How to Wake up Your Neighbors: Safe and Nearly Optimal Generic Energy Conservation in Radio Networks

Varsha Dani ✉
Department of Computer Science, Rochester Institute of Technology, NY, USA

Thomas P. Hayes ✉
Department of Computer Science, University at Buffalo, NY, USA

──── **Abstract** ────

Recent work [7, 8, 11] has shown that it is sometimes feasible to significantly reduce the energy usage of some radio-network algorithms by adaptively powering down the radio receiver when it is not needed. Although past work has focused on modifying specific network algorithms in this way, we now ask the question of whether this problem can be solved in a generic way, treating the algorithm as a kind of black box.

We are able to answer this question in the affirmative, presenting a new general way to modify arbitrary radio-network algorithms in an attempt to save energy. At the expense of a small increase in the time complexity, we can provably reduce the energy usage to an extent that is provably nearly optimal within a certain class of general-purpose algorithms.

As an application, we show that our algorithm reduces the energy cost of breadth-first search in radio networks from the previous best bound of $2^{O(\sqrt{\log n})}$ to polylog(n), where n is the number of nodes in the network

A key ingredient in our algorithm is hierarchical clustering based on additive Voronoi decomposition done at multiple scales. Similar clustering algorithms have been used in other recent work on energy-aware computation in radio networks, but we believe the specific approach presented here may be of independent interest.

1 Introduction

For large networks of tiny sensors equipped with radio transceivers, the energy used in communication can be a bottleneck cost. Although it has become standard practice to measure the cost of communication in terms of the number of messages or bits sent, it has been pointed out (see, for instance, [2, 30]) that the cost of using the receiver to listen for messages is often comparable to, or even more than, the cost of transmission. This becomes even more true as the size of the devices is scaled down.

Obviously, this fact suggests that we should try hard to avoid using our receiver to listen for messages except at times when one is being sent. However, this is easier said than done! In general, effective economization of receiver use probably requires redesigning our communication protocols from the ground up. As an example, previous work by Dani, Gupta, Hayes and Pettie [11] on solving the maximal matchings problem on a radio network with low energy expenditure, presented an algorithm that was very specific to its problem, in the sense that the algorithm is able to cleverly combine efficient message delivery with energy conservation and residual degree manipulation, all at once.

Nevertheless, it is tempting to ask whether, in spite of this, some general-purpose technique can be used to minimize receiver usage when no messages are being sent nearby. In this paper we give a positive answer to this question. More precisely, we present a generic way to convert any given protocol into one that attempts to use its receiver more wisely, in a way tailored to the activity pattern of the given algorithm. The amount of energy saved depends on properties of the algorithm being simulated. For some protocols, our construction will actually make them less energy efficient. However, for at least a few protocols of interest, our technique improves the overall energy cost on size n networks from $\text{poly}(n)$ to $\text{polylog}(n)$. Moreover, there is a sense in which our algorithm is nearly the best possible, as we shall see.

This potential decrease in overall energy cost is not completely free. Our algorithm incurs modest (up to a $\text{polylog}(n)$ factor) penalties in terms of overall running time, (sent) message complexity, and local computation. These costs should be weighed against the possible benefits before deciding whether to deploy our algorithm in a particular context.

We use a simple abstract model of distributed computation on Radio Networks [10], specifically the variant introduced in [7]. In this model, local computation is treated as free, and in each of a series of globally synchronized timesteps, each processor decides whether to SEND, LISTEN, or SLEEP. Sending and Listening cost 1 unit of energy, but sleeping is free. Messages can be up to $O(\log n)$ bits long. In this setting, we try to simultaneously minimize the total time complexity of our protocol, as well as the maximum per-node energy usage.

The motivation for our new construction comes from the structure of the simplest possible network algorithm: a naive broadcast protocol, in which a message is repeatedly transmitted across the network at a rate of unit distance per timestep. Considering the behavior of the "active set," that is, the nodes that are actually transmitting or receiving a message in a given timestep, we see that it behaves like a ripple moving across a pool of water, or the frontier of a parallel BFS algorithm, or even a light-cone propagating across spacetime.

One aspect of this is that the active set moves with a known finite speed, which can potentially allow us to shut down nodes when the active set is known to be far away from them. In particular, any time a node is at distance at least d from the active set, it can afford to sleep for at least d timesteps, because information cannot be transmitted faster than one edge per timestep. Of course, to use this in practice, our hypothetical node needs some way of getting advance warning of how far it is from the active set.

We achieve this by organizing nodes into clusters that spread information about the location of the active set faster than the "speed of light" in our simulated protocol. Or, to put it more prosaically, at the cost of increased latency, our network is able to transmit advance notice of the active set before it arrives. To save as much energy as possible, we try to do this at all distance scales simultaneously. Thus, while nodes whose distance to the active set is only a few dozen may only know they can safely sleep for a dozen timesteps; at the same time, nodes whose distance to the active set is in the thousands may know they can safely sleep for hundreds of timesteps. At a high level, this sounds very similar to the approach used by Chang, Dani, Hayes and Pettie [8] for reducing the energy cost of BFS in the same model. Indeed, the main difference in our current approach is in the way the clusters at different scales are generated and communicate with each other. As a result of these differences, we are able to exponentially improve on the energy usage of BFS, while at the same time, being immediately applicable to a broader range of algorithms.

We present a new general-purpose algorithm which we call SAF Simulation (see Algorithm 2) after its mnemonic, "Sleep when Activity is Far." SAF Simulation accomplishes both the goal of building layered cluster decompositions, as well as using this structure to effectively simulate a large class of algorithms. Our main result can be summarized (at a high level) as

▶ **Theorem 1.** *For every Radio Network algorithm, A, SAF(A) runs in time $O(\mathrm{polylog}(n)\,\mathrm{TIME}(A))$, and with probability $1 - 1/\mathrm{poly}(n)$ produces the same output as A. Moreover, its energy cost satisfies, for every vertex v,*

$$\mathrm{ENERGY}(\mathrm{SAF}(A), v) \leq \mathrm{polylog}(n)\mathrm{OPT}(A, v),$$

where OPT *is the least possible energy cost for a safe, synchronized, one-pass generic simulation algorithm.*

A more precise (and technical) statement appears in Theorem 14. Some specific problems that can be solved by Algorithm 2 for only $\mathrm{polylog}(n)$ per-node energy include: Broadcast, BFS, Leader Election, and Approximate Counting. Loosely speaking, the same can be said for any algorithm that does the bulk of its computation in $\mathrm{polylog}(n)$ BFS-like sweeps across the graph. In all cases, this represents an exponential improvement in energy usage, as compared to the naive implementation.

Comparison to Previous Work

Our main tool for clustering our nodes is the ESTCluster algorithm of Miller, Peng and Xu [26] (see also [27].) This results in clusters that are the cells of a kind of generalized Voronoi diagram, as we discuss in Section 3.3. These clusters were used previously in the context of low energy computation in the same radio network model by Chang, Dani, Hayes and Pettie [8] for the problem of constructing breadth-first search.

In that work, they estimated long-range distances by recursively solving BFS on their level-1 cluster graph. This has the apparent advantage that, in the resulting hierarchy of clusters, each cluster is a coarsening of the previous lower-level cluster. However, it has the disadvantage that the distortion of distances accumulates multiplicatively as one moves up levels. This limits its usefulness for putting processors to Sleep, since one cannot risk missing the arrival of the active set.

In the current work, by contrast, instead of recursively constructing clusters of clusters, we instead run the ESTCluster algorithm on the base graph with different radius parameters, to generate our entire hierarchy of clusters. An apparent disadvantage of this is that the resulting clusterings are not coarsenings and refinements of each other. Fortunately, it turns out that we do not need this property. Although it would be possible to use a sort of rounding to replace our clusterings with ones that are nested, there are algorithmic disadvantages to doing so. Specifically, one nice thing about the generalized Voronoi cells created by the ESTCluster algorithm is that they are star-shaped, so the entire cluster is connected by a BFS spanning tree rooted at the cluster center. This property would be lost if redefined the clustering to be a coarsening of the lower-level clusterings. This would appear to necessitate neighboring clusters to assist in what was previously intra-cluster communications.

Other related work on energy complexity in radio networks

A number of authors have studied energy complexity on *single-hop* radio networks. Here the energy model is still that listening costs as much as sending, while sleeping and local computation are free, but the underlying graph is a clique. When nodes choose to transmit they are broadcasting their message to the entire graph, or at least anyone who is listening, and the main issue is just how to resolve contention for the channel. Moreover, in much of this work, there is no bound on message sizes. In this model, Nakano and Olariu [28] gave an $O(\log \log n)$ energy algorithm for selecting distinct IDs in $\{1, \ldots, |V| = n\}$. Bender, Kopelowitz, Pettie

and Young [3] showed that $O(\log(\log^* n))$ energy suffices for all devices to send messages if there is collision-detection available on the channel. Chang, Kopelowitz, Pettie, Wang and Zhan [5] showed this to be optimal, and studied the problems of Leader Election and Approximate Counting, both with and without randomization, with collision detection. They also studied tradeoffs between time, energy and error probability. Similar tradeoffs were also studied by Kardas et al. [21]. Jurdzinski, Kutylowski and Zatopianski [16, 15, 17, 18, 19]. studied the Leader Election and Approximate Counting problems in the absence of collision detection.

The single-hop notion of energy complexity was extended to arbitrary networks by Chang, Dani, Hayes, He, Li, and Pettie [7] where they studied the Broadcast problem, with and without collision detection, and with and without randomization. The energy complexity was shown to be $\text{polylog}(n)$ in all cases. However, their broadcast algorithm did not transmit messages along shortest paths. Later Chang, Dani, Hayes and Pettie [8] gave a $2^{O(\sqrt{\log n})}$ algorithm for BFS. They also showed that the diameter of a network is hard to approximate to within factor 2 using sublinear energy.

Other models of energy cost have also been studied in the radio network literature. Gasnieniec et al. [12], Berenbrink et al. [4] and Klonowski and Pajak [23], studied broadcast and gossiping problems under a cost model where the goal is to minimize the worst case number of transmissions per device. Klonowski and Sulkowska [24] defined a distributed model in which devices can transmit messages at varying power levels, which can be chosen online. The devices in question are at random locations in the d-dimensional cube. A number of works [25, 20, 13, 22] also considered robustness of low-energy algorithms against a jamming attack. Here an adversary attempts to foil the devices' attempts to communicate by making noise on the channel. The goal here is not precisely low-energy use, but rather "resource competitive" energy use, where the processors combined cost for getting messages through should be commensurate with the adversary's budget for wreaking havoc.

A related model in distributed computing on wired networks is the Sleeping Model of Chatterjee *et al.* [9]. In their model, which is a variant of CONGEST, nodes can also save energy by going to sleep, in which case they may miss incoming messages. However, their model is considerably more powerful than its radio network analog, in that they allow a node to send or receive distinct messages from each of its awake neighbors in a single round.

Organization of the Paper

Section 2 contains a more detailed introduction to the radio network model. Section 3 discusses our method of cluster formation, and related issues. In Section 4, we discuss the problem of simulating radio network algorithms, and give a formal definition of what we mean by "optimal" simulation. In Section 5, we describe our simulation algorithm, and give formal statements of our main results. In Section 6, we apply our simulation algorithm to the breadth-first search problem, showing it can be solved in polylog energy. Due to space considerations, all proofs have been relegated to the Appendix. A longer version of this paper is available at arXiv:2205.12830[cs.DC].

2   Preliminaries: the Low Energy Radio Network Model

The Network

We assume there is a communication network on an arbitrary, undirected connected graph $G = (V, E)$. At each node in G, there is a processor equipped with a transmitter and receiver to communicate with other nodes. There is an edge between nodes u and v in the graph if u and v are within transmission range of each other.

The processors are identical, except for having unique IDs. They do not know the underlying graph G; indeed we assume that initially they do not even know their neighbors in the graph. A processor will become aware of (and remember) any neighbor once it has communicated with it. The processors do have a shared estimate on the size of the graph; that is, a number $n \geq |V|$ is known to all the processors and may be used in any algorithms they run. Accuracy of this estimate is not required for correctness of the algorithm, but the time and energy usage of algorithms will depend on it. Additionally we assume that the processors can generate independent streams of random bits. There is no shared randomness. **For randomized algorithms, we assume that each node does all of its coin flipping prior to the beginning of the algorithm, generating a string of random bits to be read off and used at the appropriate time in the algorithm.** This is a standard construct for viewing a randomized algorithm as a deterministic algorithm with an additional random input.

Time

Time is divided into discrete, synchronous timesteps, and the processors agree on a time $t = 0$. In each timestep a processor can choose to do one of three actions: transmit, listen, or sleep. When a processor decides to transmit at time t, it sends a message of size $O(\log n)$ bits, whereupon it is potentially heard by any of its neighbors who happen to be listening at time t. We say "potentially" because there are a number of different models for what happens if a listening node encounters a collision, *i.e.* two or more of its neighbors are broadcasting messages in the timestep that it is listening. Nevertheless, at a minimum, we can say that a message travels from a node u to a neighbor v of u at time t if

- u decides to transmit at time t,
- v decides to listen at time t and
- no other neighbor of v decides to transmit at time t.

Collisions and Message Delivery

There are several different models for how to handle collisions, that is, what information does a listening node receive when two or more of its neighbors are sending messages in a single round? In the most permissive of these, the **Broadcast CONGEST model**, v receives all the messages sent by its neighbors, as if they were being sent on separate channels.

A more restrictive model is the **Collision Detection model (CD)** where, when a listener is next to more than one sender, a special "noise" message is received, which is distinguishable from silence (no neighbors sending), but carries no further information.

Another model of interest is the **No Collision Detection model (no-CD)**, which is even more restrictive: here, collisions are indistinguishable from silence.

Exponential backoff, introduced in the context of radio networks by Bar-Yehuda, Goldreich and Itai [1], is a commonly used technique for handling collisions. Among other uses, it can be used to eliminate the problem of collisions; see, for example, [6, 7, 8]. The basic idea is to simulate each timestep using $O(\log^2 n)$ timesteps, divided into $O(\log n)$ rounds of $O(\log n)$ timesteps each. A listener listens for the entire interval of length $O(\log^2 n)$ or until a message is received. In each round, senders flip coins at each timestep to decide whether to (re-)transmit, or to retire for the rest of the round, resulting in a constant probability that, in a particular round, there is at least one timestep in which a unique neighbor of the listener is sending, and therefore a message is successfully received, regardless of the collision model. The net result of this $O(\log^2 n)$-time backoff procedure is that, with probability $1 - 1/\text{poly}(n)$, every listener successfully receives a message, assuming at least one neighbor is a sender.

The fact that collisions can be handled in this manner inspires the definition of the following alternative message delivery model without collisions. In the "OR" model of message delivery, every time a node listens, it receives an arbitrary message sent by one of its neighbors in that timestep. The only exception is when no neighbor of the listening node chose to send in that timestep, in which case no message is received. For convenience, we will assume for the rest of this paper that we are working in the OR model of message delivery.

Energy Usage

We measure the cost of our algorithms in terms of their energy usage. We assume that a node incurs a cost of 1 energy unit each time that it decides to send or listen. When the node is sleeping there is no energy cost. We also assume that local computation is free. The goal of energy-aware computation is to design algorithms where the nodes can schedule sleep and communication times so that the per-node energy expenditure is small, ideally $\text{polylog}(n)$, without compromising the time complexity too much, i.e., the running time remains polynomial in n. In fact, our simulation algorithm does better: its time complexity is only a $\text{polylog}(n)$ factor greater than that of the simulated algorithm.

3   Cluster Graphs and Distance Approximation

In this section we introduce our framework for getting distance estimates that will be used by the nodes in the simulation algorithm.

3.1   Graph theoretic preliminaries

Let $G = (V, E)$ be a graph. Let $d : G \times G \to \mathbb{N}$ be the shortest path metric on G. Consider a partition $\mathcal{P} = (V_1, V_2, \dots, V_k)$ of V into pairwise disjoint sets of vertices. We will call each V_i a "cluster". The *cluster graph*, also called the *quotient graph* and denoted $G/\mathcal{P}$, is a graph whose vertex set is the set of clusters, $\{V_1, \dots V_k\}$. There is an edge in the cluster graph between V_i and V_j if there are vertices $u \in V_i$ and $v \in V_j$ such that $(u, v) \in E$.

Although the definition of a quotient graph does not require either G or the subgraphs induced by the clusters V_i to be connected, in our work we will assume that both are the case. It is easy to see that since G is connected, so is $G/\mathcal{P}$.

Given a partition $\mathcal{P}$, we will denote the cluster containing vertex u by $[u]$. Since the cluster graph is a graph, we can also define the shortest path distance metric on the cluster graph. We are interested in partitions where the individual clusters have comparable diameters, and the distances between clusters in the cluster graph are (approximately) scaled versions of the distances in the underlying graph.

3.2   Approximately Distance-Preserving Partitions

▶ **Definition 2.** *Let $G = (V, E)$ be a graph, and let $R, \alpha, \beta \geq 1$. Let $\mathcal{P}$ be a partition of V. Let d denote the distance metric in G and d^* denote the distance in $G/\mathcal{P}$. We say $\mathcal{P}$ is an (R, α, β)-approximately distance-preserving partition if*

- *for every pair of nodes u and v with $d(u, v) \leq R$, we have $d^*([u], [v]) \leq \alpha$*
- *for every pair of nodes u and v with $d^*([u], [v]) = 0$, (that is, u and v are in the same cluster) we have $d(u, v) \leq \beta R$.*

Taken together, the two halves of the above definition ensure that up to a multiplicative factor in the range $[1/\alpha, \beta]$, as well as possible rounding issues, we have, for all $u, v \in V$, that $d^*([u], [v]) \approx \frac{d(u,v)}{R}$. Specifically,

▶ **Lemma 3.** *Let $G = (V, E)$ be a graph, and $\mathcal{P}$ be an (R, α, β)-approximately distance-preserving partition of V. Let d and d^* denote the shortest path distance metrics in G and $G/\mathcal{P}$ respectively. Then, for all $u, v \in V$,*

$$\left\lfloor \frac{d(u, v)}{\beta R + 1} \right\rfloor \le d^*([u], [v]) \le \alpha \left\lceil \frac{d(u, v)}{R} \right\rceil.$$

A nice example of an approximately distance-preserving partition is for the square grid. Let G be the $n \times n$ square grid graph, and suppose R is a divisor of n. We partition V by rounding each point (x, y) down to the nearest multiple of R, $\left(R\lfloor \frac{x}{R} \rfloor, R\lfloor \frac{y}{R} \rfloor\right)$, and placing two vertices in the same cluster if they round to the same multiple of R. It is easy to see that this example is an $(R, 2, 2)$-approximately distance-preserving partition. Quite similar constructions can be done for all real $R \ge 1$, and for many other "homogeneous" graphs, such as lattice graphs of fixed dimension.

Although it may not be immediately obvious, such approximately distance-preserving partitions exist for all graphs, for all $R \ge 1$, and for $\alpha, \beta = O(\log n)$, as we shall see next.

3.3 Additive Weights Voronoi Diagrams and the MPX Algorithm

Additively Weighted Voronoi Decomposition (AWVD) (see, e.g., Phillips [29]), also known as hyperbolic Dirichlet tesselation, is a well-studied concept for real Euclidean domains. Start with a finite set of points in the plane, called generators, each of which is assigned a real-valued weight. Each point x in the plane is assigned to the generator g minimizing the sum $\|x - g\| - W(g)$, where $W(g)$ is the weight of g. After discarding any empty cells, this defines a partition of the plane into a finite collection of cells. When the weights are all zero, this corresponds to the usual notion of Voronoi diagram, and the boundaries of the cells are line segments and rays. For general weights, the boundaries of the cells are hyperbolic arcs. The cells are star-shaped with respect to their generators, but are generally not all convex.

For a finite graph, $G = (V, E)$, the analogous concept is a partition of V based on assigning a real-valued weight $W(v)$ to each vertex $v \in V$. We say that vertex u belongs to the cell generated by vertex v if $v = \arg\min_{v' \in V} d(v', w) - W(v')$. For convenience, we will assume that no two weights $W(v), W(v')$ differ by an integer, so that the cells are defined unambiguously. Two vertices belonging to the same cell is an equivalence relation, so each AWVD gives rise to a corresponding cluster graph.

Miller, Peng, and Xu [26] proposed a simple randomized graph-partitioning algorithm to obtain a decomposition with certain properties, specifically that the clusters have small diameter and only a small fraction of the edges of the graph are cut. In their construction, starting with a (common) parameter R, each vertex v independently samples a random variable $\delta_v \sim \text{Exponential}(1/R)$ from the exponential distribution with mean R. A cluster starts forming at each vertex v at time $-\delta_v$, and spreading through the graph at a uniform rate of one edge per time unit. Each vertex u either joins the first cluster to reach it before time $-\delta_u$ or starts its own cluster at time $-\delta_u$ if no other cluster has recruited it before that time. Haeupler and Wajc [14] showed that with minor modifications, this algorithm can be efficiently implemented in the Radio Network model.

We note that the MPX decomposition is, in fact, an AWVD where the weights $W(v)$ are independent exponentially-distributed random variables with mean R. We will now see that this decomposition has some good distance approximating properties.

As shown in [26], the clusters have diameter $O(R \log n)$. We state this more precisely:

▶ **Lemma 4.** *With probability at least $1 - \dfrac{1}{n^2}$, the clusters in the MPX decomposition with parameter R have diameter at most $3R \log n$.*

Furthermore, Chang *et. al* [8] showed that not too many clusters are close to a single vertex. Specifically, if $\mathcal{P}$ is the partition determined by the MPX algorithm with parameter R, and $G^* = G/\mathcal{P}$ is the corresponding cluster graph, then Lemma 2.1 from [8] (translated into our notation) can be stated as follows:

▶ **Lemma 5.** *For every positive integer j and $\ell > 0$, the probability that the number of G^*-clusters intersecting $\mathrm{Ball}_G(v, \ell)$ is more than j is at most $(1 - \exp(-2\ell/R))^j$.*

▶ **Corollary 6.** *For any $v \in V$, $\mathbf{P}\left(At\ most\ 20 \log n\ clusters\ intersect\ \mathrm{Ball}_G(v, \ell)\right) \geq 1 - \frac{1}{n^2}$.*

Combining Lemma 4 and Corollary 6, we have shown that

▶ **Proposition 7.** *With probability at least $1 - \frac{2}{n^2}$, the partition $\mathcal{P}$ produced by the MPX algorithm with parameter R is a $(R, 20 \log n, 3 \log n)$-approximately distance-preserving partition*

3.4 Multi-Scale Clustering

▶ **Definition 8.** *Suppose, for $1 \leq i \leq \ell$, we have an approximately distance-preserving partition $\mathcal{P}_i$ with parameters (R_i, α_i, β_i), where $R_1 < \cdots < R_\ell$. By convention, we extend this definition to the case $i = 0$ by setting $R_0 = \alpha_0 = \beta_0 = 1$, and letting the $i = 0$ partition be the partition of V into singleton vertex sets. Suppose further that for $1 \leq i \leq \ell$, we have*

$$\frac{R_{j+1}}{R_j} \geq (2\alpha_j + 1)(\beta_j + 1) - 1. \tag{1}$$

We call this a multi-scale clustering with parameters $(\mathbf{R}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = ((R_i), (\alpha_i), (\beta_i))_{i \in \{0, \ldots \ell\}}$.

Notice that there is no requirement that the lower-level clusters be refinements of the higher-level clusters. Despite this, as we will see, multi-scale clusterings have a useful, albeit weaker, nesting property that controls the relationship between clusters at consecutive scales.

Suppose $\mathcal{P}_1, \mathcal{P}_2, \ldots \mathcal{P}_\ell$ is a multi-scale clustering on G, with parameters $(\mathbf{R}, \boldsymbol{\alpha}, \boldsymbol{\beta})$. Let $G_1, \ldots G_\ell$ be the corresponding cluster graphs, and $d_1, \ldots, d_\ell$ the corresponding distance metrics. For $v \in V$, let $[v]_j$ denote the cluster containing v in $\mathcal{P}_j$. As usual, $\mathrm{Ball}_{G_j}([v]_j, r)$ denotes the ball of radius r in G_j. We will also need a notation for the vertices in the underlying graph G, whose clusters belong to this ball. To this end we define

$$B_j(v, r) = \{w \in V \mid d_j([w]_j, [v]_j) \leq r\}.$$

The following nesting property is true for all vertices, at all scales.

▶ **Lemma 9.** *Let $1 \leq j < \ell$, $u \in V$, and $v \in \mathrm{Ball}_G(u, R_{j+1} - R_j)$. That is, $d(u, v) \leq R_{j+1} - R_j$. Then $B_j(v, 2\alpha_j) \subset B_{j+1}(u, 2\alpha_{j+1})$.*

Although the only restriction imposed so far on the parameters $(\mathbf{R}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ is that the R_js grow at a certain rate relative to the α_js and β_js, an interesting and natural special case is when (R_j) is an increasing geometric sequence, while sequences (α_j) and (β_j) are approximately constant.

▶ **Definition 10.** *In the special case when, for $1 \leq j \leq \ell$, $R_j = R^j$, for some $R > 12$, and $\alpha_j = \beta_j = \lfloor \sqrt{R}/2 \rfloor$, we call this a geometric multi-scale clustering with parameter R.*

We saw earlier that the MPX clustering with parameter R is, *w.h.p,* a $(R, 20 \log n, 3 \log n)$-approximate distance-preserving partition. Naturally, it is also a $(R, 20 \log n, 12 \log n)$ approximate distance-preserving partition. Setting $\alpha = \beta = 20 \log n$ and $R = \Theta(\log^2 n)$ we can get a geometric multi-scale clustering by constructing MPX partitions for all scales $R, R^2, R^3 \ldots$

3.5 Simulating cluster-graph algorithms on the underlying graph

Chang et al. [7] defined protocols called UPCAST, DOWNCAST, and INTERCAST for communicating within and between clusters of nodes in a Radio Network. In the corresponding section of the Appendix, we describe these protocols at a high level, and remind the reader of some of their relevant properties.

4 Simulating Radio Network Algorithms

Let A be a radio network algorithm. When talking about the time and message complexity of A, since one does not charge for listening, one usually only specifies which timesteps A requires a node to SEND, with the implicit assumption that whenever a node is not SENDing it is LISTENing. Given any algorithm in this form, it is trivial to naively convert it to what we view here as the standard form, by making all LISTENs explicit. After such a naive conversion, the algorithm will have per-node energy cost equal to its time complexity, which is not good, unless the time complexity is already very small. Our plan for improved energy efficiency is now to simulate this explicitly wasteful algorithm by one in which many of the LISTENs have been replaced by SLEEPs without compromising the correctness of the algorithm, and with a relatively small overhead in time complexity.

Whenever we talk about simulating a radio network algorithm in this paper, we mean replacing it with another equivalent algorithm which is moreover *safe black-box* and *synchronized one-pass*, terms which we are about to define. By equivalent, we mean that at the end of the simulation, each node, v, in the network will have computed its view of a correct transcript of the original algorithm; that is, a record of all messages v sent or received at each timestep in the original algorithm, as well as the correct final internal state. By *black-box*, we mean that when the simulating protocol wants to perform a step of the simulated algorithm, it does so by invoking an oracle that tells it how to update its state, whether to SEND, LISTEN, or SLEEP, and what messages to send. By *safe*, we mean that, the simulating protocol never causes a node to SLEEP during a round in which the original protocol would SEND or receive a message from a SENDing neighbor, regardless of what oracle is provided. Specifically, we don't just mean that the simulating algorithm performs correctly when simulating A; it must avoid SLEEPing incorrectly for all possible simulated algorithms. Finally, by *synchronized one-pass*, we mean that, at certain designated timesteps $\tau(t)$, all nodes either SLEEP or simulate step t of the simulated algorithm. Each timestep is simulated only once, and the timesteps $\tau(t)$ are a strictly increasing function of t. It so happens that for our particular simulation algorithm, the timesteps $\tau(t)$ are a fixed function of t, known in advance, rather than determined adaptively, but this will not be important for our analysis.

4.1 Characterizing Optimal Simulation

In order to come closer to getting our hands on what it would mean for a safe simulation algorithm to be optimal or nearly optimal, we introduce the following generous abstract model for what it needs to do.

Our notion of a generic algorithm simulator is a radio network algorithm, where, at some timesteps, it invokes a black-box simulation of one timestep of the simulated algorithm, A, and at other timesteps, it does its own thing, which may involve metadata gathered from the usage pattern of A. Since A is arbitrary and unknown, we cannot count on knowing the meaning of the messages sent by A, but it may, nevertheless, be useful to know which nodes of A are SENDing, LISTENing, or SLEEPing at a given timestep.

With the above in mind, we define the Generic Simulation Model as follows. For each timestep, at each node, our algorithm must choose to either SEND, LISTEN, SLEEP, or SIMULATE. Steps when our algorithm SENDs, LISTENs or SLEEPS work the same way as in the radio network model. On a SIMULATE step, a black-box for the simulated algorithm, A, simulates one timestep, which may involve SENDING, LISTENING, or SLEEPing. Each node v is given the following information:

1. whether v chose to SEND, LISTEN, or SLEEP in the simulated step.
2. whether any message was received by v in the simulated step.
3. the identity of the next timestep, t_v, at which A would make v SEND, under the assumption that no further messages were received by v before t_v.

At the beginning of the algorithm, we further suppose that the simulator at each node v is given to know the first timestep t_v, at which A would make v SEND, under the assumption that no messages were received by v before t_v. We note that, if the simulated algorithm A were fully event-driven and deterministic, the "unprovoked next-send times" t_v would all be $+\infty$, except for those nodes that send in the first timestep. However, for general algorithms, and in particular, for randomized algorithms, such as the MPX clustering algorithm, there may be many times at which new sequences of SEND steps start without being directly preceded by an incoming message. For randomized algorithms, since we have assumed that each node does all of its coin flipping prior to the beginning of the algorithm, note that the times t_v can be accurately predicted in advance. Since the definition of t_v assumes that no new external information reaches node v between the current step and timestep t_v, the information needed for this calculation is always available to the simulator at the current timestep. Here, we are taking full advantage of the standard assumption that local computation is free; however, we also believe that in most settings, calculating t_v at time t can be done much more quickly than the ultra-naive approach of doing $t_v - t$ steps of black-box simulation of A.

Next, we define a Psychic Synchronized Generic Simulation Model, as follows. Everything about it is the same as for the Generic Simulation Model, above, except that, in addition to being informed about messages received by v in the preceding time step, in this model, we assume that the simulator starts out knowing the entire graph, including node IDs, and is informed, at each time step, of the entire history of metadata at all nodes, including the information about times at which the current node was asleep. In particular, this includes knowing exactly which nodes SENT, LISTENed, and SLEPT at all times $\leq t$, as well as all timesteps $t_{\mathrm{next}}(w, t)$ at which future messages are scheduled to be sent in the absence of provocation, for all vertices $w \in V$.

The other crucial assumption we make in the PSGSM is that all nodes advance their simulation of A in a synchronized way. That is, every node simulates the behavior of A at time t at the same time. This assumption seems natural, considering that the success

or failure of the message deliveries depends on whether collisions occur, but it is still an assumption. Now, considering that in the PSGSM model, all nodes receive, for free, the entire history of relevant metadata for the entire graph, at each timestep they are awake, there seems to be no point in further communication between nodes, except for when steps of A are being simulated. Thus, we find that, in the PSGSM model, the simulated algorithm A should always run in real time.

Now, in both of the above models, we have at least one expectation of a simulation algorithm, and that is *correctness*: at the end of the simulation, we want every node to have its local view of the transcript of the actual computation done by A from the given inputs. In the case of randomized algorithms, we view each node's pre-flipped coins as part of its input, which allows us to reduce to the case of a deterministic algorithm in the usual way.

Our motivation in introducing the Psychic Synchronized (PSGS) model is that, in this model, we can precisely characterize the optimal energy usage of any correct simulation in terms of the behavior of a simple greedy algorithm, Algorithm 1.

■ **Algorithm 1** The Greedy Psychic Algorithm: conserve energy while simulating a given Radio Network agorithm in the PSGSM model.

---

procedure GP(A)   ▷ A: the simulated algorithm
  $t^* \leftarrow 1$
  for $t \leftarrow 1$ to T **do**
      if $t < t^*$ **then**
          SLEEP
      else
          SIMULATE timestep t of A
          $t^* \leftarrow t_{\text{next}}(v, t)$ ▷ our next scheduled SEND
          for every vertex $w \neq v$ **do**
              psychically receive $t_{\text{next}}(w, t)$ ▷ time of w's next scheduled SEND
              $t^* \leftarrow \min\{t^*, t_{\text{next}}(w, t) + (\text{dist}(v, w) - 1)\}$
                                                                                          ▷ SLEEP until time t^*

▶ **Theorem 11.** *This "Greedy Psychic" algorithm is optimal among all correct simulation algorithms in the PSGSM model, in the sense that, for every simulation algorithm SIM in this model, either there exists a radio network algorithm A and an input x such that $SIM(A, x)$ makes a mistake (some node does not end with the correct transcript), or, for every algorithm A, input x, and vertex v, we have $\text{ENERGY}(GP, A, x, v) \leq \text{ENERGY}(SIM, A, x, v)$.*

The main consequence of Theorem 11 will be that Algorithm SAF is within a polylog(n) multiplicative factor of optimal, at least among algorithms that perform a single synchronized simulation of the simulated algorithm A. This near-optimality holds in a vertex-by-vertex and algorithm-by-algorithm manner. We point out that without the assumption of synchronicity, we cannot expect to get guarantees of this kind. For example, if we want to make an asynchronous algorithm that minimizes the listening cost for a specific vertex v, we can make v wake up very infrequently, and ensure that its neighbors are always "holding v's messages ready for it." In this way, the energy complexity for v can be reduced essentially to the number of messages that v needs to send or receive; this favoritism towards v would presumably be more than paid for by increased costs incurred at other vertices. This strongly suggests that, without the assumption of synchronous one-pass simulation, there may not exist a single simulation algorithm that simultaneously minimizes the energy costs for all nodes.

5   The Simulation Algorithm

In this section we describe our recursive simulation algorithm. We begin with a section describing the underlying assumptions, and then give a high-level overview of the algorithm. The actual pseudocode appears in Section 5.3. We analyze the the algorithm and its energy complexity in Section 5.4.

5.1   Assumptions

We assume that we are given a function f describing protocol to be run on G, that is guaranteed to succeed in the OR model of message delivery. Recall that, as mentioned in Section 2, we may assume f is deterministic without loss of generality, since each node may do any necessary coin flipping in advance, at the beginning of the algorithm. Note that, since nodes are not privy to each others inputs, this assumption does not imply that randomness is shared. In particular, if a node v were to randomly choose a timestep at which to SEND a message, the other nodes would not automatically know which timestep it was; however, v would know in advance, and the Simulation algorithm would have v warn its neighbors.

   We also assume that, prior to attempting to execute our algorithms indexed by j, at least levels $1, 2, \ldots, j$ of a (R, α, β) multi-scale clustering have been computed. Each node knows the ID of its cluster center, as well as its graphical distance to the cluster center. (This information is used in UPCAST/DOWNCAST.)

   Our algorithms all have the property that each call to one of the defining functions takes the same number of timesteps, including those spent in recursive calls. In some cases, this number of timesteps can be tedious to compute; therefore, we have taken the liberty to refer to the number of "simulated" timesteps, namely the number of timesteps of the simulated algorithm. This should be understood as a shorthand only. Here is a complete list of the state information that each node must store in its memory.

- Shared knowledge of the graph parameters n, Δ, D. Here, n is an upper bound on the number of nodes in G, Δ is an upper bound on the maximum degree of G, D is an upper bound on the diameter of D. If Δ and D are not specified, $n - 1$ can be used in their place, (although better bounds lead to better performance). We require that all nodes in the network start out initialized with the same values of these parameters.
- The current timestep, t. Since we are assuming a synchronous network, this value is always the same for all nodes.
- The cluster parameters R_j, α_j, β_j, for $1 \leq j \leq \ell$. This knowledge is also shared by all nodes in the network.
- A unique ID for our node. If these are not specified, a random string of $C \log n$ bits may be used; by the birthday paradox, these are distinct with high probability.
- For $1 \leq j \leq \ell$, the ID of our level-j cluster center, as well as our graphical distance to the cluster center. This is used for energy-efficient communication by the UPCAST/DOWN-CAST subroutines (see [7]).
- A message string, m. Initially null for all nodes. This is used in the NOTIFY protocol.
- Any state information stored by our node in the simulated protocol, f.

5.2   Algorithm Overview

At a high level, the SAF simulation algorithm gets an algorithm f and a time interval I, and has the goal of simulating a run of f on I while allowing processors that are far from the action in f to save energy by sleeping until the appropriate time. To this end, SAF
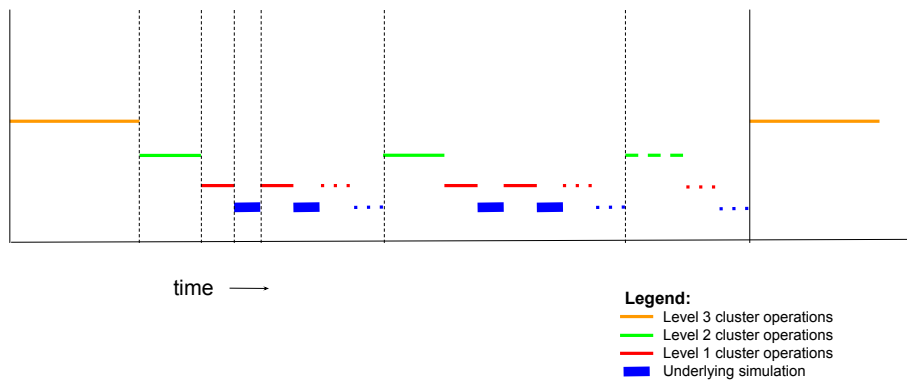
Figure 1 Timeline of how cluster operations at various levels interleave with the underlying simulated algorithm. Each node starts at the top level (here $\ell = 3$) and participates in cluster operations until it finds a level at which it can drop out. That level also determines when it will wake up next: the next round of cluster operations at that level. If the node gets to the bottom level of cluster operations without dropping out, then it stays awake for the underlying simulation for the next few simulated timesteps.

simulation uses a multi-scale clustering to estimate the distance to the nodes where the communication is happening, and to set SLEEP and WAKE schedules for the nodes that need to wait. The main idea is that the clusters at level j, which are (R_j, α_j, β_j) approximately distance-preserving, are equipped to make a decision (SLEEP/WAKE) from their cluster that is pertinent to the next R_j steps of the simulated algorithm. This decision is made for each cluster, collectively by its members, by performing internal cluster operations to decide whether there is activity nearby. However, since their confidence in the prediction of inactivity is only good for R_j simulated timesteps, this would appear to mean that they must wake up and perform cluster operations after every R_j steps of the simulated algorithm. When R_j is small, this would not actually help them save much energy. However, this is where the clusterings at multiple scales come in. At the highest scale, ℓ, the clusters wake up every R_ℓ steps, so as long as the total simulated time that f needs to run is at most $R_\ell \mathrm{polylog} n$, the algorithm is in good shape. The notion of what it means for the action to be far away has been tuned so that nesting property described in Lemma 9 ensures that if a top level cluster has gone to sleep, then all the lower level clusters that are contained within can also afford to sleep for the full time horizon of R_ℓ steps. If a top level cluster decides to stay awake, then it sets up a recursive call to enable affected lower level clusters to decide how many times they must wake up with the shortened time horizon. Note that the top level cluster staying awake does not mean that an individual node within that cluster must sat awake for R_ℓ timesteps. Indeed, that would expend far too much energy. Rather, it means that such a node must participate in lower level cluster operations, until it can find a scale at which the computational activity of f is far away from it. Only if a node finds itself to be awake at *all* scales will it participate in the actual simulation of f, for the next interval of the smallest scale length, R_1.

But how to get the multi-scale clustering to begin with? Herein lies the beauty of the process. At the bottom level, an (R_1, α_1, β_1) approximate distance-preserving partition can be computed with $\text{polylog}(n)$ energy and $\text{polylog}(n)$ latency, with all processors staying awake the whole time. (We need R_1 to be $\text{polylog}(n)$.) Thereafter, the simulation has a multi-scale clustering to work with, with $\ell = 1$ the first time around; and the partitioning algorithm is itself a function that is suitable for being simulated recursively. Thus, each subsequent level of the final multi-scale clustering is bootstrapped off the previous ones.

5.3   Pseudocode

Our code is divided into several pieces, which are presented as Algorithms 2 through 5. The SAF Simulation algorithm is stated in a form where it can be applied using an arbitrary multi-scale clustering. In general, we want to apply it to the multi-scale clustering we know how to construct for general graphs, namely the ESTCluster algorithm of Miller, Peng, and Xu. To build this clustering efficiently, we simulate it using the SAF algorithm inductively. Finally, to efficiently run BFS from an arbitrary start node, we apply SAF Simulation to the naive parallel BFS algorithm that has each node listen until the BFS frontier reaches it, then send a message to its neighbors to advance the frontier while incrementing its level counter.

■ **Algorithm 2** SAF Simulation Algorithm: Adaptively power down receiver to save energy while simulating a given algorithm, f. Assumption: we have a hierarchy of α, β, R.

---

procedure $\text{SAF}(j, I, f)$ ▷ j: current cluster height, I: interval of simulated times, f: update rule for the simulated algorithm (on level 0 clusters)

> **if** $j = 0$ **then**   ▷ Bottom level (single node)
>> (Naively) execute f for $|I|$ timesteps.
>
> **else**
>> Partition I into disjoint subintervals, $I_1, I_2, \ldots, I_k$, each of length R_j.
>>
>> **for** $i \leftarrow 1$ to k **do**
>>> Internally simulate f on I_i assuming no messages received.
>>>
>>> $m \leftarrow \begin{cases} \text{null} & \text{if we never sent during the simulation.} \\ \text{"Activity nearby!"} & \text{if we sent anything during the simulation.} \end{cases}$
>>>
>>> **if** $\text{NOTIFY}(j, m, 2\alpha_j)$ returns "OK TO SLEEP" **then**
>>>> SLEEP until end of last simulated timestep in I_i.
>>>>
>>>> Update state based on the above simulation.
>>>
>>> **else**
>>>> $\text{SAF}(j - 1, I_i, f)$

5.4   Formal Statements of the Results

▶ **Definition 12.** *Suppose P is an ℓ-level multiscale clustering with parameters $(\mathbf{R}, \boldsymbol{\alpha}, \boldsymbol{\beta})$. Then, for $0 \leq j \leq \ell$, we define a* level-j epoch *to be any time interval of the form*

$$\{iR_j + 1, iR_j + 2, \ldots, (i + 1)R_j\}.$$

We will use the following result relating distances from the active set to the event of being awake and made to participate in the NOTIFY subroutine.

■ **Algorithm 3** Ensure that all nearby clusters (of a given height) wake up when something interesting is happening nearby.

---

procedure NOTIFY(j, m, r)   ▷ j: current cluster height, m: message to send, r: transmission radius ▷ If $m = $ null, do not initiate sending, but do repeat any message you hear.

    for $i \leftarrow 1$ to r **do**
        UPCAST(j,m)
        DOWNCAST(j,m)
        INTERCAST(j,m)
    UPCAST(j,m)
    DOWNCAST(j,m)
    if we received or sent any messages during this call to NOTIFY **then**
        return "STAY AWAKE"
    else
        return "OK TO SLEEP"

■ **Algorithm 4** Naively build MPX clusters with radius parameter R.

---

procedure NAIVELY-BUILD-MPX(R)   ▷ R: radius parameter

    `my_cluster_center`$[j] \leftarrow$ `null`
    Sample a weight W from the exponential distribution with mean R.
    $t_{\max} \leftarrow \lceil 3R \log(n) \rceil$
    for $i \leftarrow 1$ to $t_{\max}$ **do**
        if `my_cluster_center`$[j] \neq$ `null` **then**
            SEND message (`my_cluster_center`$[j]$,`my_cluster_depth`$[j]$).
            Break out of FOR loop, and SLEEP for remaining $t_{\max} - i$ timesteps.
        else if $i + W \geq t_{\max}$ **then**
            `my_cluster_center`$[j] \leftarrow$ `my_ID`
            `my_cluster_depth`$[j] \leftarrow 0$
        else
            LISTEN this timestep.
            if message (c, d) received **then**
                `my_cluster_center[j]` $\leftarrow c$
                `my_cluster_depth`$[j] \leftarrow d + 1$

■ **Algorithm 5** Efficiently build a geometric multiscale clustering with parameter $O(\log^2 n)$.

---

procedure BUILD-MSC

    $R \leftarrow C \log^2(n)$
    for $j \leftarrow 1$ to $\log_R(n)$ **do**
        SAF($j - 1$, $[0, \lceil 3R^j \log(n) \rceil]$, Naively-Build-MPX($R^j$))

▶ **Lemma 13.** *For $j \geq 1$, $t = (i+1)R_j$, $v \in V$, if the distance from v to the nearest node that would send in the epoch ending at time t is at least $(2\alpha_j + 1)(\beta_j R_j + 1)$, then, v will not participate in any calls to NOTIFY(j − 1) within this epoch.*

Now we are ready to state our main result about our simulation algorithm.

▶ **Theorem 14.** *Let f be any randomized radio network protocol in the OR model. Suppose P is an ℓ-level multiscale clustering with parameters $(\mathbf{R}, \boldsymbol{\alpha}, \boldsymbol{\beta})$. Then $SAF(\ell, I, f)$ has the following properties:*
- $\mathbf{P}\left(SAF(\ell, I, f) \text{ succeeds}\right) \geq \mathbf{P}\left(f \text{ succeeds}\right) - 1/\mathrm{poly}(n)$.
- *The running time of $SAF(\ell, I, f)$ is $T_f \, \mathrm{polylog}(n)$, where T_f is the running time of f.*
- *The energy cost of $SAF(\ell, I, f)$ for a vertex v is at most $\mathcal{E} \, \mathrm{polylog}(n) + T \log n / R^\ell$, where $\mathcal{E}$ is the energy cost of the greedy psychic algorithm $GP(f)$ for vertex v, and $T = \mathrm{TIME}(f)$ is the time complexity of the simulated algorithm.*

▶ **Theorem 15.** *The Build-MSC algorithm (Algorithm 5) builds a multi-scale clustering at all scales, with probability $1 - 1/\mathrm{poly}(n)$, with total running time $O(D \, \mathrm{polylog}(n))$ and per-node energy usage $O(\mathrm{polylog}(n))$.*

6 BFS Revisited

In this section we apply our methodology to get a polylog energy algorithm for Breadth First Search in radio networks, thus answering an open question from [8]

The algorithm is very simple: we simply simulate the naive BFS algorithm for radio networks in our SAF simulation framework.

◼ **Algorithm 6** Solve BFS from a designated vertex, v in low energy. D is an upper bound on the diameter.

---
procedure EFFICIENT-BFS(v)
    BUILD-MSC
    SAF(ℓ, $[0, D]$, Naive-Parallel-BFS(R^j))

▶ **Theorem 16.** *The Efficient-BFS algorithm (Algorithm 6) computes the graphical distance to each node from the root vertex, with probability $1 - 1/\mathrm{poly}(n)$. Its total running time is $O(D \, polylog(n))$ and per-node energy usage is $O(\mathrm{polylog}(n))$.*

7 Conclusion

We have shown a new general-purpose methodology for reducing the energy cost of Radio Network algorithms by collaborating with clusters of nearby nodes at multiple scales to detect when it is safe to shut off the receiver due to there being no danger of message activity nearby. Although similar techniques have been used in previous work, the precise way in which we create and use our clusters leads, at least in some cases, to significantly improved results. In particular, our methodology allows us to easily, and at least in the case of BFS, significantly improve over known results.

──── **References** ────

1 R. Bar-Yehuda, O. Goldreich, and A. Itai. On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization. *Journal of Computer and System Sciences*, 45(1):104–126, 1992.

2 M. Barnes, C. Conway, J. Mathews, and D. K. Arvind. ENS: An energy harvesting wireless sensor network platform. In *Proceedings of the 5th International Conference on Systems and Networks Communications (ICSNC)*, pages 83–87, 2010.

3 M. Bender, T. Kopelowitz, S. Pettie, and M. Young. Contention resolution with constant throughput and log-logstar channel accesses. *SIAM J. Comput.*, 47:1735–1754, 2018.

4 P. Berenbrink, C. Cooper, and Z. Hu. Energy efficient randomised communication in unknown adhoc networks. *Theoretical Computer Science*, 410(27):2549–2561, 2009.

5 Y.-J. Chang, T. Kopelowitz, S. Pettie, R. Wang, and W. Zhan. Exponential separations in the energy complexity of leader election. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 771–783, 2017.

6 Yi-Jun Chang. Energy complexity of distance computation in multi-hop networks. *CoRR*, abs/1805.04071, 2018. `arXiv:1805.04071`.

7 Yi-Jun Chang, Varsha Dani, Thomas P Hayes, Qizheng He, Wenzheng Li, and Seth Pettie. The energy complexity of broadcast. In *Proceedings of the 37th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 95–104, 2018.

8 Yi-Jun Chang, Varsha Dani, Thomas P Hayes, and Seth Pettie. The energy complexity of BFS in radio networks. In *Proceedings of the 39th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 273–282, 2020.

9 Soumyottam Chatterjee, Robert Gmyr, and Gopal Pandurangan. Sleeping is efficient: Mis in o (1)-rounds node-averaged awake complexity. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, pages 99–108, 2020.

10 I. Chlamtac and S. Kutten. On broadcasting in radio networks-problem analysis and protocol design. *IEEE Transactions on Communications*, 33(12):1240–1246, 1985.

11 Varsha Dani, Aayush Gupta, Thomas P. Hayes, and Seth Pettie. Wake up and join me! an energy-efficient algorithm for maximal matching in radio networks. In Seth Gilbert, editor, *35th International Symposium on Distributed Computing, DISC 2021, October 4-8, 2021, Freiburg, Germany (Virtual Conference)*, volume 209 of *LIPIcs*, pages 19:1–19:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPIcs.DISC.2021.19`.

12 L. Gasieniec, E. Kantor, D. R. Kowalski, D. Peleg, and C. Su. Energy and time efficient broadcasting in known topology radio networks. In *Proceedings 21st International Symposium on Distributed Computing (DISC)*, pages 253–267, 2007.

13 S. Gilbert, V. King, S. Pettie, E. Porat, J. Saia, and M. Young. (Near) optimal resource-competitive broadcast with jamming. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 257–266, 2014.

14 B. Haeupler and D. Wajc. A faster distributed radio broadcast primitive. In *Proceedings of the 35th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 361–370, 2016.

15 T. Jurdzinski, M. Kutylowski, and J. Zatopianski. Efficient algorithms for leader election in radio networks. In *Proceedings of the 21st Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 51–57, 2002.

16 T. Jurdzinski, M. Kutylowski, and J. Zatopianski. Energy-efficient size approximation of radio networks with no collision detection. In *Proceedings of the 8th Annual International Conference on Computing and Combinatorics (COCOON)*, pages 279–289, 2002.

17 T. Jurdzinski, M. Kutylowski, and J. Zatopianski. Weak communication in radio networks. In *Proceedings of the 8th International European Conference on Parallel Computing (Euro-Par)*, pages 965–972, 2002.

18 T. Jurdzinski, M. Kutylowski, and J. Zatopianski. Weak communication in single-hop radio networks: adjusting algorithms to industrial standards. *Concurrency and Computation: Practice and Experience*, 15(11–12):1117–1131, 2003.

19 T. Jurdzinski and G. Stachowiak. Probabilistic algorithms for the wakeup problem in single-hop radio networks. In *Proceedings of the 13th International Symposium on Algorithms and Computation (ISAAC)*, pages 535–549, 2002.

20   J. Kabarowski, M. Kutylowski, and W. Rutkowski. Adversary immune size approximation of single-hop radio networks. In *Proceedings Third International Conference on Theory and Applications of Models of Computation (TAMC)*, pages 148–158, 2006.

21   M. Kardas, M. Klonowski, and D. Pajak. Energy-efficient leader election protocols for single-hop radio networks. In *Proceedings 42nd International Conference on Parallel Processing (ICPP)*, pages 399–408, 2013.

22   V. King, S. Pettie, J. Saia, and M. Young. A resource-competitive jamming defense. *Distributed Computing*, 31:419–439, 2018.

23   M. Klonowski and D. Pajak. Brief announcement: Broadcast in radio networks, time vs. energy tradeoffs. In *Proceedings 37th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 115–117, 2018. `doi:10.1145/3212734.3212786`.

24   Marek Klonowski and Malgorzata Sulkowska. Energy-optimal algorithms for computing aggregative functions in random networks. *Discrete Mathematics & Theoretical Computer Science*, 17(3):285–306, 2016.

25   M. Kutylowski and W. Rutkowski. Adversary immune leader election in ad hoc radio networks. In *Proceedings 11th Annual European Symposium on Algorithms (ESA)*, pages 397–408, 2003. `doi:10.1007/978-3-540-39658-1_37`.

26   G. L. Miller, R. Peng, and S. C. Xu. Parallel graph decompositions using random shifts. In *Proceedings of the 25th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 196–203, 2013.

27   Gary L Miller, Richard Peng, Adrian Vladu, and Shen Chen Xu. Improved parallel algorithms for spanners and hopsets. In *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 192–201, 2015.

28   K. Nakano and S. Olariu. Energy-efficient initialization protocols for single-hop radio networks with no collision detection. *IEEE Trans. Parallel Distrib. Syst.*, 11(8):851–863, 2000.

29   Daisy Phillips. Tessellation. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6(3):202–209, 2014.

30   J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 364–369, 2005.

A   Proofs

Proof of Lemma 3. Let $u, v \in V$. Let $d^*([u], [v]) = k$ and let $W_0, W_1, \ldots, W_k \in V(G/\mathcal{P})$ be such that $W_0 = [u], W_k = [v]$ and $W_0, W_1, \ldots, W_k$ is a shortest path from $[u]$ to $[v]$ in $G/\mathcal{P}$. Then, for $0 \le i < k$, since W_i and W_{i+1} are adjacent in $G/\mathcal{P}$, there are vertices $y_i \in W_i, x_{i+1} \in W_{i+1}$ such that $(x_{i+1}, y_i) \in E(G)$. Let $x_0 = u$ and $y_k = v$. For all i, since x_i and y_i are in the same cluster, $d(x_i, y_i) < \beta R$, and if we connect these pairs by shortest paths, we have constructed a path of length at most $k + (k+1)\beta R$ from u to v in G. It follows that

$$d(u, v) \le k + (k+1)\beta R = k(\beta R + 1) + \beta R$$

Since $k = d^*([u], [v])$, rearranging terms gives us the first inequality.

Now suppose $d(u, v) = qR + r$, where $r < R$. Consider a shortest path from u to v in G, and let w_i be the vertices at distance iR from u on the path. The path has thus been cut into $q' = q + \lfloor r/R \rfloor = \lceil d(u,v)/R \rceil$ pieces each of length at most R. Let $w_{q'} = v$. By the first property, $d^*([w_i], [w_{i+1}]) \le \alpha$. Therefore, by the triangle inequality,

$$d^*([u], [v]) \le \alpha \left\lceil \frac{d(u,v)}{R} \right\rceil$$

which completes the proof.   ◀

Proof of Lemma 4. If X is an exponentially distributed random variable with mean R then $\mathbf{P}\left(X > t\right) = e^{-t/R}$. Thus, the probability that $X > 3R \log n$ is at most $1/n^3$. By a union bound, it follows that with probability at least $1 - \dfrac{1}{n^2}$ the first cluster starts forming *after* time $-3R \log n$ and since each vertex either joins a cluster or starts its own by time $t = 0$, the cluster diameters are at most $3R \log n$.   ◀

Proof of Lemma 5. This is just a restatement in our notation of Lemma 2.1 from [8], and is proved there.   ◀

Proof of Corollary 6. Setting $\ell = R$ we see that for any $v \in V$, the probability that there are more than $C \log n$ clusters intersecting the ball of radius R around v is at most

$$\left(1 - \frac{1}{e^2}\right)^{C \log n} \le \exp\left(\frac{-C \log n}{10}\right).$$

The result follows.   ◀

Proof of Lemma 9. Let $w \in B_j(v, 2\alpha_j)$. Then, by definition, $d_j([v]_j [w]_j) \le 2\alpha_j$. Since $\mathcal{P}_j$ is an (R_j, α_j, β_j) approximate distance-preserving partition, by Lemma 3,

$$\left\lfloor \frac{d(v, w)}{\beta_j R_j + 1} \right\rfloor \le 2\alpha_j$$

Removing the floor and rearranging the terms, we get

$$\begin{aligned}
d(v, w) &\le (\beta_j R_j + 1)(2\alpha_j + 1) \\
&\le (2\alpha_j + 1)(\beta_j + 1)R_j \\
&\le R_{j+1} + R_j
\end{aligned}$$

where the last line follows from Equation (1). By the triangle inequality,

$$d(u, w) \le d(u, v) + d(v, w) \le R_{j+1} - R_j + R_{j+1} + R_j = 2R_{j+1}$$

Applying Lemma 3 again, we have

$$d_{j+1}([u]_{j+1}, [v]_{j+1}) \le \alpha_{j+1} \left\lceil \frac{d(u, v)}{R_{j+1}} \right\rceil \le 2\alpha_{j+1}$$

so that $w \in B_{j+1}(u, 2\alpha_{j+1})$, completing the proof.   ◀

Proof of Theorem 11. The proof is by induction of the number of computational steps. Suppose we have fixed A, v, x, and that SIM is known to be an always-correct simulator. Our inductive hypothesis is that, by the times the cumulative energy costs of $Greedy(A, v, x)$ and $\text{SIM}(A, v, x)$ both reach i, the greedy simulation will have completed simulating at least as many timesteps of A as SIM will have. Assume the hypothesis for $i - 1$. Then SIM wakes up for the i'th time at or before Greedy does, say at time t. At this point it can go back to sleep, but the latest it can sleep is $t^*(v, t)$, defined as above, since SIM needs to be safe. Greedy wakes up for the i'th time at some time $t' \ge t$, and goes to sleep until time exactly $t^*(v, t')$. Now, the crucial point is that the function $t^*(v, \cdot)$ is an increasing function of its second argument, which is clear from its intention, but also from the facts that $t_{\text{next}}(w, \cdot)$ can only decrease when w receives a message from a neighbor, which implies that $t^*(w, t) - t \le 1$, and that $\mathsf{dist}(w, \text{SENDERS})$ never decreases by more than 1 in a single timestep. Hence $t^*(v, t') \ge t^*(v, t) \ge$ whenever SIM decides to wake back up, which completes the inductive step.   ◀

Proof of Lemma 13. Let u be the nearest node to v that would initiate a SEND in the epoch. In order to participate in a call to NOTIFY$(j-1)$, v must receive a "WAKE UP" return value from the call to NOTIFY(j) at the beginning of the epoch. However, this will not happen because, by Lemma 3, $d^*([u],[v]) \geq \left\lfloor \frac{d(u,v)}{\beta_j R_j + 1} \right\rfloor \geq 2\alpha_j$, but this means, if v participates in the NOTIFY, it will instead get a return value of "OK TO SLEEP." Here d denotes distance in the underlying graph, and d^* denotes distance in the level-j cluster graph. ◀

Proof of Theorem 14. First, observe that, after each NOTIFY operation, a node v stays awake if and only if it is within $2\alpha_j$ level-j clusters of the set of active nodes for the beginning of the corresponding level-j epoch. That is, assuming that all of our cluster operations (Upcast/Downcast/Intercast) succeed. Since, with high probabiltity, this happens at all timesteps, let us discount the $1/\text{poly}(n)$ chance of a failure.

Since the level-j epoch is R^j time units long, every active node in the entire epoch is within distance R^j of the initial set of active nodes. By the definition of approximately distance-preserving partition, these nodes are all within distance α_j of the initial active set in the level-j cluster graph. Since we doubled this radius of notification in the cluster graph, it follows by Lemma 9 that all nodes in lower-level clusters that wake up during the simulation of this level-j epoch are already awake at level j, and therefore their lower level cluster operations will succeed.

In particular, at level 0, it follows that any time protocol f has a node Send, and one of its neighbors Listen, that both nodes in question will be awake and simulating f at the corresponding timestep. It follows by induction that the state of all processors at any simulated time t is consistent with f run under the OR model until time t. Hence, the final outcome will also be consistent with f run under the OR model.

For the latency analysis, we observe that the number of level-j epochs is T_f/R^j where T_f is the running time of f. For each such epoch, we do $O(\alpha_j)$ level-j cluster operations (Upcast/Downcast/Intercast), which each require time $O(\beta_j R_j \log(n) \log\log(n))$, since the clusters have diameter at most $\beta_j R_j$, and the backoff requires $\log(\log(n))$ time because each node is, with high probability next to $O(\log n)$ different clusters; the further $O(\log(n))$ factor is to guarantee success with high probability. Summing over all epochs, we get a total running time of $O(T_f \alpha_j \beta_j \log(n) \log\log(n))$ attributable to level-j operations. Finally, summing over the $\leq \log(n)$ levels, and using that $\alpha_j, \beta_j = O(\log n)$ for all j, we get a total running time of $O(T_f \log^4(n) \log(\log(n)))$.

For the energy analysis, we consider the time interval between two consecutive non-SLEEP actions by the Greedy Psychic algorithm, so that OPT is spending one energy. If this interval has length L, then we know that after the first i timesteps of this interval, the distance from v to the nearest active vertex is always at least $L-i$. Consequently, applying Lemma 13, we know that v can, at most, participate in a subset of the final $(2\alpha_j + 1)\beta_j$ of the calls to NOTIFY(j) during this time interval, assuming $j < \ell$. Since each call to NOTIFY costs at most $O(\alpha_j \log n)$ energy per participating vertex, this means our algorithm spends at most $O(\ell \alpha_j \log n)$ times more energy than the Greedy Psychic algorithm does. Finally, for the top level, we get T/R_ℓ calls to NOTIFY(ℓ) in all, each at a cost of $\alpha_j \log n$, regardless of distances. Summing these energy costs completes the proof. ◀

Proof of Theorem 15. By Theorem 14, we know that the algorithm succeeds in building each level of the clustering with essentially the same success probability as Naive-Build-MPX (Algorithm 4). Since the naive algorithm runs in time proportional to the radius parameter, and these radius parameters form a geometric sequence from 1 up to $O(D)$, the total running

time for all the calls to Naive-Build-MPX is $O(D\mathrm{polylog}(n))$. Similarly, we can estimate the expected number of active j'-epochs for Naive-Build-MPX(R^j) as $O(R)$. This is because the active set moves out from the cluster centers at unit velocity, so once it gets within distance $R^{j'}$ of a vertex, all activity within distance $R^{j'}$ ceases within at most $2R^{j'}$ timesteps, by the Triangle Inequality. Thus the total number of active j-epochs for a given node is $O(1)$. Summing over all $0 \le j \le \ell$, we get a total energy use of $\tilde{O}(\ell)$, which is polylog(n). ◀

Proof of Theorem 16. Theorem 15 gives us the running time and cost for the cluster formation. Then we can use Theorem 14 to analyze the SAF simulation of the naive BFS algorithm. The analysis of the number of active epochs here is essentially the same as in the proof of Theorem 15, since the active set again passes very quickly through each region that it enters. We leave the details to the reader. ◀

B   Simulating cluster-graph algorithms on the underlying graph

Suppose we already have approximately distance-preserving partitions at all scales from 1 up to the diameter of G. The ability to run graph algorithms on the corresponding quotient graphs will be a rather useful primitive to add to our toolbox. Because the distances in these graphs are scaled down by a large factor, we may reasonably expect that they can be run at a much lower cost. But what is the cost of simulating these algorithms on the actual network?

For the most part, this question was already answered by Chang, Dani, Hayes and Pettie [7]. We briefly describe the approach used to simulate one timestep of computation on the cluster graph. Each node within one cluster in our partition uses part of its memory to record the state of a hypothetical processor corresponding to the cluster. At the beginning and end of the simulated timestep, we require that this state be the same for every node in the cluster. Depending on whether the cluster state indicates we should Send, Listen, or Sleep, every node in the cluster does this (INTERCAST). Next, if the operation was Listen, all the nodes that received a message propagate these up towards the cluster center (UPCAST). Since every node in the cluster knows its distance from the root, this propagation can be synchronized so that each node only needs to Listen once, in the same timestep that its children might Send. Since we are in the OR model, we only require that, from among the messages received, an arbitrary one is received. Next, the cluster center updates its state based on the received message, and broadcasts the result within the cluster (DOWNCAST), after which the simulation of one computational step on the cluster graph is complete.

■ **Algorithm 7** High-level description of Upcast, Downcast, and Intercast algorithms, from [7].

▷ For each of the subroutines below, we require that either all the nodes in a particular cluster participate, or none do.
procedure UPCAST(j, m)   ▷ j: current cluster height, m: message to send
| ▷ Guarantee: if any node in the cluster participates with a non-null message, then one of these messages is received by the cluster center, who then stores it in their message variable.
procedure DOWNCAST(j, m)   ▷ j: current cluster height, m: message to send
| ▷ Guarantee: if the cluster center participates with a non-null message, then this message is received by each node in the cluster, who then stores it in their message variable.
procedure INTERCAST(j, m)   ▷ j: current cluster height, m: message to send
| ▷ Guarantee: if at least one neighboring node is contained in a participating level-j cluster that has a non-null message, then this node receives such a message, and store it in their message variable.

Since the messages are being broadcast, rather than sent along edges, there are some subtleties in the details. For instance, even though we have described the algorithm under the assumption that message delivery in both the underlying graph, and in the cluster graph, takes place using the collision-free OR model, there is still a need for backoff, to prevent messages accidentally crossing between adjacent clusters during UPCAST and DOWNCAST. However, for purposes of the present work, these details are unimportant.

Since each of the three stages, INTERCAST, UPCAST, and DOWNCAST, costs $\tilde{O}(1)$ energy per node in the cluster, the per-node energy cost for running an algorithm on the cluster graph is within a polylog factor of the per-node energy cost for simulating it on the underlying graph. When the cluster "radius" parameter is R, the latency for the UPCAST and DOWNCAST is $\tilde{O}(R)$, so this becomes a multiplicative factor for the time complexity.