total time is $O(n_S \log n_S)$ and the total communication $O(m_S \log n_S)$.

Therefore in total the time complexity for an event set of size n_S is bounded by $\sum_{i=1}^{\lceil \log n_S \rceil} O(2^i \cdot \log 2^i) = O(n_S \log n_S)$. Similarly, the total communication is $O(m_S \log n_S)$. By adding the complexity of neighborhood discovery (cf. Lemma III.2), the claim follows. ∎

Note that arboricity of a planar graph is 3 [24]. Thus, MinID runs in time $O(n_S \log n_S)$ using $O(n_S \log n_S)$ messages in planar graphs as shown in Figure 6.

C. Search-Based Algorithms

To complement the results on general graphs from Section IV-A and Section IV-B, we study algorithms for arbitrary graphs based on depth/breadth first search routines. They are inferior to the already presented results if we consider the product of number of rounds and the number of messages as the performance metric, but can be useful if one wants to optimize the time or message complexity alone. We provide a qualitative comparison in Figure 4 and Figure 5.

First, we note that it is possible to implement the DFS and BFS routines in a distributed fashion in our environment.

Lemma IV.4. *In both off-duty and on-duty scenarios, a distributed* DFS *procedure initiated at a single event node finishes in time $O(n_S)$ using $O(n)$ messages. If each event node knows its event neighbors, the message complexity can be reduced to $O(n_S)$.*

Proof: First, we assume that nodes do not know their event neighborhoods. We fix any starting event node. We say that this node holds the "token": the token indicates the node that would be processed in the centralized DFS. This token represents the current knowledge about all nodes: nodes are either known to be event nodes, known to be non-event ones, or of unknown state; this knowledge is passed on with the token. During our procedure, the token node tries to forward the token to the neighbor that would be next in the DFS tree. This is done as follows. First,

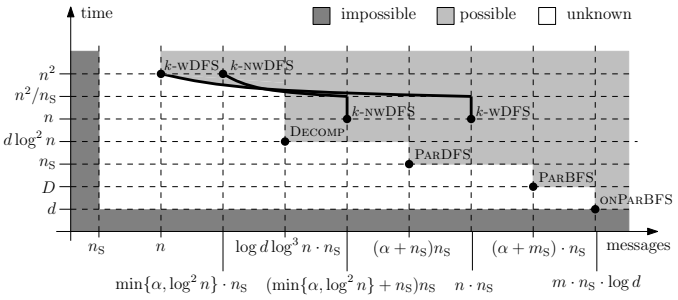

Fig. 4. Overview of formal results and comparison of different algorithms for arbitrary graphs in the on-duty scenario. All the values are asymptotic ones. The complexities of algorithm Decomp are deterministic, but the algorithm may err with some small, inversely polynomial probability.

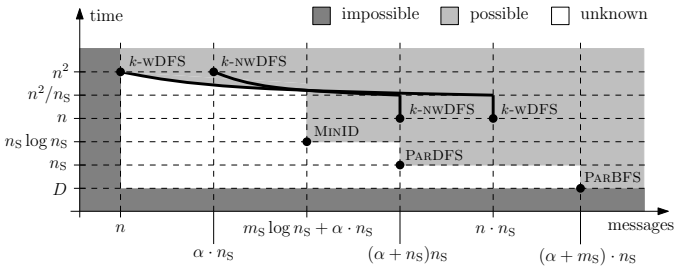

Fig. 5. Overview of results and comparison of different algorithms for arbitrary graphs in the off-duty scenario. All the values are asymptotic ones. The lower bound of $\Omega(n)$ on the number of messages holds, e.g., for cliques, by Corollary II.3. The lower bound of $\Omega(n \log \log n)$ on the time-message product in cliques (cf Theorem II.4) is not depicted in this figure.

the token node "pings" all its neighbors with unknown state and event neighbors respond immediately. Then, as in the centralized DFS algorithm the token is passed to any unvisited event node, and if there is none, the token is sent back to the node it came from. As DFS proceeds along the DFS tree spanning all event nodes in a single component, it takes time $O(n_S)$. In the process of gaining knowledge each node changes its state just once, so the number of messages is $O(n)$.

Second, we observe that in case of a known neighborhood, the part with pinging all neighbors can be omitted, and the token can be passed to a non-visited event neighbor. This reduces the number of messages to $O(n_S)$. ∎

Lemma IV.5. *In both off-duty and on-duty scenarios, a distributed* BFS *procedure uses $O(D)$ rounds and $O(m)$ messages. If each event node knows its event neighbors, the messages complexity can be reduced to $O(m_S)$. Furthermore, the time complexity can be reduced to $O(d)$ in the on-duty scenario, but the number of messages remains $O(m)$ (even if each node knows its event neighborhood); the resulting procedure is called* onBFS*.*

Proof: A BFS procedure is just a simple flooding operation and its time and message complexities are immediate. Algorithm onBFS floods also non-event nodes. ∎

The DFS, BFS and onBFS procedures are useful if there is a predefined leader. In our setting, there is no such distinguished node, and hence we have to start this procedure at multiple event nodes, in parallel, sequentially, or a mixture of the two. If our primary goal is to optimize the runtime, we should parallelize as many procedures as possible: in fact, we may run independent DFS, BFS or onBFS routines from each event node. The resulting algorithms will be called ParDFS, ParBFS, and onParBFS, respectively.

If we run these algorithms without neighborhood discovery, the total number of used messages for any of them will be $\Omega(n \cdot n_S)$. Thus, we can run any neighborhood discovery beforehand without increasing their asymptotic message complexity. We choose arboricity based discovery as it is available in both on-duty and off-duty scenarios and has a lower runtime.

Theorem IV.6. *The algorithms* ParDFS, ParBFS *and* onParBFS *solve the problem in time $O(n_S)$, $O(D)$, and $O(d)$, respectively, using $O(n_S \cdot (\alpha + n_S))$, $O(n_S \cdot (\alpha + m_S))$, and $O(n_S \cdot m))$ messages. The first two algorithms can be used in both the on-duty and the off-duty scenario, the third is for the on-duty scenario only.*

Proof: For all algorithms, we start with an arboricity based neighborhood discovery as described in Section III-B which takes two rounds and $O(\alpha \cdot n_S)$ messages. The time complexity is simply the worst-case complexities of a single DFS, BFS, or onBFS procedure, respectively. On the other hand, the total number of messages is the number of messages used by these procedures, times the number of event nodes n_S. For the performance of the onParBFS algorithm we observe that $O(n_S \cdot (\alpha + m)) = O(n_S \cdot m)$ as $\alpha \leq n \leq m$. ∎

One way to reduce the number of messages is to have a graded start of the DFS procedures. Of course, as we do not know which nodes are event ones, we may need to wait for potentially non-event nodes. Concretely, in our algorithm k-wDFS (where $k \in \{1, \ldots, n\}$), we divide time into $\lceil n/k \rceil$ epochs of length $\Theta(n)$. This length is chosen in such a way that for any choice of event nodes, the worst-case execution of a DFS initiated at any event node ends within one epoch. In epoch i, we call event nodes with identifiers between $k \cdot (i-1) + 1$ and $\min\{k \cdot i, n\}$ *busy*. All busy nodes start their DFS procedures. The algorithm terminates in the first epoch, in which there was any busy node.

Theorem IV.7. *For any integer $k \in \{1, \ldots, n\}$, the algorithm k-*wDFS *solves the problem in $O(n^2/k)$ rounds using $O(\min\{k, n_S\} \cdot n)$ messages.*

Proof: The algorithm terminates in the first epoch, in which there was any busy node. In the worst-case, the algorithm finishes after $\lceil n/k \rceil$ epochs, i.e., after $O(n^2/k)$ rounds. In this epoch, all busy nodes (at most $\min\{k, n_S\}$ of them) start their DFS procedure, transmitting in total $O(\min\{k, n_S\} \cdot n)$ messages. ∎

Again, the message complexity can be reduced to $O(\min\{k, n_S\} \cdot n_S)$ if the algorithm is preceded by the

neighborhood discovery routine. As the runtime of the k-wDFS is already $\Omega(n)$ for any k, we may choose either of the two routines of Section III without increasing its asymptotic runtime. Note that their message complexities are $O(n_S \cdot \log^2 n)$ and $O(n_S \cdot \alpha)$ and hence in the on-duty scenario, we may choose the more efficient one, while in the off-duty scenario, we have to use the latter one. We call the resulting algorithm k-nwDFS, immediately obtaining the following result.

Theorem IV.8. *For any integer $k \in \{1, \ldots, n\}$, the algorithm k-nwDFS solves the problem in $O(n^2/k)$ rounds. In the off-duty scenario, it uses $O((\alpha + \min\{k, n_S\}) \cdot n_S)$ messages while in the on-duty scenario it uses $O((\min\{\alpha, \log^2 n\} + \min\{k, n_S\}) \cdot n_S)$ messages.*

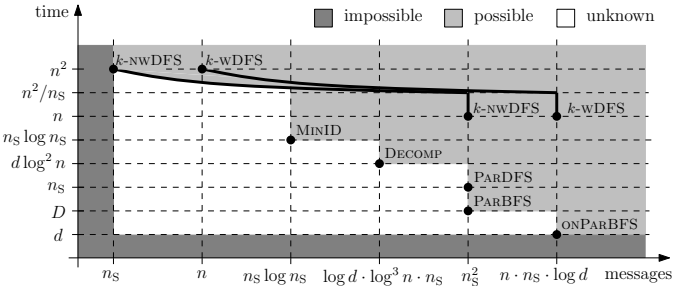

Fig. 6. Overview of results and comparison of different algorithms for planar graphs in both off-duty and on-duty scenarios. Algorithms Decomp and onParBFS use the capabilities of the on-duty scenario. All the values are asymptotic ones. The depicted performance of the algorithms is an immediate consequence of setting $\alpha = O(1)$ and $m_S = O(n_S)$. The algorithms k-wDFS and ParDFS are superseded by others.

V. Handling Multiple Components

We conclude our technical contribution with a discussion of scenarios where event nodes S constitute multiple connected components. In this scenario, we aim to ensure that at least one node in each component knows this entire component. First, note that such an extension does not affect the off-duty algorithms (even when run in on-duty scenarios) as event nodes from two different components do not interact.

To make Decomp work for multiple components, we can insert a special cleanup stage at the end of each epoch i. The general idea is that if it is possible to identify a whole event component, the nodes of this component should be instructed to switch to *idle* state; henceforth, they act as if they were non-event nodes. Specifically, such a verification can be performed in each epoch i, right after line 12 of Algorithm 1. To this end, each active node v forwards its new responsibility areas $\delta(R_i(v))$ to heads of all clusters from $\mathcal{D}_{i+1}$ it belong to. Thus, each cluster head has a complete picture of all nodes that are in its cluster plus their immediate neighborhoods. For each event component $S' \subseteq S$ that is contained entirely in the cluster, the cluster head sends the description of S' to all active nodes (in its cluster) whose responsibility areas contain some nodes of S'. Finally, each such active node v tells all nodes from $\delta(R_i(v)) \cap S'$ to switch to idle state and removes them from its responsibility area. If, in effect, v's responsibility area becomes empty, it switches to inactive state. (Note that if $v \in S'$, then it is possible that it remains active although it became idle: it will play its usual role in choosing the independent set L_{i+1} in the next epoch.) The introduced change also modifies slightly the meaning of the main *while* loop condition (line 8 of Algorithm 1): the algorithm may now terminate because there are no more active nodes.

VI. Conclusion

We consider our work as a first step to shed light onto the important problem of distributed alarming. We identify two main directions for future research. First, many of our results do not come with lower bounds, and it remains to study the optimality of our solutions and to close possible gaps. That said, our algorithms perform quite well, e.g., for the case of planar graphs, cf Figure 6.

Second, our models are still simple and, depending on the application, additional aspects may have to be taken into account. For instance, for wireless network scenarios, our models need to be extended with an appropriate interference model. However, to some extent, the design of efficient medium access schemes is orthogonal to our approach, in the sense that our algorithms can be combined with existing algorithms, e.g., [26]: the resulting time and message complexities must be multiplied by the medium access overhead. Finally, our algorithms should be extended to be able to deal with misbehaving or malfunctioning nodes that do not cooperate despite having sensed an event.

References

[1] A. A. Abbasi and M. Younis. A survey on clustering algorithms for wireless sensor networks. *Comput. Commun.*, 30:2826–2841, 2007.

[2] N. Alon and J. Spencer. *The Probabilistic Method*. John Wiley, 1991.

[3] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Fast network decomposition. In *Proc. 11th ACM Symp. on Principles of Distributed Computing (PODC)*, pages 169–177, 1992.

[4] B. Awerbuch and D. Peleg. Sparse partitions (extended abstract). In *Proc. IEEE FOCS*, pages 503–513, 1990.

[5] Y. Birk, I. Keidar, L. Liss, A. Schuster, and R. Wolff. Veracity radius: Capturing the locality of distributed computations. In *Proc. 25th ACM Symp. on Principles of Distributed Computing (PODC)*, pages 102–111, 2006.

[6] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[7] A. Czygrinow, M. Hańćkowiak, and E. Szymańska. Fast distributed approximation algorithm for the maximum matching problem in bounded arboricity graphs. In *Proc. 20th International Symposium on Algorithms and Computation (ISAAC)*, pages 668–678, 2009.

[8] A. Davison. Laptops as earthquake sensors. In *MIT Technology Review*, 2008. http://www.technologyreview.com/computing/20658/.

[9] M. Elkin. A faster distributed protocol for constructing a minimum spanning tree. In *Proc. 15th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 359–368, 2004.

[10] H. N. Gabow and H. H. Westermann. Forests, frames, and games: Algorithms for matroid sums and applications. *Algorithmica*, 7(1):465–497, 1992.

[11] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983.

[12] M. Göös and J. Suomela. Locally checkable proofs. In *Proc. 30th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 159–168, 2011.

[13] M. Grohe, S. Kreutzer, and S. Siebertz. Deciding first-order properties of nowhere dense graphs. In *Proc. 46th Annual Symposium on the Theory of Computing (STOC)*, 2014.

[14] H.-A. Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In *Proc. 13th Usenix Security Symposium*, pages 271–286, 2004.

[15] D. E. Knuth. *The art of computer programming, volume 3: (2nd ed.) sorting and searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[16] A. Korman and S. Kutten. Distributed verification of minimum spanning trees. In *Proc. 25th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 26–34, 2006.

[17] A. Korman, S. Kutten, and D. Peleg. Proof labeling schemes. *J. Distributed Computing*, 22(4):215–233, 2010.

[18] K. Kothapalli and S. Pemmaraju. Distributed graph coloring in a few rounds. In *Proc. 30th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 31–40, 2011.

[19] F. Kuhn, T. Locher, and R. Wattenhofer. Tight bounds for distributed selection. In *Proc. 19th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, 2007.

[20] F. Kuhn, T. Moscibroda, and R. Wattenhofer. What cannot be computed locally! In *Proc. 23rd ACM Symp. on the Principles of Distributed Computing (PODC)*, 2004.

[21] C. Lenzen and R. Wattenhofer. Minimum dominating set approximation in graphs of bounded arboricity. In *Proc. 24th International Conference on Distributed Computing (DISC)*, pages 510–524, 2010.

[22] M. Luby. A simple parallel algorithm for the maximal independent set problem. In *Proc. 17th Annual ACM Symp. on Theory of Computing (STOC)*, pages 1–10, 1985.

[23] S. Massoud Amin and B. Wollenberg. Toward a smart grid: power delivery for the 21st century. *Power and Energy Magazine, IEEE*, 3(5):34–41, 2005.

[24] C. S. J. Nash-Williams. Edge-disjoint spanning trees of finite graphs. *J. of London Mathematical Society*, 36:445–450, 1961.

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

[26] A. Richa, C. Scheideler, S. Schmid, and J. Zhang. An efficient and fair mac protocol robust to reactive interference. *IEEE/ACM Transactions on Networking (ToN)*, 2013.

[27] S. Schmid and J. Suomela. Exploiting locality in distributed sdn control. In *Proc. ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, August 2013.

[28] J. Suomela. Survey of local algorithms. *ACM Comput. Surv.*, 45(2):24:1–24:40, 2013.

Dr. Marcin Bienkowski is currently an assistant professor in the Combinatorial Optimization Group of the Computer Science and Mathematics department at the University of Wroclaw, Poland. He received his Ph.D. degree in 2005 from the University of Paderborn, Germany, where he worked in the Algorithms and the Complexity Theory group. His research interests focus on online and approximation algorithms.

Prof. Leszek Gąsieniec received MSc (1991) and PhD (1994) degrees in Computer Science from Warsaw University in Poland. He was later a postdoctoral researcher in Quebec University (94-95) in Canada, and in Max-Planck Institute für Informatik (95-97) in Germany. Since 1997 he has been with University of Liverpool where he is now a full professor and the head of Department of Computer Science. The main strand of his research explorations include parallel and distributed algorithms.

Dr. Marek Klonowski received Ph.D. degrees in computer science from Adam Mickiewicz University (2003) and in mathematics from Wrocław University of Technology (WUT). Habilitation in 2012 from IPIPAN (Warsaw). Since 2012 he has been an associate professor at WUT. His current research interests include formal analysis of distributed, probabilistic protocols.

Dr. Miroslaw Korzeniowski Miroslaw Korzeniowski was born in Wroclaw in Poland in 1978. He studied at the University of Wroclaw and as an exchange student at the University of Paderborn, and defended his master thesis in 2002. He did his PhD under the supervision of Friedhelm Meyer auf der Heide at the University of Paderborn in Germany. Currently he is an assistant professor at the Wroclaw University of Technology where he works on heterogeneity in distributed systems.

Prof. Bernard Mans is Professor, and currently Head of Department, for the Department of Computing at Macquarie University, Sydney, Australia, which he joined in 1997. His research interests centre on algorithms and graphs for distributed and mobile computing, and in particular on wireless networks. In 2003, he was the HITACHI-INRIA Chair at INRIA, France. He received his Ph.D. in Computer Science from University Pierre et Marie Curie, while at INRIA France, in 1992.

Dr. Stefan Schmid is a senior research scientist at the Telekom Innovation Laboratories (T-Labs) and at TU Berlin, Germany. He received his MSc and Ph.D. degrees from ETH Zurich, Switzerland, in 2004 and 2008, respectively. 2008-2009 he was a postdoctoral researcher at TU Munich and at the University of Paderborn, Germany. Stefan Schmid is interested in distributed systems, algorithms, and incentive aspects.

Prof. Roger Wattenhofer is a full professor at the Information Technology and Electrical Engineering Department, ETH Zurich, Switzerland. He received his doctorate in Computer Science in 1998 from ETH Zurich. From 1999 to 2001 he was in the USA, first at Brown University in Providence, RI, then at Microsoft Research in Redmond, WA. He then returned to ETH Zurich, originally as an assistant professor at the Computer Science Department. Roger Wattenhofer's research interests are a variety of algorithmic and systems aspects in computer science and information technology.