Towards Desynchronization of Multi-hop Topologies

Julius Degesys and Radhika Nagpal
School of Engineering and Applied Sciences, Harvard University

E-mail: {degesys,rad}@eecs.harvard.edu

Abstract

In this paper we study desynchronization, a closely-related primitive to graph coloring. A valid graph coloring is an assignment of colors to nodes such that no node's color is the same as a neighbor's. A desynchronized configuration is an assignment of real values in S^1 to nodes such that each node's value is exactly at the midpoint of two of its closest neighbors' values. Recent work has shown that a simple, self-organizing algorithm, DESYNC*, can solve desynchronization in single-hop networks, with applications to collision-free wireless broadcast [1] and duty-cycling [2]. Here we generalize this work by defining and analyzing desynchronization for multi-hop networks and experimentally analyzing the* DESYNC *algorithm's behavior for multi-hop networks. We describe desynchronized configurations for several classes of graphs (lines, rings, two-colorable, and hamiltonian cycles) and discuss the relationship with other variants of graph coloring. We extend the* DESYNC *algorithm and* DESYNC*-based resource allocation to multi-hop networks and study the performance and efficiency of resource allocation in simulation. While many applications for graph coloring require synchronization and an agreement on a schedule to be effective, we show that the self-organizing algorithm,* DESYNC*, does not require either of these to achieve desynchronization and to define a resource-allocation schedule. Although applications to wireless sensor networks pose some unique problems, the results suggest that* DESYNC *has significant potential as a lightweight method for providing non-overlapping variable-sized slots in ad-hoc multi-hop settings.*

1 Introduction

We consider the following problem: given a large, ad-hoc group of agents, how does one manage resource that needs to be periodically or continually shared among them. The agents want to access this resource as much as possible, but some are constrained by one another, meaning that particular subsets of the agents cannot access the resource at the same time. The primary question is how to get a set of constrained agents to agree on a resource-access schedule amongst themselves in a simple and lightweight manner.

An example of where this might be useful is a wireless sensor network (WSN), where many wireless sensor devices are often densely deployed in an ad-hoc fashion. The purpose of the network is to sense the environment and communicate the data to other nodes in the network (such as a base station) in a timely manner. If all nodes simply try to send their data at the same time (as may happen after an interesting event triggers the collection of data (e.g., a volcano eruption [8]), then significant data loss would occur. Just as an individual gets confused by trying to listen to more than one person talking at a time, devices are often constrained to receive data from only one other device at a time.

Efficiently utilizing the network requires that a schedule be determined and agreed upon that can allow for the collision-free transmission of sensed information. This is usually accomplished through graph coloring, a classic and well-studied technique [3]. In order to use it for resource scheduling, the following tasks must be performed (not necessarily sequentially):

- All agents agree on a set of available colors.
- All agents agree on a schedule that specifies when particular colors have access to the resource. This is usually done by having nodes synchronize time, divide time into slots of fixed size, and then assign each slot a color.
- Each agent chooses a color such that no two mutually constrained nodes have the same color.

Once these three tasks have been accomplished, each agent can use its color and the agreed-upon schedule to access the resource in a conflict-free manner. However, this

is not always a practical means for many systems for several reasons. In order to minimize latency and increase efficiency, the minimal number of colors should be used, but this is NP-Hard. Approximation algorithms [5] exist, but the nodes must still synchronize, agree on a schedule, and color themselves before beginning to share the resource. Furthermore, accomodating changes in the network topology can require a full re-coloring of the graph, causing a global reaction to a local change. These difficulties are sufficient to motivate the study of an alternative approach.

In this paper we study desynchronization, a closely-related primitive to graph coloring. Whereas graph coloring seeks to assign colors such that no two mutually constrained nodes have the same color, desynchronization is an assignment of positions on a circle such that mutually constrained nodes are as "far apart" from one another. Each position on the circle is called a "phase" and is represented by a real value $\phi \in S^1 = [0, 1]$, where 0 and 1 are glued together and are treated as the same phase. Just as graph coloring maps colors to fixed-size slots, desynchronization maps phases to variable-size slots that can be used for resource allocation.

Recent work in desynchronization has shown that very simple, self-organizing algorithms (such as DESYNC[1]) can be used to achieve desynchronization. They have been effectively used for tasks such duty-cycling [2] and collision-free wireless broadcast [4] on single-hop networks. The desynchronization of multi-hop networks, however, has not been deeply studied or even well-defined.

This paper generalizes this work by defining and analyzing desynchronization for multi-hop networks and experimentally analyzing the DESYNC algorithm's behavior for multi-hop networks. We show that many desynchronized configurations can exist for a given graph, each with various resource-allocation efficiencies. Furthermore, we give experimental evidence showing that the DESYNC algorithm still achieves desynchronized configurations on general topologies; however, the particular schedule's efficiency is determined by the starting conditions. We discuss the use of DESYNC for collision-free wireless transmissions in multi-hop networks and show that if DESYNC is modified to operate on the constraint network, then it can provide an effective and lightweight solution. While significant work remains to extend DESYNC to this setting, DESYNC may be preferable to graph-coloring for ad-hoc system because DESYNC does not require global time synchronization and can quickly and continually adapt to any changes in the topology.

The rest of this paper is organized as follows: Section 2 formally defines desynchronization, its framework, and analyzes several simple types of graphs, providing intuition and evidence that a solution to desynchronization exists for most graphs. Section 3 gives a summary of the DESYNC algorithm and uses experimental evidence to suggest that DESYNC converges to a desynchronized state on a multi-hop topology. Section 3.5 describes the hidden terminal problem, its role in desynchronization, and how it can be overcome. Lastly, we briefly discuss related work and conclude in Sections 4 and 5.

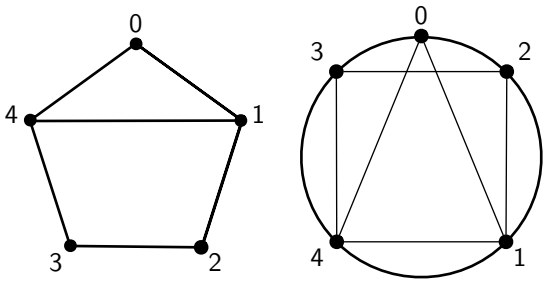

Figure 1. (a) The "house" topology (b) The configuration $\vec{\phi} = (0, 0.375, 0.125, 0.875, 0.625)$ displayed on the phase ring. The lines in between nodes represent the edges from the corresponding topology.

2 Desynchronization

Desynchronization on a single-hop network is defined as assignment of phases to nodes such that each phase $\phi_i \in S^1$ has a minimum separation of $1/n$ from neighboring phases [1, 4]. Visually, this corresponds to evenly distributing n phases on the circumference of the circle S^1 (see Figure 3b).

We use a similar definition for desynchronization of a multi-hop network. First, however, we must introduce some notation: define the distance from a phase ϕ_i to ϕ_j as

$$\Delta_{i,j} \equiv \phi_j - \phi_i \pmod 1,$$

and define the previous and next phase neighbors as

$$p(i) = \operatorname*{argmin}_{j \in N(i)} \Delta_{i,j} \quad (1)$$

$$n(i) = \operatorname*{argmax}_{j \in N(i)} \Delta_{i,j}, \quad (2)$$

where $N(i)$ is the set of neighbors of i in the multi-hop network. We then say that a configuration is **desynchronized** if $\Delta_{i,p(i)} = \Delta_{n(i),i}$ for all i. For the sake of brevity, we will refer to these distances from i to its previous and next neighbors as $\Delta_p(i)$ and $\Delta_n(i)$, respectively.

Example 2.1. As an example, consider the "house" topology shown in Figure 1. The depicted configuration is $\vec{\phi} = (0, 0.375, 0.125, 0.875, 0.625)$. Looking at Node 1, for instance, we find that the previous and next phase neighbors (respectively) are nodes 4 and 2, which is given by the min-

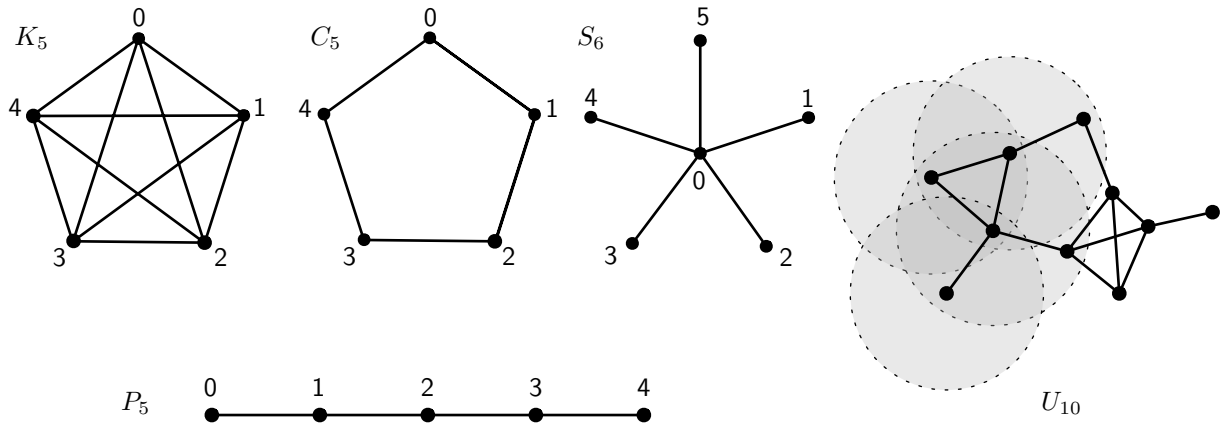

Figure 2. Examples of simple graphs (left-to-right, top-to-bottom): A complete graph, K_5; a cycle graph, C_5; a star graph, S_6, a 10-node unit-disk graph, U_{10}; a path graph, P_5. The circles that are overlaid on the unit-disk graph show the radius of connectivity for a few nodes.

imum and maximum of the following:

$$\begin{aligned}\Delta_{1,0} &= \phi_0 - \phi_1 \equiv 0.625 \pmod 1 \\ \Delta_{1,2} &= \phi_2 - \phi_1 \equiv 0.750 \pmod 1 \\ \Delta_{1,4} &= \phi_4 - \phi_1 \equiv 0.250 \pmod 1.\end{aligned}$$

Thus, $\Delta_p(1) = 0.250$ and $\Delta_n(1) = 1 - \Delta_{1,4} = 0.250$. Since $\Delta_p(i) = \Delta_n(i)$ for all i, this assignment of phases is desynchronized.

2.1 Simple classes of connected graphs

With a definition of desynchronization in hand, we now consider several simple, canonical graphs and define the desynchronization states as a means of building intuition. While these topologies may not appear in real deployments, they become apparent when one considers just the traffic patterns within the network. For example, point-to-point communication is effectively a path graph, and tree graphs commonly occur in sensor networks when there is a single base station.

Fully-connected Graph (K_n): each node is connected to every other node.

Path Graph (P_n): two nodes have degree 1 and $n-2$ nodes have degree 2.

Cycle Graph (C_n): every node has degree 2.

Star Graph (S_n): one node has degree $n-1$ and all other nodes have degree 1.

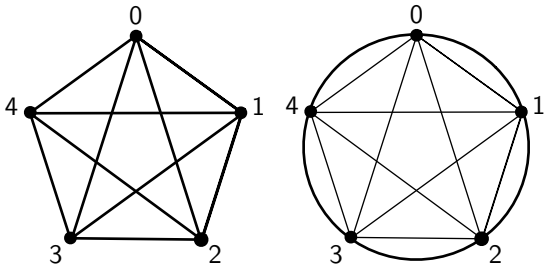

Figure 3. (a) The K_5 topology is on the left. (b) A desynchronized configuration for K_5 on the right.

Unit-disk Graph (U_n): a graph for which every vertex $v \in V$ can be mapped to a 2-dimensional point, $f(v) = (x, y)$, where $(u, v) \in E \iff \|f(u) - f(v)\|_2 \leq r$ for some radius r.

A **tree graph** is any graph that does not contain cycles. Note that trees are always two-colorable.

2.2 Fully-connected Graphs: K_n

For a fully-connected network, desynchronization corresponds to only one qualitative configuration. This is the configuration in which all phases are evenly spread out on the phase ring (i.e., where $\Delta_p(i) = \Delta_n(i) = 1/n$ for all i). A desynchronized configuration can be seen in Figure 3.

To see that this is the only possible desynchronized state, note that $p(n(i)) = i$ for all i. In other words, oscillator i is the previous phase neighbor of its next phase neigh-

bor. Thus, $\Delta_n(i) = \Delta_p(n(i))$, which must be equal to $\Delta_n(n(i))$. Repeatedly applying this leads to the constraint that the following n non-overlapping intervals must be equal: $\Delta_n(i) = \Delta_n(n(i)) = \cdots = \Delta_n(n(\cdots(n(i))\cdots))$. Since there are n of them, the only way in which they can sum to 1 is each one is equal to $1/n$.

2.3 Two-colorable graphs

Path graphs: P_n

There is only one desynchronized configuration for the path graph: $\Delta_p(i) = \Delta_n(i) = 1/2$.

To see this, label the nodes of the line 0 to $n-1$ and consider node 0. Since it has only one neighbor, node 1, we require $p(0) = n(0)$ for the configuration to be desynchronized. The only assignment that can satisfy $\Delta_p(0) = \Delta_n(0)$ is $\Delta_p(0) = \Delta_n(0) = 1/2$. Since node 1 only has two phase neighbors (0 and 2), node 0 is either the previous or the next phase neighbor of node 1. Since either $\Delta_p(1)$ or $\Delta_n(1)$ is already 1/2, the other must be as well. This argument applies to all nodes $i < n-1$, and node $n-1$ is the same case as node 0.

Thus, all even nodes share the same phase and all odd nodes share the antipodal phase on the phase ring. This is equivalent to two-coloring the line graph.

Star graphs: S_n

The star graph shares the same desynchronized configuration as the line graph. Label the center node of the star 0. Then, a desynchronized configuration corresponds to $\phi_0 = 0$ and $\phi_i = 0.5$ for all $0 < i < n$. As in the path graph, this assignment of phases effectively two-colors the graph. One can similarly create a desynchronized configuration for any rooted tree by assigning the root and even levels of the tree the phase 0 and odd levels of the tree the phase 0.5.

Observing the two-phase configurations that exists for these graphs leads to the following:

Claim 2.2. *Any two-colorable graph has a desynchronized configuration in which all nodes colored one color are assigned a common phase and all other nodes share the antipodal phase.*

Proof. Let the two colors be red and blue. Without loss of generality, assign all the nodes colored red the phase $\phi = 0$ and the nodes colored blue the phase $\phi = 0.5$. For any node $i \in V$, all of $N(i)$ must be assigned the phase $\phi_i + 0.5 \pmod 1$. Thus, $\Delta_p(i) = \Delta_n(i) = 1/2$ for all nodes. □

Remark 2.3. Similarly, if a two-phase desynchronized configuration exists for a graph then that graph is bi-partite, and therefore, two-colorable. Note, however, that a two-colorable graph may contain many desynchronized configurations other than the one that contains only two phases. Cycle graphs provide an example of this.

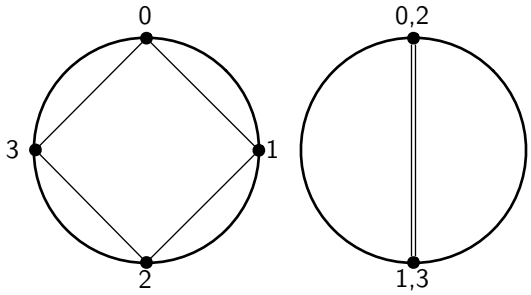

Figure 4. Two unique desynchronized states for C_4.

2.4 Cycle graphs: C_n

We have already found the desynchronized states for C_2 and C_3, since they can also be represented as P_2 and K_3, respectively. The cycle graph on 4 nodes presents the first non-trivial case. There are two unique desynchronized configurations for C_4 (Figure 4), one with four distinct phases and another with two. The first configuration shows why there can be multiple desynchronized configurations for two-colorable graphs apart from two antipodal phases. In fact, we can generalize this configuration to make the following claim:

Claim 2.4. *For C_n, there exists a desynchronized configuration such that $\Delta_p(i) = \Delta_n(i) = 1/n$.*

Proof. We give a constructive proof. Label the nodes along the cycle 0 to $n-1$ such that $(i, i+1 \pmod n) \in E$ for all $0 \leq i < n$. Assign the nodes the phases $\phi_i = i/n$. Since $p(i) \equiv i+1 \pmod n$ and $n(i) \equiv i-1 \pmod n$, $\Delta_p(i) = \Delta_n(i) = 1/n$. □

Corollary 2.5. *Any graph that contains a Hamiltonian cycle has at least one desynchronized configuration.*

Proof. As in the previous claim, we can label the nodes along the cycle 0 to $n-1$ and assign the phases $\phi_i = i/n$. See Figure 5. □

Claim 2.6. *There is a desynchronized configuration for C_n associated with each factor of n greater than 1, including n.*

Proof. Label the nodes 0 to $n-1$ according to a traversal of the cycle. Given any factor f of n such that $1 < f \leq n$, node i is not constrained by node $i + kf \pmod n$ for any $0 < k < n$ since $i + kf \not\equiv i \pm 1 + kf \pmod n$. Assign

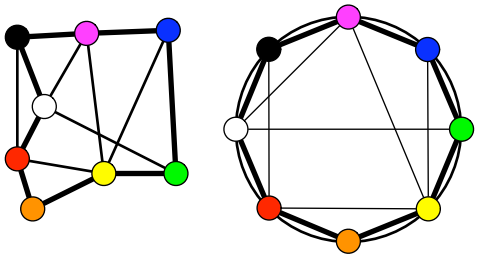

Figure 5. (a) A graph that contains a Hamiltonian cycle (depicted in bold) (b) A desynchronized configuration that exists in any Hamiltonian graph

to each node the phase given by $\phi_i = \frac{i \pmod f}{f}$. Observe that $i \pm 1 + kf \pmod n$ are the phase neighbors of i to which the phases $\frac{i\pm1 \pmod f}{f}$ have been assigned. Since ϕ_i is at the midpoint of these phases, $\Delta_p(i) = \Delta_n(i)$, and thus, configuration is desynchronized. Note that $f = n$ gives the same desynchronized configuration as in Claim 2.4. □

Example 2.7. For $n = 4$, there are two unique factors: 2 and 4. Figure 4 shows the associated configurations (2 is on the right, 4 is on the left).

3 DESYNC

We have shown that several desynchronized solutions exist for simple graphs; however, we have not provided a way to find these solutions. Previously, the DESYNC algorithm [1, 4] has been shown to solve desynchorinaziotn on fully-connected networks. Here, we give a summary of the DESYNC algorithm and provide intuitive and experimental results that suggest that DESYNC also converges to a desynchronized state on multi-hop topologies.

3.1 Pulse-coupled Oscillator Framework

We assume that each agent is equipped with an oscillator, a device that fires periodically with a constant frequency $\omega = T$. We can model the evolution of an oscillator as a point racing around a circle. The point moves at a fixed speed and completes a lap every T seconds. When the point reaches the top of the circle, the oscillator fires. The percentage of the lap that is completed at any given point in time defines oscillator's state, $\theta(t) \in [0, 1]$ (where 0 and 1 are equated). When $\theta_i(t) = 1$, oscillator i fires and immediately resets its state to 0. We associate each agent's phase, ϕ_i with its oscillator by having it act as an offset in defining an oscillator's state, θ_i:

$$\theta_i(t) = \omega t + \phi_i(t) \pmod 1.$$

3.2 Algorithm

When a node i hears a neighbor j fire at time t_f, i can compute $\Delta_{i,j}(t_f) = \theta_j(t_f) - \theta_i(t_f)$ using only local information since at the time of j's firing, $\theta_j(t_f) = 1$. The last firing that i hears before its own firing belongs to the previous phase neighbor, $p(i)$. Likewise, the firing that i hears immediately after it fires is i's next phase neighbor. Once i has this information, it can adjust its phase, ϕ_i, towards a configuration that is closer to desynchronization. This simple and intuitive algorithm for any node i is given below:

1. When $j \in N(i)$ fires,

```
if (justFired == true)
  justFired = false;
  next = θ_i(t);
  φ_i(t+) = φ_i(t) + α(prev-next)/2;
else
  prev = Δ_{i,j};
end
```

2. When i fires, set `justFired` = `true`.

The variables `prev` and `next` correspond to the respective values $\Delta_p(i)$ and $\Delta_n(i)$ at the time that the update is performed. Thus, each oscillator move its phase in a direction so as to equalizes the phase distances to its phase neighbors. The parameter α acts as a step size.

3.3 Convergence to desynchronization

Previously it was shown that the DESYNC algorithm provably solves desynchronization on K_n for any $\alpha \in (0, 1)$ [1, 4]. For arbitrary topologies, this theoretical question remains open. In the previous section, we showed that for many classes of graphs there exist solutions to desynchronization. We can also show that solutions to desynchronization are fixed points for the DESYNC algorithm. However, it still remains to be proven that these fixed points are stable.

In this section we use simulation experiments to study the behavior of DESYNC on multi-hop networks, using some of the insights developed in the previous section. Our results suggest that DESYNC is always able to converge to a desynchronized solution. We study the DESYNC on several topologies: the poth graph, the cycle graph, and the unit disk graph. The line and cycle graph allows us to compare the simulation behavior to known theoretical results, while the unit disk graph represents more commonly occurring network topologies such as wireless networks. We also hemonstrate a candidate Lyapunov function for DESYNC with small α.

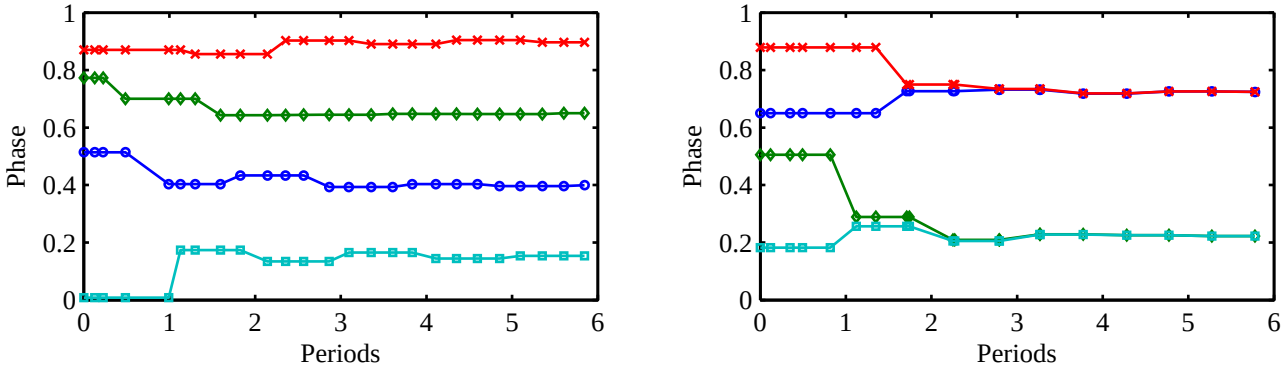

Figure 6. The phase evolution of the DESYNC algorithm over time for two different starting configurations for C_4. The left and right starting configurations converge to the left and right configurations shown in Figure 4.

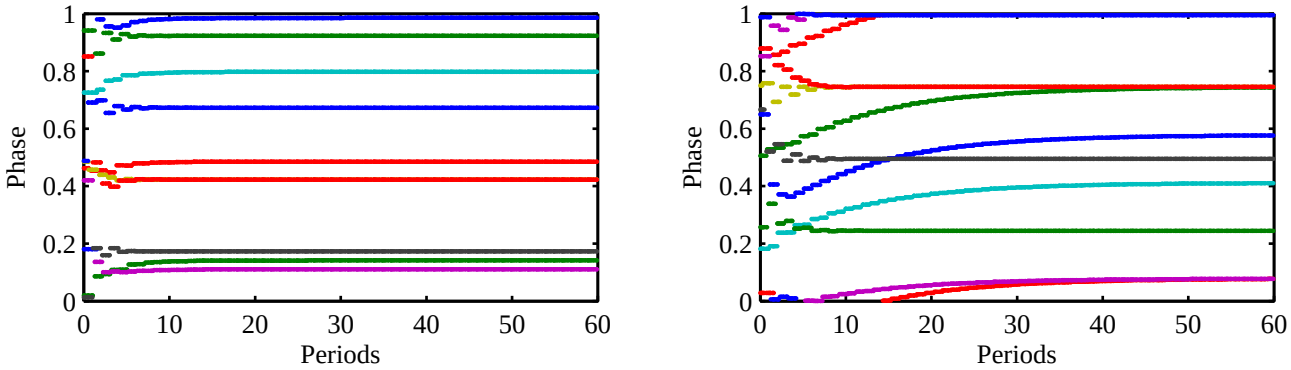

Figure 7. The phase evolution of the DESYNC algorithm over time for two different starting configurations for U_{10} (shown in Figure 2).

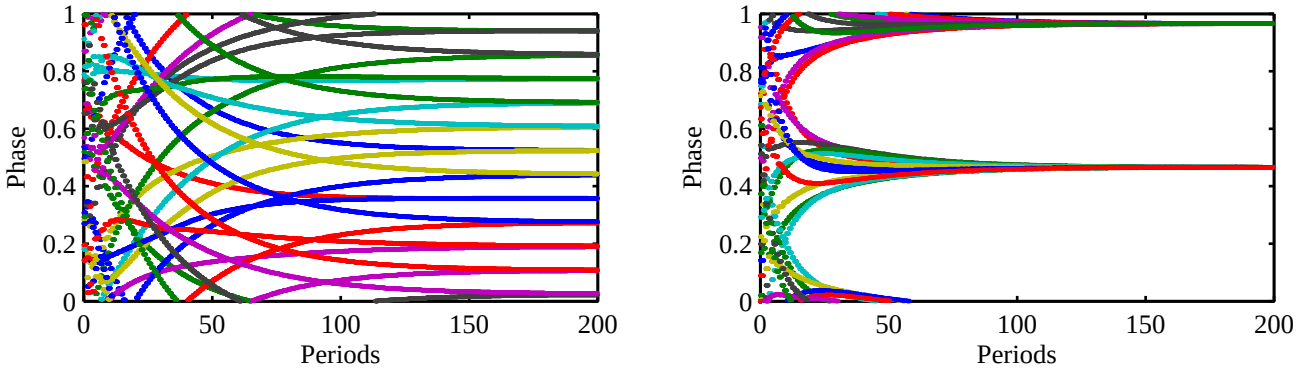

Figure 8. The phase evolution of the DESYNC algorithm over time for two different starting configurations for C_{24}. The left converges to 12 phase clusters of 2 nodes each, whereas the right converges to 2 phase clusters of 12.

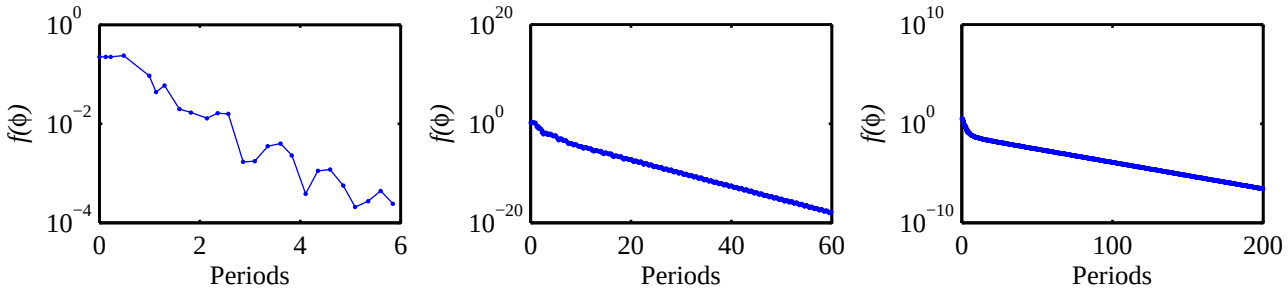

Figure 9. The time evolution of the Lyapunov candidate function, $f(\vec{\phi}) = \sum_i (\Delta_p(i) - \Delta_n(i))^2$, shown on a semilog-y plot. Left-to-right: C_4, U_{10}, C_{24} (correspond to the three graphs on the left from the previous page). Large α can overshoot (all experiments in the paper used 0.9), causing oscillations in the phases (see Figure 6) and an increase in f (seen in the left graph for C_4). Sufficiently small α, however, experimentally leads to a monotonically decreasing lyapunov function.

Figure 6 shows the phase evolution over time of the DESYNC algorithm for two different starting configurations for C_4 with $\alpha = 0.9$.[1] The left and right plots converge to the left and right configurations shown in Figure 4 respectively. Figure 7 shows the phase evolution from two different starting configurations for a more complicated, and more realistic, topology (the topology is U_{10}, which is shown in Figure 2). Lastly, Figure 8 shows DESYNC running on a 24-node cycle topology.

As can be seen for each of these graphs, different starting conditions can lead to different desynchronized states. And despite the crossing of non-constrained phases, the DESYNC algorithm always converges in our experiments.

As further evidence for the fixed points being stable, we suggest the following Lyapunov candidate function:

$$f(\vec{\phi}) = \sum_i (\Delta_p(i) - \Delta_n(i))^2 . \quad (3)$$

Figure 9 shows the time evolution of this function corresponding to the the phase evolutions shown in the left graphs in Figures 6, 7, and 8. While f seems to decrease over time for all of the topologies, it encounters some "bumps" in the left graph. We conjecture this to be due entirely to the large α used for the experiments, which can cause overshooting of the midpoints and oscillations before converging. The middle and right graphs for Figure 9 show that f is monotonically decreasing at an exponential rate. We conjecture that since the nodes are not jumping a large amount due to the density of the network, a large α will not cause oscillations. This, however, is just experimental evidence. We must currently defer the exploration of f as a Lyapunov function to future work.

[1]The parameter α was set to 0.9 for all experiments. This selection of α is arbitrary. While any $\alpha \in (0, 1)$ seems to converge, larger values can cause oscillations that are slowly damped out (e.g., see Figure 6). Analyzing the performance tradeoffs of this parameter's selection is left for future work, but it can be intuitively thought of as a damping parameter.

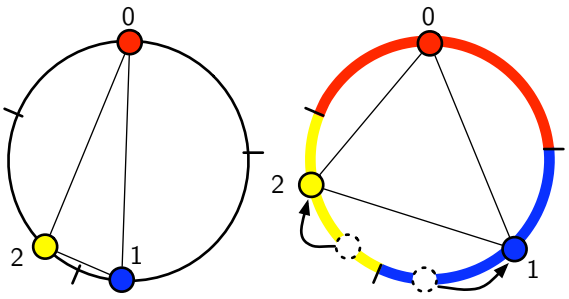

Figure 10. The slot boundaries (shown right) are defined by midpoints of the previous firings (shown left). It is guaranteed that the slots contain the next firing [1, 4].

3.4 Resource Scheduling with DESYNC

A significant motivation for the study of graph-coloring is that it can be used for the scheduling of shared resources. Here we show that even if we did not have convergence of the DESYNC algorithm, it is still possible to define slots for any assignment of phases while running DESYNC. Importantly, these slots are able to be defined without the assumptions of synchronized time or globally-agreed upon schedules.

We define the slot for node i as the interval

$$s_i = \left[1 - \frac{1}{2}\Delta_p(i), \frac{1}{2}\Delta_n(i)\right],$$

where the previous firings are used to compute the Δ values. When $\theta_i(t) \in s_i$, node i is allowed to access the resource. A simple visualization of this slot is that node i's slot is determined by the the previous firings of node i and its two phase neighbors; the slot starts at the midpoint between i's and its previous phase neighbor's previous firings and ends at the midpoint between i's and its next phase neighbor's previous firings. Note that this definition of non-overlapping

slot boundaries is the same as defined by Degesys et al [1]. The same proof that the slots are non-overlapping applies here, but within each node's neighborhood. One difference is that the slots within a neighborhood may not cover all of the time available. Figure 10 gives a graphical depiction of the slot boundaries.

Since all oscillators are identical and have a common frequency, the slots are always well-defined, even if the system has not reached a desynchronized configuration. This implies that even if a network is not yet desynchronized, nodes can still access the resource in a collision-free manner. This is in stark contrast to graph-coloring algorithms that require a full and correct coloring before being able to access the shared resource.

3.5 Wireless Constraint Graphs

One important application of DESYNC-based resource scheduling is the scheduling of collision-free wireless transmissions in a wireless sensor network. In this case the slots represent the times when nodes can talk without interference from their neighbors. However, extending the DESYNC-based approach poses some difficulties. We assumed that nodes are able to communicate to neighbors to which they are constrained. However, this is not the case for many settings—wireless networks being the primary example. Certainly, in a wireless network, neighboring devices are restricted from communicating at the same time (otherwise, receiving devices that are in range of both would hear two messages jumbled together). This implies that the communication graph must at least be a subgraph of the constraint graph.

Wireless networks also contain the indirect constraints imposed by hidden terminals. Consider three devices, A, B, C in a path topology, where B is the middle node connected to both A and C. In this example, if both A and C send at the same time to B, neither message will be received. Thus, A acts as a hidden terminal to C and C acts as a hidden terminal to A. Without forwarding of messages, nodes are unable to communicate directly to conflicting hidden terminals. In order to take hidden terminal effects into account, the constraint graph must also have the edge set $\{(u, v) : (u, w) \in E, (w, v) \in E\}$. In other words, all nodes that can be reached by a path of length 2 on the communication graph should have an edge between them in the constraint graph.

To observe the effect that hidden terminals might have if the wireless medium was the resource that was being shared, we compare the results of running DESYNC on the communication and constraint topologies. The duration of all experiments was 5 minutes with $T = 1$ second $= 1/\omega$ and $\alpha = 0.9$. Figures 11 and 12 show the topologies and the final desynchronized configurations. The top phase ring in each figure shows the result of running DESYNC on just the communication topology and the bottom rings show the resulting configuration if DESYNC could be run on the constraint topology.

As a metric, we compare the average slot sizes that nodes used to the average slot sizes that were safe (i.e., the fraction of the period during which a node would have its slot defined and not encounter hidden-terminal effets). For example, for P_3, node 0 and node 2 will have slots that entirely overlap. If both nodes are continually sending data, then neither node would be able to get any data through to node 1. Figure 11 confirms this intuition. The top graph shows that not taking hidden-terminal constraints into account can be highly detrimental since neither 0 nor 2 has a safe moment in its slot. The bottom graph indicates that to avoid hidden terminals, it is necessary to run DESYNC on the constraint topology. Doing this results in all nodes in P_3 receiving a slot size of 1/3.

To give an indication of how "connected" a node is (apart from the topology), we also show $1/(k_1(i) + 1)$ and $1/(k_2(i) + 1)$ for each node, where $k_1(i)$ is the the node's degree and $k_2(i)$ is the number of nodes within a path-length of 2 of node i (i.e., the size of its two-hop neighborhood). These value are an indication of the slot size that might be achieved with a standard distributed graph coloring algorithm that performs max-degree + 1 coloring.

It may seem that P_3 is a bit contrived and that most real topologies do not encounter the hidden-terminal problem. Running the exact same experiment for the U_{10} topology invalidates this thought. The top graph in Figure 12 shows a very similar story to the one given by P_3. Namely, most nodes will end up suffering greatly from the hidden-terminal effects if all of them were to send data throughout their slots. The bottom graph shows that when DESYNC is run on the constraint topology, the slots are fairly distributed and unaffected by hidden terminals. Furthermore, this configuration achieves a larger slot size than could be achieved with a max-degree + 1 coloring.

This configuration, however, is just one of many possible desynchronized configurations for this graph. As mentioned previously, determining the final configuration (and therefore the final resource-allocation scheme) that results from DESYNC seems to be a very difficult problem. As an initial step, though, DESYNC on a multi-hop graph is able to achieve comparable performance to the max-degree + 1 coloring, and does so without requiring the nodes to be synchronized and without requiring a graph coloring.

These results indicate that directly running DESYNC on a communication graph will result in data loss due to hidden-terminal effects. However, modifying DESYNC to operate on the constraint graph will provide a reasonable solution. We are currently developing a simple 2-hop DESYNC algorithm in which nodes forward the phases of their neigh-

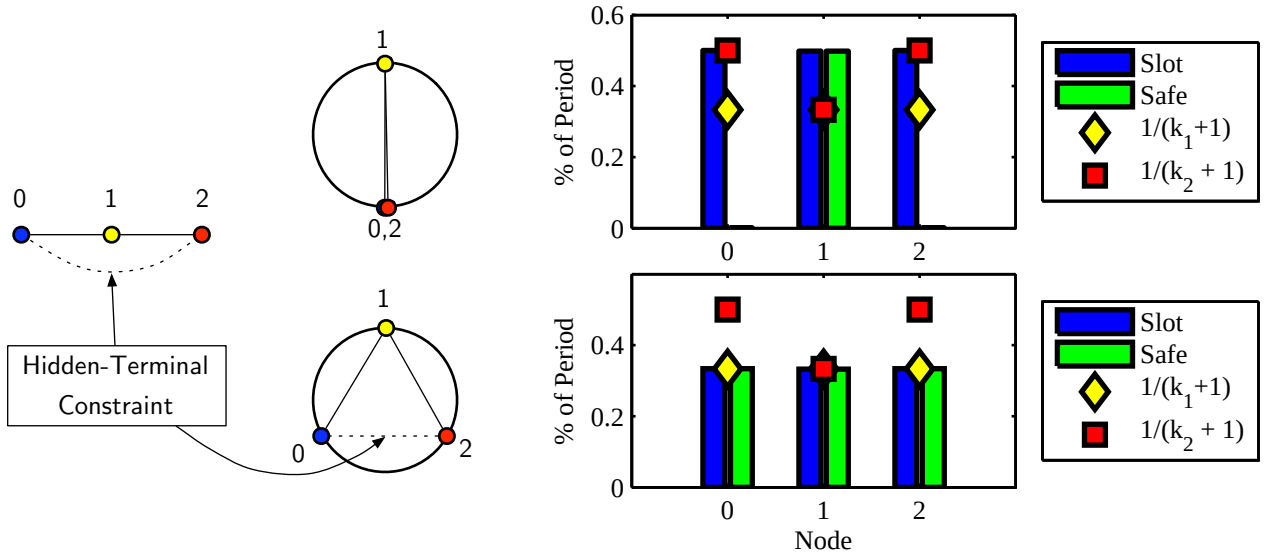

Figure 11. Left: The constraint topology for P_3. The solid lines represent the neighbor constraints and the dashed line represents the hidden-terminal constraint. The rings depict the final desynchronized configurations (top: communication topology, bottom: constraint topology). The top graph shows that nodes 0 and 1 achieve slot sizes of 1/2, but cannot use them because of the hidden terminal problem. The bottom graph shows a conservative and fair allocation of entirely safe slots (no hidden terminal problems will result). (k_1 = degree, k_2 = two-hop degree.)

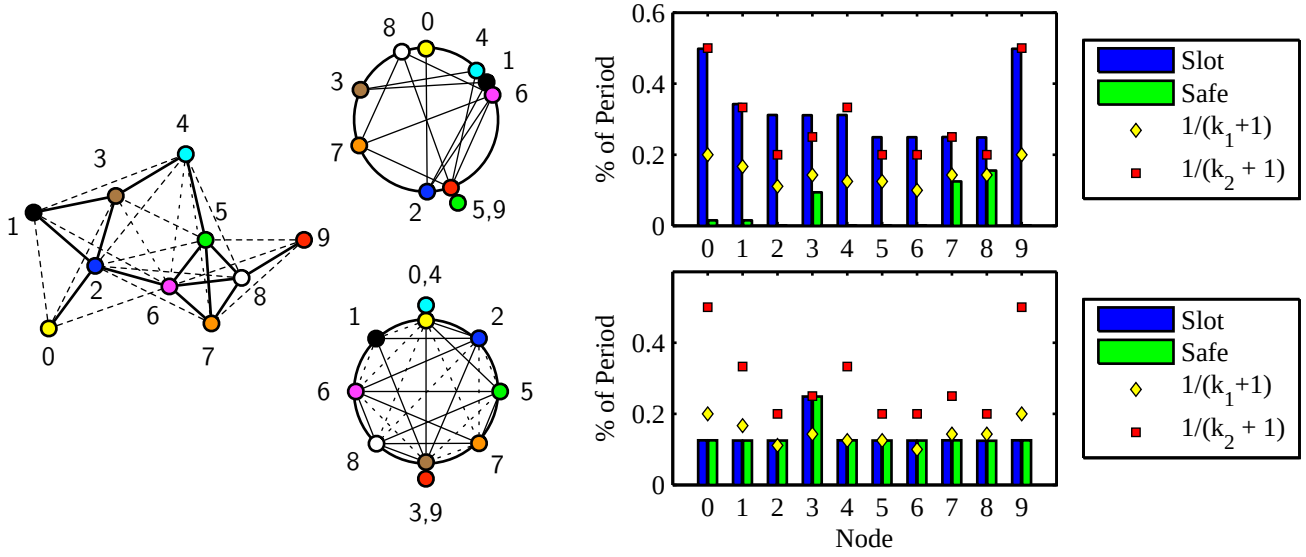

Figure 12. Left: The constraint topology for a wireless network. The solid lines represent the neighbor constraints and the dashed lines represent the hidden-terminal constraints. The rings depict the final desynchronized configurations (top: communication topology, bottom: constraint topology). The top graph shows that the nodes achieve large slot sizes, but only three nodes can use them (none of the others are safe). The bottom graph shows a conservative and fair allocation of entirely safe slots (no hidden terminal problems will result). (k_1 = degree, k_2 = two-hop degree.)

bors along with their own firings. This adds only minimal complexity to the algorithm, but it does increase the message size from a single pulse to $O(\Delta \log \Delta)$ bits, where Δ is the maximum degree of all vertices in the communication graph. Furthermore, because the forwarded firings require more time to propagate, it seems that α must be much smaller in order to achieve convergence. In the future we plan to implement this 2-hop DESYNC algorithm and analyze its performance in more detail.

4 Related Work

We have shown that it is possible for desynchronized phases to be loosely interpreted as a graph coloring. Here we briefly mention other variants of graph coloring that may also be connected to the study of desynchronization.

4.1 Fractional Graph Theory

A recent variation of graph coloring attempts to reduce the total number of colors required by associating fractional colors with fractional slot sizes [6]. In other words, instead of coloring a graph with 3 colors, it might be able to be colored using 5 "half-colors," corresponding to an effective coloring of 2.5. In fractional coloring, instead of assigning each node 1 full color, each node is assigned 2 half-colors that correspond to two half-slots. As before, neighboring nodes cannot have any half-colors in common with their neighbors.

While this technique can use fewer effective colors, mapping the fractional coloring to a schedule still requires synchronized and slotted time to be useful as a resource-scheduling algorithm. Furthermore, finding the minimal number of fractional colors required to color a graph is still NP-Hard [6]. We are also unaware of any distributed or self-organizing algorithm that can fractionally color a graph.

4.2 Continuous graph coloring

Similar to the use of real-valued phases in desynchronization, relaxation techniques have attempted to use real values instead of discrete colors. However, instead of using variable-sized slots, these techniques focus on "rounding" the real-valued colors back to discrete colors.

Wah Wu has even explored the use of simulating coupled circuits that can be interpreted as continuously coupled oscillators [7]. This is similar to our work of pulse-coupled oscillators; however, once the oscillators achieved a frequency-locked state, the phases were again mapped to discrete colors.

Relaxation techniques aren't always appropriate as approximations to constraint problems since it is not clear how to map the real values to discrete values. In certain instances, relaxed solutions cannot even be mapped to a reasonably close discrete counterpart, making this technique insufficient in those cases.

5 Conclusions

This paper gives a first step towards understanding desynchronization on multi-hop topologies. Several examples and experimental results suggest that there always exists a desynchronized configuration for any topology. And the self-organizing algorithm DESYNC experimentally converges to these configurations. In the future we hope to solve some of the remaining open problems such as theoretically proving that DESYNC solves desynchronization for all graphs and implementing a 2-hop version of DESYNC for wireless sensor networks.

References

[1] J. Degesys, I. Rose, A. Patel, and R. Nagpal. DESYNC: Self-organizing desynchronization and TDMA on wireless sensor networks. In *Proceedings of the sixth international conference on information processing in sensor networks (IPSN)*, April 2007.

[2] A. Giusti, A. L. Murphy, and G. Pietro Picco. Decentralized scattering of wake-up times in wireless sensor networks. In *Proceedings of the fourth European conference on wireless sensor networks*, pages 245–260, 2007.

[3] T. R. Jensen and B. Toft. *Graph Coloring Problems*. Wiley-Interscience Series in Discrete Mathematics and Optimization. wiley, 1995.

[4] A. Patel and J. Degesys. Desynchronization: The theory of self-organizing algorithms for round-robin scheduling. In *Proceedings of the first IEEE International Conference on self-adapting and self-organizing systems (SASO)*, July 2007.

[5] I. Rhee, A. Warrier, J. Min, and L. Xu. DRAND: Distributed randomized TDMA scheduling for wireless ad-hoc networks. In *MobiHoc*, 2006.

[6] E. R. Scheinerman and H. Ullman, Daniel. *Fractional Graph Theory: A Rational Approach to the Theory of Graphs*. Wiley-Interscience, 1997.

[7] C. Wah Wu. Graph coloring via synchronization of coupled oscillators. *IEEE Transactions on Circuits and Systems—Part 1: Regular Papers*, 45(9):974–8, September 1998.

[8] G. Werner-Allen, J. Johnson, M. Ruiz, J. Lees, and M. Welsh. Monitoring volcanic eruptions with a wireless sensor network. In *Proc. European Workshop on Wireless Sensor Networks (EWSN)*, Jan 2005.