

Merging Ultra-low Duty Cycle Networks

Matthew Dobson(m.c.dobson@vu.nl), Spyros Voulgaris(spyros@cs.vu.nl), Maarten van Steen(steen@cs.vu.nl)
Department of Computer Science, Vrije Universiteit, Amsterdam, Netherlands

Abstract—Energy is the scarcest resource in ad-hoc wireless networks, particularly in wireless sensor networks requiring a long lifetime. Intermittently switching the radio on and off is widely adopted as the most effective way to keep energy consumption low. This, however, prevents the very goal of communication, unless nodes switch their radios on at *synchronized* intervals, a rather nontrivial coordination task.

In this paper we address the problem of synchronizing node radios to a single universal schedule in very large scale wireless ad-hoc networks. More specifically, we focus on how independently synchronized clusters of nodes can detect each other and merge to a common radio schedule. Our main contributions consist in identifying the fundamental subproblems that govern cluster merging, providing a detailed comparison of the respective policies and their combinations, and supporting them by extensive simulation. Energy consumption, convergence speed, and network scalability have been the driving factors in our evaluation. The proposed policies are extensively tested in networks of up to 4,096 nodes. Our work is based on the GMAC protocol, a gossip-based MAC protocol for wireless ad-hoc networks.

I. INTRODUCTION

Recent advances in electronics and embedded systems have made wireless devices become smaller, lighter, less intrusive, and significantly cheaper: a commodity. This enables the deployment of increasingly larger collections of such devices for a multitude of applications, mainly for the collection of observed data (sensor networks). There is no indication of a slow down in this trend. Quite on the contrary, we anticipate wireless sensor networks consisting of tens of thousands of nodes to be common in the near future.

Of major concern to wireless networks is their *lifetime duration*, and *energy* is the main factor determining it. Decreasing the energy footprint of a wireless device boosts its lifetime in a reversely proportional way. It comes as no surprise that most research aiming at prolonging the lifetime of wireless networks focuses on limiting the radio operation of their devices. Indeed, the radio circuit of some sensor devices are measured to consume three orders of magnitude more power than the rest of the hardware (CPU, memory, etc.), either when the radio is in transmitting or receiving mode.

The main way to limit the operation of the radio is to limit the time for which the radio circuitry is switched on. This implies intermittently switching the radio on and off. The periods during which a node's radio is on or off are known as its *active period* and *inactive period*, respectively. The fraction of the time that a node's radio is on, is known as the *duty cycle*. That is,

$$\text{duty cycle} = \frac{T_{\text{active}}}{T_{\text{active}} + T_{\text{inactive}}} = \frac{T_{\text{active}}}{T}$$

For example, a node that is active for 100ms every second has a duty cycle of $\frac{100ms}{1000ms} = 10\%$.

Lifetime predictability is an equally important property of a wireless network for certain applications. Prolonging the network's life in a best effort manner is not enough for some applications that additionally require a reasonable estimation of the network's lifetime. To guarantee lifetime predictability, the use of energy should be constant and independent of operation-specific conditions, such as coincidentally high traffic or unexpected topology changes.

The requirements of *long* and *predictable* lifetime duration have led to the GMAC¹ family of protocols. In GMAC, nodes use a very small duty cycle (in the order of 1%), and broadcast messages at fixed intervals, following a gossip-based communication model.

It is clear that to enable communication between two or more nodes, their active periods should be—at least partially—overlapping. In fact, to fully utilize the energy nodes spend on their radio circuits, their active periods should be synchronized as *accurately* as possible, to maximize the shared communication window. Synchronization of active periods in ad-hoc wireless networks is a nontrivial problem, notably due to the lack of a central coordinator and the inherently restrained nature of such devices. When confronted with the additional requirement of fixed-rate use of energy, it becomes a far more challenging problem, as solutions that asymmetrically put more burden either on the sender or the receiver, are ruled out.

Maintaining the active periods of nodes in an ad-hoc network synchronized to a single schedule is decomposed in two orthogonal subproblems:

- First, clusters of nodes synchronized independently to non-overlapping schedules, should detect the existence of each other and merge to a single, universal schedule.
- Second, once a set of nodes is synchronized to a common schedule, corrective actions should be continuously taken to alleviate the tendency of nodes to drift apart due to different clock drifts.

We have dealt with the latter subproblem in [1], which turns out to be addressed by an extensive number of researchers, at various different forms and scenarios, as discussed in Section II.

In this paper we address the former subproblem. In particular, we are interested in sets potentially consisting of thousands of nodes. Despite its key importance in forming large ad-hoc

¹GMAC is protected by US Patent Application 12/215,040. GMAC is available free of charge for academic use.

networks with a single synchronized schedule, this problem has not been addressed extensively by the research community.

Although our solution is presented in the context of GMAC, the methodologies, principles, and algorithms we propose can be generalized to virtually any MAC protocol with a very low duty cycle.

II. RELATED WORK

There are two MAC-level protocols that are particularly relevant for our discussion. S-MAC [2], one of the main representatives of *slotted* access protocols, divides time in fixed-length slots of 1-3s and uses a 300ms active period, during which nodes compete for the channel using carrier-sensing to avoid collisions. T-MAC [3] improves upon S-MAC by adding adaptivity to traffic. Active nodes time out if they hear no traffic for 15ms, drastically reducing energy use in idle networks.

In both S-MAC and T-MAC, when a new node joins it listens for at least the duration of a whole slot to detect the presence of other nodes. If other nodes are present, it follows their schedule. Otherwise it picks an arbitrary schedule of its own. When multiple schedules are detected, a node follows them all, acting as a bridge between independently synchronized clusters. This, however, imposes on bridge nodes an energy cost that is a multiple of the cost for nodes following a single schedule, which is against our goal of fixed energy consumption and predictable lifetime.

Most importantly, both protocols ignore the fact that in the course of time, notably in large networks where maintaining synchronization across a long diameter is nontrivial, such a policy will eventually lead to the coexistence of a number of diverse schedules, multiplying the amount of energy used, while at the same time hindering the operation of broadcast-based communication protocols. Although this issue does not arise in small-diameter and short-lived networks, in networks of the size, longevity, and mobility we target at it constitutes a major shortcoming.

SCP-MAC [4] is a further optimization of the aforementioned protocols, lowering duty-cycles to as low as 0.3% by allowing channel polling at very short, scheduled intervals. Although SCP-MAC is significantly more sensitive to a tight synchronization than S-MAC and T-MAC, the issue of merging independently synchronized “virtual clusters” to a common schedule is completely overlooked in SCP-MAC, implicitly assuming a set of nodes that is and remains tightly synchronized.

In [5], Liu et al. describe a method for merging clusters in multi-hop 802.11 ad-hoc networks, in contrast to the standard solution of bridging the clusters. Their method is based exclusively on the passive listening method (extensively described in Section IV). There are no details on the merge process itself, presumably nodes simply ‘jump’ to their new schedule during the merge.

Mank et al. present Mobile LMAC in [6] and [7], removing assumptions about static topologies and using gateway nodes to bootstrap synchronization. The proposed merge protocol

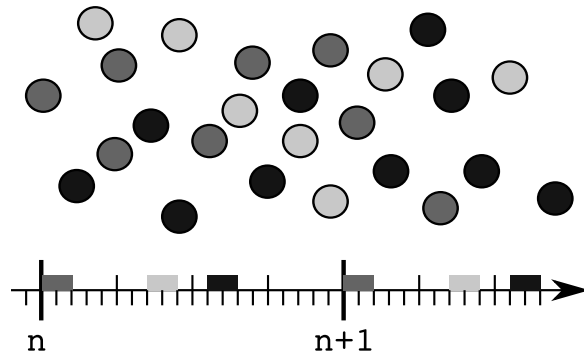


Fig. 1: Graphical depiction of the physically connected but separately synchronized clusters of nodes

comes close to ours with respect to the part making a decision on which cluster to prevail. However, their evaluation is limited to networks of up to nine sensors, which is too limited to draw any conclusion with respect to scalability. Additionally, the Mobile LMAC protocol focuses on enabling nodes to achieve a high throughput channel even in the case of high network load, contrary to GMAC which is designed for constant-rate gossiping between nodes.

In [8], Cidon and Sidi propose an algorithm that allows a multihop network of N nodes to dynamically agree on a conflict-free TDMA schedule. However, it requires $O(N)$ slots per frame, which renders it inappropriate for the scenarios we are targeting.

In [9], Arumugam and Kulkarni present an algorithm that deterministically establishes a TDMA schedule by a gateway node circulating a token. However, no attention is paid to joining clusters and keeping them synchronized, as nodes are assumed to be de facto synchronized.

The same authors propose SS-TDMA [10], a self-stabilizing MAC protocol for sensor networks. It assigns slots deterministically based on (known) locations in a grid topology and is bootstrapped from a gateway node that also acts as a sink. The protocol is tailored to TDMA schedules for gossiping, however no duty-cycling or other energy-awareness is discussed.

The issue of clock synchronization in the face of clock drifts is addressed by Tjoa et al. in SMART [11]. Although this paper is an inspiration for the clock synchronization algorithm adopted in GMAC, it does not deal with the orthogonal problem of joining clusters with non-overlapping schedules, neither does it consider duty cycling.

Finally, Pussente and Barbosa address the clock synchronization problem too in [12]. Like the previous paper, this paper focuses on clock synchronization alone, not dealing with duty cycling or merging of different clusters.

Concluding, although a multitude of MAC layer protocols have been designed for a plethora of different target scenarios, to the best of our knowledge no work exists that exhaustively addresses the issue of dynamically merging clusters independently synchronized to non-overlapping schedules.

III. DECENTRALIZED CLUSTER MERGING

The duty cycle-based operation of the nodes makes the synchronization of the active periods of their frames essential. Nodes whose active periods do not overlap cannot communicate with each other, effectively partitioning the whole network into separate sub-networks (see Figure 1). In this paper, we focus on one aspect of that synchronization, namely the merging of these separate sub-networks, or *clusters*, into a single connected network, with all nodes sharing the same active period.² When all clusters have been merged together, we say that the nodes have *converged* to a single cluster. The problem of convergence can be broken-down into three sub-problems: *detection*, *decision* and *notification*.

A. Detection

Before the clusters can be merged, they must first become aware of each other. We distinguish two methods of detection: **passive**, where nodes *listen* during the inactive portion of their duty cycle to detect messages from other nodes, and **active**, where nodes *broadcast* a join message during the inactive portion of their duty cycle allowing other nodes using a *different* active period to detect and join the sending node's cluster.

Passive detection offers a trade-off of increased energy consumption for faster detection. For example, a node could detect any other node in its range if it listened to the entire inactive portion of its frame. However, this obviously defeats the purpose of duty cycling, and would rapidly deplete the node's battery. We could apply the duty cycle method and instruct nodes to listen to some percentage, p_l , of the inactive period³, reducing energy consumption but also effectiveness.

The effectiveness of active detection can be affected by energy spent (i.e., sending more messages), but is mainly determined by the duty cycle of the network, τ . This is because the probability p_d of a detection event, that is, the probability that a message transmitted during one cluster's inactive period will be received during another cluster's active period (ignoring collisions), is equal to:

$$p_d = \frac{T_{active}}{T_{inactive}}$$

Based on the definition of the duty cycle, τ , we have:

$$\tau = \frac{T_{active}}{T_{active} + T_{inactive}} \Rightarrow$$

$$T_{inactive} = \frac{(1 - \tau) \times T_{active}}{\tau}$$

By substituting back in the formula for the detection probability we get:

$$p_d = \frac{\tau}{1 - \tau}$$

²As we showed in [1], GMAC's median algorithm is capable of maintaining tight synchronization within clusters.

³This can be implemented as listening for an additional $p_l \times T_{inactive}$ seconds every frame or by listening to the entire frame (an additional $T_{inactive}$ seconds) with probability p_l . We chose to implement the latter method.

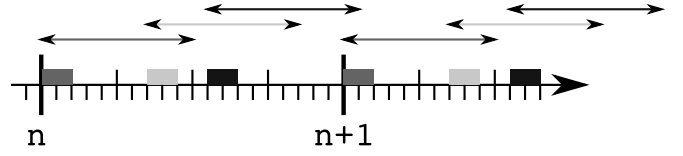


Fig. 2: Graphical representation of the cycle problem in GMAC's decision mechanism

If $T_{active} > T_{inactive}$, then $\tau > 50\%$, so nodes are active for more than half of each frame. This implies that all nodes' active periods must overlap, so separate clusters cannot form. For this reason, we do not consider duty cycles greater than 50%. For duty cycles less than 50%, as the duty cycle increases towards 50%, the probability of detecting another network increases rapidly to 100%. Unfortunately the inverse also holds, indicating that ultra-low duty cycles will lead to ultra-low detection rates.

Active detection does have a decided advantage over passive detection. A whole *group* of nodes may detect the existence of another cluster at once, by a *single* message broadcast by one node of that cluster, provided that that message hits the active period of the group of nodes. In the case of passive detection, each node would have to *individually* detect the presence of the foreign cluster, by paying the price of keeping the radio in listening mode during their inactive period.

The disadvantage, however, of active detection is an increased chance of collisions, as the join messages sent from one cluster may collide with each other, or with application messages belonging to a different cluster. Both active and passive detection schemes will be heavily affected by the density of the network.

GMAC is designed to use active detection, with each node sending one join message during its inactive period.

B. Decision

Regardless of how detection happens, once a node from cluster *A* is aware of another cluster *B*, it must decide whether it should merge into *B* or if it should stay in *A*. Nodes cannot merge unconditionally, because otherwise the whole network may never converge as nodes merge back and forth between multiple clusters. In order to fully utilize a network, all nodes must be able to, eventually, communicate with all other participating nodes. This makes convergence an absolute requirement, so we should try to minimize the amount of time and energy spent on it.

The decision algorithm should be a relation \succ that provides a *total ordering* of the set of existing clusters. That is, the decision relation $A \succ B$ determines whether cluster *A* is superior to cluster *B*. Thus, when a node in *A* receives a join message from a node in *B*, it should join cluster *B* if and only if $B \succ A$. The relation \succ should provide the following properties:

- 1) antisymmetric: if $A \succ B$ and $B \succ A$ then $A = B$
- 2) transitive: if $A \succ B$ and $B \succ C$ then $A \succ C$
- 3) total: $A \succ B$ or $B \succ A$

If these properties are provided, then a connected network will eventually converge as long as nodes eventually detect all other nodes in their range. This is assured because nodes will always merge from an *inferior* cluster into the *superior* cluster.

GMAC uses a heuristic mechanism to decide when a node should join a newly discovered cluster: if the join message was sent during the first half of the sender’s frame, then it is accepted as valid, otherwise it is discarded. This is meant to provide antisymmetry (Property 1), since for any two clusters only one of them can send a join message in the first half of its inactive period that the other can receive during its active period. This relation also provides totality (Property 3), because the two clusters cannot be desynchronized by more than half a frame, implying the active period of one overlaps with the first half of the other’s frame. However, this relation does not provide transitivity (Property 2). If more than two clusters exist in each other’s range, there can be ‘cycles’, i.e., where nodes can merge from *A* to *B* to *C* and then back to *A*. In the best case, one (or more) of the clusters in the cycle can be eliminated if the others can get all of its nodes to merge.

For example, if clusters *B* and *C* could get all of the nodes in *A* to join their respective clusters before *A* can get any nodes from *C* to join it, then the cycle would collapse. In the worst case, these cycles can persist forever, leading to a network that never converges. A visual example of this effect is seen in Figure 2. Nodes in cluster *B* will accept join messages from cluster *C*, because the first half of the cluster *C*’s frame overlaps with the active period of cluster *B*. Similarly, nodes in cluster *A* will only respond to join messages from cluster *B*, and so on. Contrarily, cluster *B*’s nodes will ignore join messages from cluster *A*’s nodes for the same reason. In this example, the three clusters form a cycle, allowing for a node to merge from *A* to *B* to *C*, and then back to *A*, ad infinitum. We will provide a solution to this problem below.

C. Notification

Once a node has decided that it must merge into a new cluster, it should notify its own cluster of the merge. Though not strictly necessary, notification of the decision to merge one cluster into another can be rapidly propagated through the already synchronized inferior cluster, saving the need for repeated detections of the same superior cluster.

GMAC does not use any notification of discovered clusters. Nodes that decide to join a different cluster just silently merge. That is, they leave their old cluster by adjusting the length of their current frame to align their next frame with their new cluster. In situations with many clusters, this can lead to isolated nodes as neighbors discover better clusters and leave them behind.

IV. PROPOSED POLICIES

In our previous work [1], we demonstrated that the cluster merge behavior of GMAC was sufficient to ensure convergence for small networks, but struggled to consistently converge larger networks. The main contribution of this work

is a thorough analysis of various methods of merging large networks composed of multiple desynchronized clusters. In this section, we discuss some proposed improvements to GMAC’s current cluster merging mechanism. In the following section, we will analyze several distinct combinations of these improvements.

A. Detection

In addition to active detection, provided by default in GMAC, we also implemented passive detection functionality, to allow for a comparison of the effectiveness of the two methods. In our implementation of passive detection a node listens to the whole inactive portion of its frame with probability p_l .

We also implemented a second version of passive detection: Rather than immediately merging into a newly discovered cluster, a node can listen to an entire frame first, to try to discover an even *better* cluster (see below what makes a cluster “better”). In very large networks that may have many clusters before finally converging, it may prove effective to skip joining “second-best” clusters within range, if there are better ones. We call this technique *listen-before-merge*.

B. Decision

Ideally, we would like the cluster with fewer nodes to always join a cluster with more nodes, to minimize disruption to the network. However, computing such network metrics in a decentralized fashion is a difficult problem. Even if our nodes all knew the exact size of their cluster, we would still need a method of breaking ties between clusters of equal size. Such a tie-breaker method can also serve as the primary criteria for the cluster merge decision. This may lead to sub-optimal merge operations, forcing many nodes to resynchronize to match a few. Convergence is more important than optimality, particularly because in a stable network, merge operations can be assumed to be infrequent.

We propose to solve the convergence problem using *cluster IDs*. A cluster id is simply an identifier used by all nodes that share a common active period, to identify their particular cluster. We assume that all nodes have a unique identifier, and nodes initially use their own identifier as the id for their singleton cluster. If a node hears a better cluster id during its active period, it should discard its old id and adopt the newly discovered one. When a node *detects* a different cluster (either by hearing a join message during its active period, or by overhearing an application message while listening to its inactive period), it can simply compare its own cluster’s id to that of the other cluster. If its id is higher, it is already in the superior cluster and will ignore the message. However, if the other cluster’s id is higher, the node can *decide* that it should merge its inferior cluster into the other and react accordingly. By assuring that the nodes in a cluster with a higher id never merge into a cluster with a lower id, we can eliminate the cycling problem in GMAC’s decision mechanism. Eventually all other clusters will merge into the cluster with the best id.

C. Notification

We have added a *merge* field to the header of application messages. This allows a node to notify its neighbors when it detects a superior cluster. After discovering a cluster with a better id, a node can record the time difference, or offset, between its own cluster and the superior one. Then, rather than immediately merging into the new cluster, it can stay synchronized to its current cluster for one more frame, in order to communicate to its old neighbors one more time and inform them about the new cluster. By sending this merge offset along with its message in the following frame, its current neighbors can be made aware of both the existence and the offset of this superior cluster *without* the need to detect it on their own. This notification should greatly reduce the time and energy spent on detection, particularly at low duty cycles which reduce the probability of detecting other clusters.

V. EXPERIMENTAL SETUP

We conduct our simulations using the MiXiM extensions to the OMNET++ simulation environment. For more detail about our simulator, GMAC, and the MyriaNed nodes we simulate, please see [1].

A. Clocks

We designed our own OMNET++ modules to represent the clocks found in our sensor nodes. OMNET keeps track of the global simulation time, T_{sim} , while an individual node x computes its own local time, T_x . A node bases this on its own clock's frequency multiplier (F_x) and phase offset (P_x), provided as OMNET simulation parameters. Thus, x can compute $T_x = (T_{sim} \times F_x) + P_x$. A node's phase offset determines the length of time between the global start of the simulation and the start of that particular node. The frequency multiplier determines how much faster or slower than simulation time a node's clock runs. Unless otherwise specified, all clocks use a random frequency multiplier $0.99998 < F_x < 1.00002$, i.e., ± 20 parts per million.

B. GMAC Configurations

In order to facilitate discussion of GMAC's behavior with various improvements switched on or off, we will analyze several specific combinations, called *configurations*.

- **<Active>** This is the default GMAC behavior, as described in Section III.
- **<Active+Ids>** The same as **<Active>**, but using cluster ids in order to make consistent merge decisions.
- **<Passive+Ids>** Purely passive detection with $p_l = 0.4\%$, using cluster ids. We explain our choice for the value 0.4% below.
- **<Active+Ids+MergeMsgs>** The same as **<Active+Ids>**, but nodes do not immediately join newly discovered clusters, rather they wait one frame in order to send merge messages.
- **<Active+Ids+Listen>** The same as **<Active+Ids>**, but nodes do not immediately join newly discovered clusters, rather they listen for a whole frame in order to discover

TABLE I: Networks Investigated

Nodes	Dimensions	Spacing
64	640m × 640m	80m Matrix
256	1280m × 1280m	
1024	2560m × 2560m	
4096	5120m × 5120m	

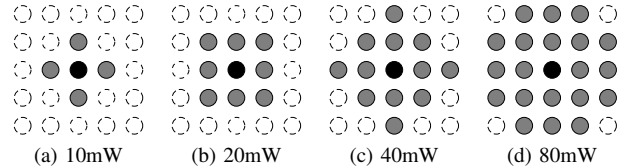


Fig. 3: Graphical representation of the four simulated transmit ranges

the best cluster in range (which we also refer to as *listen-before-merge*).

- **<Active+Ids+MergeMsgs+Listen>** This configuration uses active detection, cluster ids, and uses both listen-before-merge and merge messages.

GMAC uses 8 active slots, and with a default half-second frame length, a full frame has 584 slots. In terms of slots, $T = 584$, $T_{active} = 8$ and $T_{inactive} = 576$. This gives GMAC a duty cycle of $\tau = \frac{T_{active}}{T} = \frac{8}{584} = 1.37\%$, and an active detection probability of $p_d = \frac{T_{active}}{T_{inactive}} = \frac{8}{576} = 1.39\%$.

In order to compare active and passive detection, we would like to spend approximately the same amount of energy in both cases. Sending a join message costs an amount of energy equal to the sum of the energy required to turn on the node's radio, broadcast a message, and turn off the node's radio again. On our hardware, this costs about the same as two active receive slots, so we would like to listen to two slots per frame. With the default 584 slots per frame, a node should be listening to a whole frame every 200-300 frames on average. For this reason we use a default $p_l = \frac{1}{250} = 0.4\%$, meaning a node will randomly listen to an entire frame once every 250 rounds.

C. Topology

To better assess the strengths and weaknesses of the various configurations, we investigate the effect of topology on cluster merging. In particular, we look at network size and node density. In all of our experiments the nodes are deployed in a regular *matrix* pattern. N^2 nodes are deployed in an $N \times N$ grid, with rows (and columns) placed 80m apart (see Table I). It is important that the networks we examine are connected, because otherwise complete synchronization would be impossible. Though not the most representative of real-world deployments, matrix topologies allow us to directly observe the effects of node density with a regular topology.

We set the transmit power for all nodes in the network on a per-run basis in order to vary the *density* of a given topology. By increasing a simulated node's transmit power, the simulator increases the node's transmit range. This effectively decreases the diameter of the network and increases its density. We have

chosen the transmit power values based on our grid spacing of 80 meters, and unless otherwise specified, use $20mW$ as our default setting. In Fig. 3 we show a group of nodes spaced $80m$ apart, depicting the four transmit power level ranges from the perspective of the sender (black) and the potential receivers (gray). This parameter strongly influences the connectedness of a given topology.

D. Measurements

Each simulated node x logs the global simulation time $T_{x,i}$ at the beginning of each new round i . Using this data, we can see not only which nodes are synchronized to which (i.e., whether their active periods overlap), but how tightly they are synchronized (i.e., how much their active periods overlap).

We know that nodes who are desynchronized by more than the duration of one active period cannot communicate. We consider groups of nodes whose reported start times for round i differ by less than some ϵ to be part of a synchronized cluster. We look at the reported times in increasing order and count the clusters. In our measurements we consider a cluster to be a group of nodes for which, if sorted by the active period starting times, consecutive nodes have a relative offset of at most $2ms$ (about 65 ticks, or just over 2 slots). For each cluster, we compute its size ($1..|Nodes|$).

In order to evaluate how tightly synchronized the nodes are, we compute the standard deviation of reported start times for round i across all nodes. By looking at how the standard deviation changes as the simulated run progresses, we can see whether the synchronization mechanism is able to reach and/or maintain tight temporal-coupling of the nodes. Given that we are simulating 32 kHz clocks, one timer tick is approximately $30 \mu s$ and one slot is approximately $850 \mu s$ (28 ticks). Therefore, we can consider an entire network to be synchronized when the standard deviation of start times drops below $1000 \mu s$, or $1 ms$.

E. Scenarios

We utilize different *scenarios* in order to evaluate different aspects of the merge behavior.

1) *Asynchronous Start*: All nodes start up at a random time $1s \leq t_{start} \leq 15s$ in the CATCHING state. In this state, nodes know they are unsynchronized and search for a cluster to join. Initially they will continuously listen for a message for a random initial period t_{catch} , lasting between one and two frames. If they do not catch a cluster (by hearing a message) before the end of this extended frame, they then broadcast a single **HELLO** message. After sending their message, they switch back to continuous listening mode and remain in that mode. When a CATCHING node hears a message, it will enter the CAUGHT state, and try to synchronize its next frame with the node (and cluster) that it heard. Here, nodes that use the *listen-before-merge* option will continue listening until the end of their frame, rather than going to sleep. Once the node has performed its frame-length adjustment, it will enter the NORMAL state, and will assume that it is synchronized. When in a normal, synchronized state, nodes will execute the

GMAC duty cycle of eight active TDMA slots followed by a long inactive period, dependent upon the length of the frame. Note that nodes that use cluster ids will ignore them while CATCHING, because otherwise a node with a high id may stay isolated and silent, ignoring all its neighbors that happen to have lower ids. Once a single node has found an initial cluster and synchronized with it, it will respect the ordering of cluster ids from then on.

2) *Singleton*: We simulate two different variations of this scenario: a singleton cluster with an *inferior* id detecting and merging into the established superior cluster, and an existing cluster detecting and merging into a singleton cluster with a *superior* id. We call these scenarios *SingletonWorst* and *SingletonBest*, respectively.

This is by far the simplest set of scenarios that we investigate. That is particularly so in the *SingletonWorst* case, where the desynchronized node has an inferior cluster id. In this case, the network will converge after that single node detects the other cluster, and merges into it. In the *best* case, the single desynchronized node has a superior id, and must get all other nodes to merge into its cluster. In both variations of this scenario, the isolated node is located in the top-left corner of the grid. We have chosen this location because it maximizes the distance (hops) that the synchronization information must travel to reach all nodes in the network.

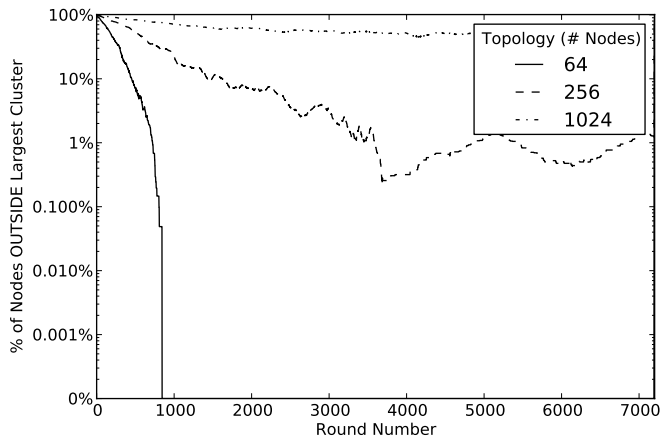
3) *Cluster Merge*: This scenario is designed to give us insight into the behavior of cluster merging in the case of multiple synchronized clusters. Here we only look at the 32×32 (1024-node) topology. The sixteen columns of nodes on the right-hand side of the grid begin as one synchronized cluster, and the fourteen columns of nodes on the left-hand side begin as another. For the first five seconds of the simulation, the rightmost two columns of the left-half of the grid are inactive. At $T_{sim} = 5s$, these 64 nodes start up as a third synchronized cluster. The left cluster has id 1, the middle cluster has id 2, and the right cluster has id 3. This final scenario is more complex than the singleton one, but more straight-forward than the asynchronous start.

VI. EVALUATION

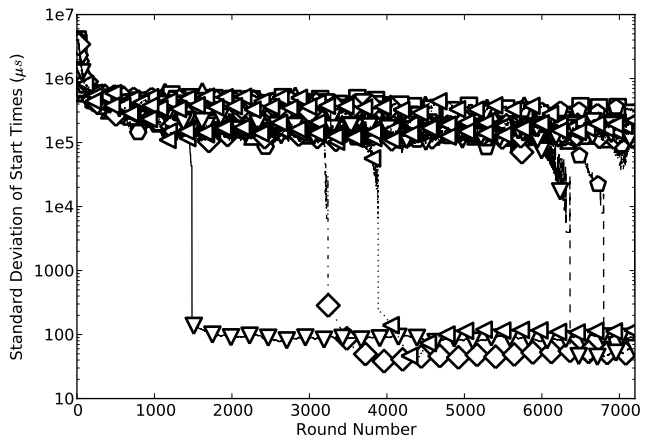
We look at three different scenarios in order to evaluate the three different aspects of merge behavior. First, to examine the *decision* aspect of merging, we use *AsynchronousStart*. In the *Singleton* scenarios, we explore the *detection* aspect of cluster merging. Finally, using the *ClusterMerge* scenario, we look more generally at two established clusters merging. For this scenario, we can investigate how the size, topology and distribution of the clusters affects the merge behavior, as well as examine the effects of our *notification* improvements.

A. Decision

We begin with the *AsynchronousStart* scenario for two reasons. First, this is the scenario that initially pointed at failings in GMAC's synchronization mechanisms, so it makes sense to reproduce those results here. Second, by demonstrating the performance of all our test configurations in the most



(a) Performance of $\langle Active \rangle$ for increasing network sizes



(b) Per-round standard deviation of start times for 32 runs of $\langle Active \rangle$, 1024-node grid

Fig. 4: Cluster merging using the $\langle Active \rangle$ configuration

demanding circumstances, we can focus our analysis on the best few.

In Figure 4 we see the performance of the $\langle Active \rangle$ GMAC configuration. Figure 4a shows a plot of the standard deviation of start times as a function of the round number, averaged across 32 runs. We see that GMAC’s $\langle Active \rangle$ configuration works acceptably in the small 64-node topology, converging all nodes to a single cluster in an average of about 1000 rounds (about 8 minutes). However, it does not consistently synchronize the 256-node and 1024-node test cases. In the 256-node networks, the majority of all nodes synchronize to the best cluster, but not all runs synchronize. In Figure 4b, we see a composite plot showing that only six of the thirty-two 1024-node runs converged to a completely synchronized network, and in those cases it often took almost a full simulated hour.

To examine the cause of the $\langle Active \rangle$ configuration’s difficulties, we look at the results of some individual runs that clearly show the problem. The three graphs on the left side of Figure 5 show the difference in μs between each node’s start time and the average start time for the round on the x-axis. Those on the right show the percentage of nodes *outside* the largest cluster, calculated as described in Section V-D. Figure 5a shows a 256-node run that properly converges to a single cluster around round 2500. The individual clusters can be seen previous to that point as clusters of points, converging to bounds of about $\pm 100\mu s$. Figure 5b is another view of the same data, showing the nodes outside the largest cluster dropping to zero. The results of 1024-node runs show similar initial behavior, but generally fail to converge. For an example of a non-converging run, see Figures 5c and 5d. These results demonstrate the problem of cyclic ordering of clusters described in Section III-B. The $\langle Active \rangle$ merge protocol can only consistently synchronize two clusters. If there are more than two clusters a cycle may exist, preventing convergence

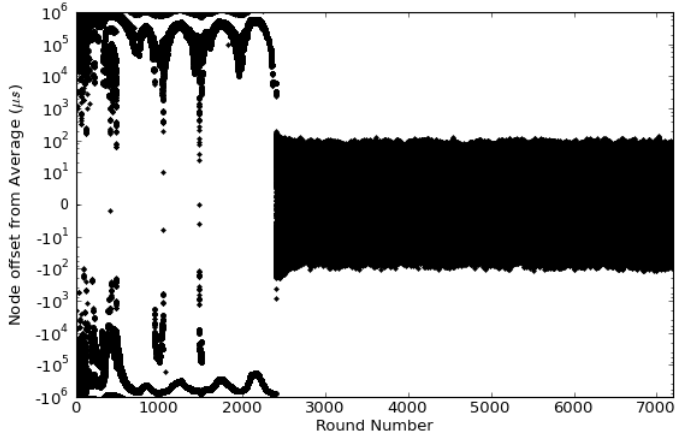
for an arbitrarily long time. The severity of the problem is dependent on the size of the network, and will make it unsuitable for very large networks.

In order to solve this *decision* problem, we have proposed using cluster ids, described in Section IV-B. In Figures 5e and 5f, we show the performance of the $\langle Active+Ids \rangle$ configuration for a representative 1024-node run. The general merge mechanism is the same as $\langle Active \rangle$, but nodes use the cluster ids to reliably decide whether to join a discovered cluster, rather than arbitrary timing heuristics. When compared to the results for the $\langle Active \rangle$ configuration, the ids clearly provide for superior performance. In Figure 6a we show the average performance of the $\langle Active+Ids \rangle$ configuration, and we can see that it is able to consistently synchronize a 1024-node network in a little over 1000 rounds. This is the same number of rounds it takes the $\langle Active \rangle$ configuration to synchronize a 64-node network. Based on these findings we can eliminate the $\langle Active \rangle$ configuration from further study, and all of the following results utilize cluster IDs.

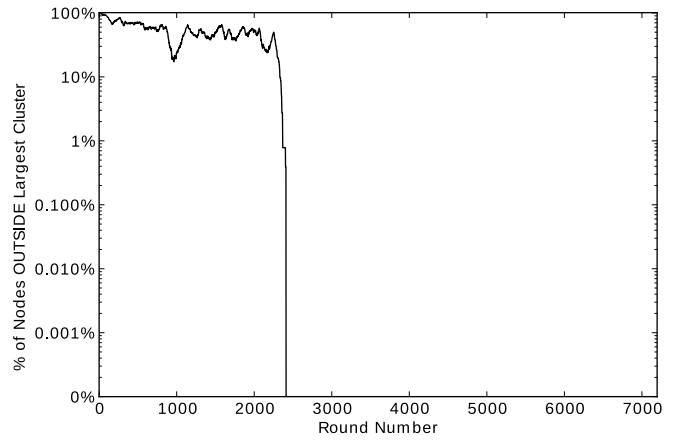
B. Detection

GMAC currently uses an active detection mechanism, as described earlier. Here we will examine the performance of the new passive detection configurations we have designed, i.e. $\langle Passive+Ids \rangle$, in order to see whether they are superior to the existing active mechanism. We begin with more results from the *AsynchronousStart* scenario, then proceed to evaluate the *SingletonBest* and *SingletonWorst* scenarios.

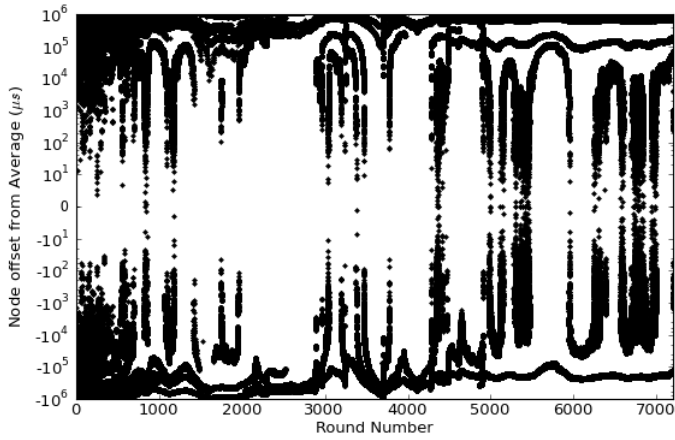
In Figure 6c we show the average behavior of $\langle Passive+Ids \rangle$ over 32 simulated runs. The performance is clearly superior to that of $\langle Active \rangle$ (Fig. 4a) on larger topologies, though it does take longer to converge for small networks, as seen from the 64-node topology results. At larger network sizes, the consistent ordering of clusters turns out to be an essential element that overcomes passive detection’s



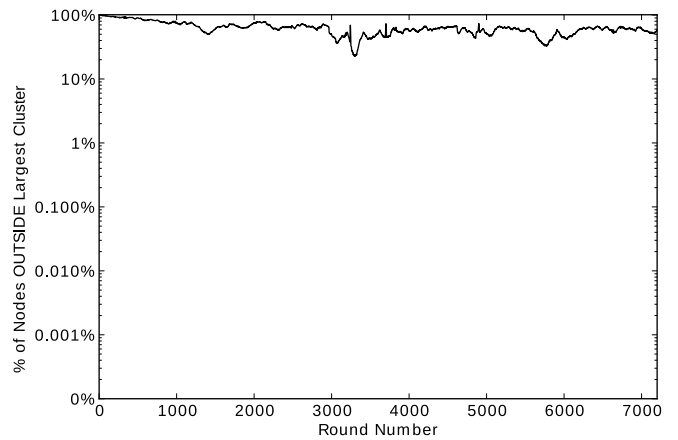
(a) A 256-node run, showing proper synchronization



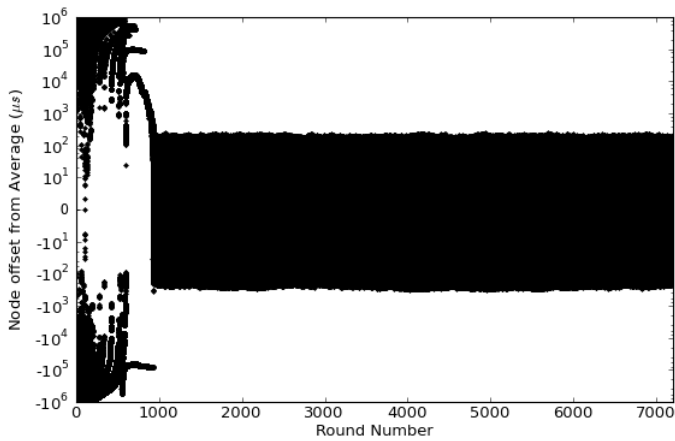
(b) The simulated run as Fig. 5a, showing the size of the largest cluster



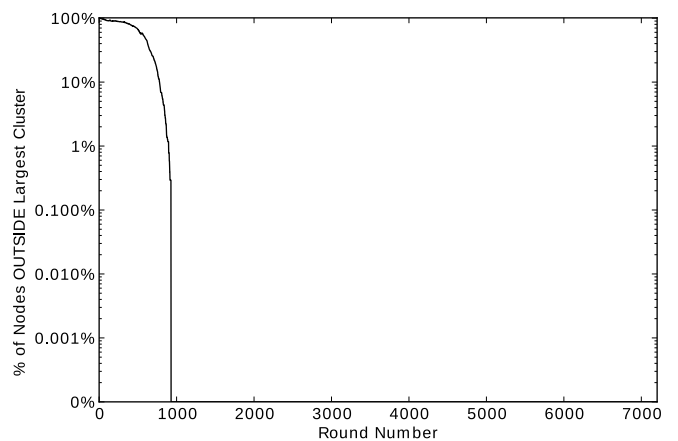
(c) Example 1024-node simulation that fails to reach complete synchrony



(d) Cluster size in the same 1024-node run as Fig. 5c

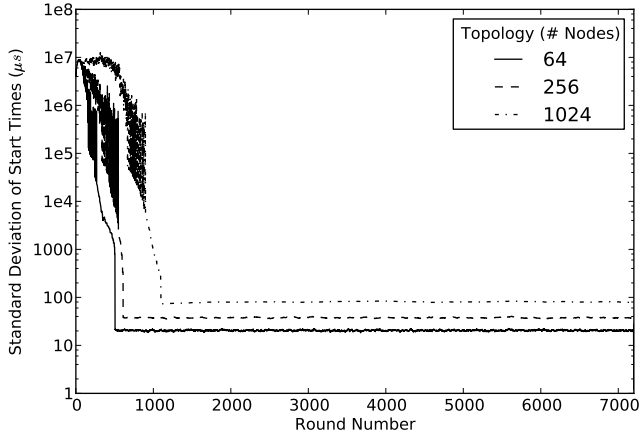


(e) A 1024-node $\langle \text{Active+Ids} \rangle$ run demonstrating correct synchronization

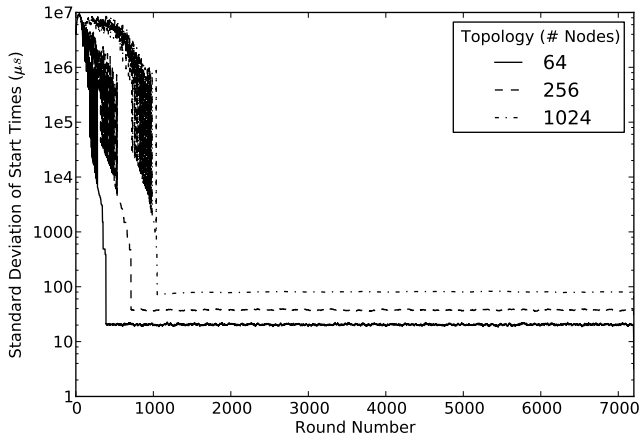


(f) Cluster size in the same 1024-node run as Fig. 5e

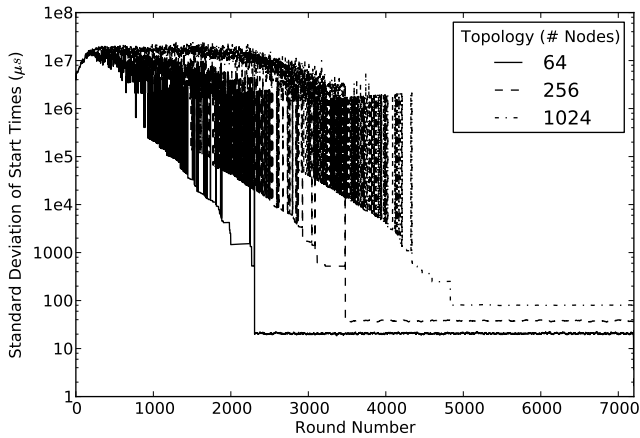
Fig. 5: The problem with $\langle \text{Active} \rangle$'s merge mechanism and a proposed solution, $\langle \text{Active+Ids} \rangle$



(a) $\langle \text{Active+Ids} \rangle$



(b) $\langle \text{Active+Ids+Listen} \rangle$



(c) $\langle \text{Passive+Ids} \rangle$

Fig. 6: Comparison of configurations using *cluster IDs*

inherent performance disadvantage. However, when compared to the $\langle \text{Active+Ids} \rangle$ configuration (Fig. 6a), there is really no competition. Active detection is several times faster than passive detection, because of active detection’s ability to recruit multiple inferior nodes with a single broadcast.

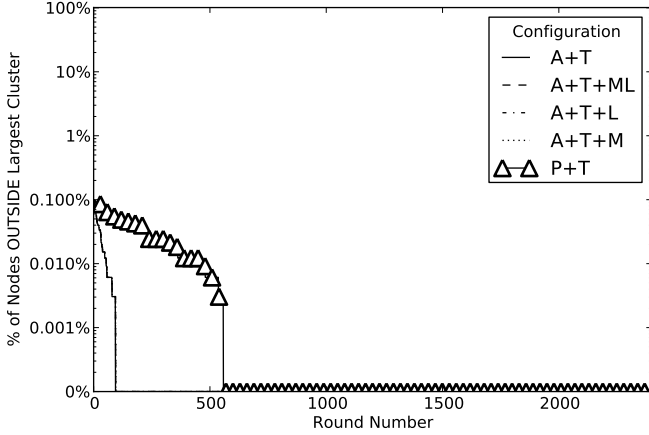
We evaluate our other proposed improvement to detection, *listen-before-merge*, using the same chaotic start scenario in Figure 6b. Though hard to see in the graphs, the *listen-before-merge* behavior does give a performance improvement on the largest test-case, but it is quite minimal. This scenario should be the one in which this behavior provides the most benefit, as there may be many different clusters in a node’s transmit range. However, this additional listening can only help if there are more than one *superior* cluster in range, and even then, it will only help if the node detects a second-best cluster to begin with.

We continue our evaluation by looking at our singleton cluster scenarios. In Figure 7 we can compare the performance of $\langle \text{Passive+Ids} \rangle$, labeled ‘P+I’, to that of two of our active configurations: $\langle \text{Active+Ids} \rangle$ and $\langle \text{Active+Ids+Listen} \rangle$ (labeled ‘A+I’ and ‘A+I+L’, respectively). We simulate these configurations using both versions of the singleton merge scenario and a 1024-node topology. Each graph shows the percentage of nodes that are not synchronized to the largest cluster as a function of the simulated round. First, in Figure 7a, we examine $\langle \text{SingletonWorst} \rangle$, where the single node has an inferior cluster id and should therefore join the other cluster (containing all other nodes). All active detection configurations handle this simple scenario identically, and manage to synchronize the isolated node in an average of 100 rounds. Passive detection, on the other hand, requires nearly five times as long to synchronize.

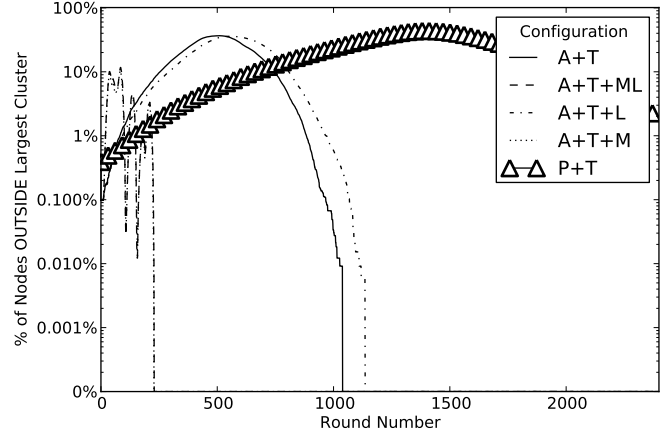
Finally, we look at the $\langle \text{SingletonBest} \rangle$ scenario. In this scenario, the isolated node has a superior cluster id, so all other nodes must leave their cluster and join the singleton cluster. In Figure 7b, we see that all the configurations using active detection eventually synchronize the network, but passive detection again performs poorly and consistently fails to synchronize the network. Unsurprisingly, the *listen-before-merge* optimization has not helped in this scenario, as there are no other clusters for it to detect. In fact, it performs slightly *worse* than the simple $\langle \text{Active+Ids} \rangle$ configuration. It seems that the additional listening causes delays in synchronization as nodes listen for better clusters that do not exist. Additionally, the *listen-before-merge* behavior costs a significant amount of energy. Using a duty cycle of 1%, a full frame of listening costs about the same energy as 100 ‘normal’ frames. Given the high cost and low success rate of *listen-before-merge*, we can discontinue investigating the $\langle \text{Active+Ids+Listen} \rangle$ configuration.

C. Notification

In this part, we delve into our final performance enhancement: notification. When one node detects another cluster and decides to join it, notifying his already synchronized neighbors can save a lot of energy. Each neighbor alerted by a merge

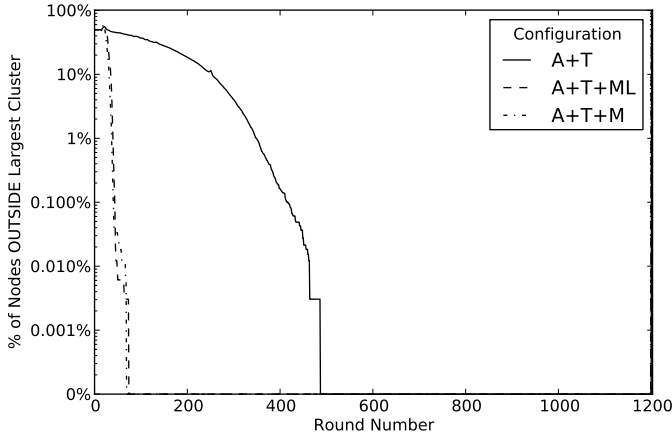


(a) SingletonWorst



(b) SingletonBest

Fig. 7: Passive detection compared to Active detection using our two Singleton scenarios

Fig. 8: Merging three separate clusters in the *ClusterMerge* scenario

message can be spared many rounds of operating in a dying subset, with few or no neighbors.

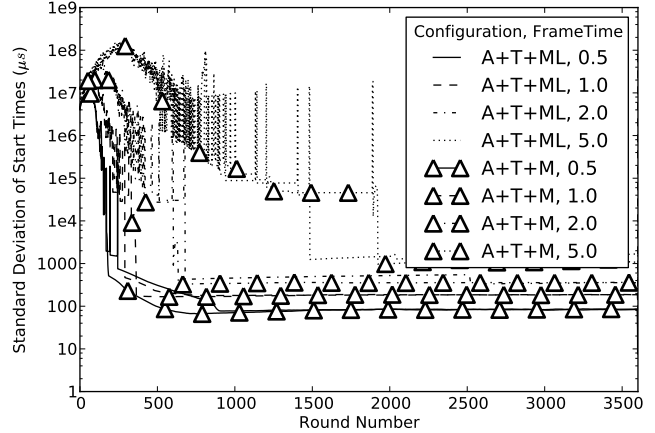
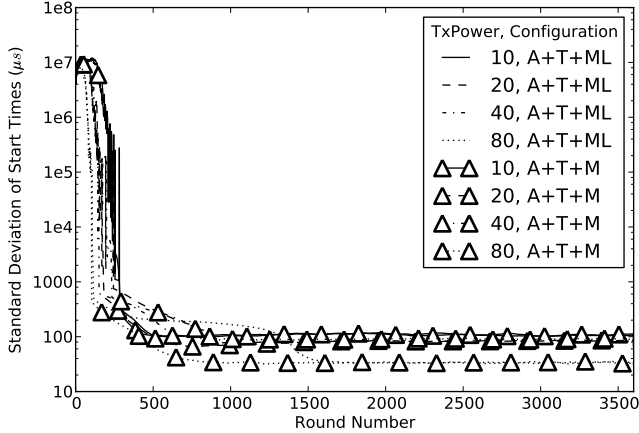
We begin by referring again to Figure 7b. The reduction in the time required to reach complete synchrony for the $\langle \text{Active+Ids+MergeMessages} \rangle$ and $\langle \text{Active+Ids+MergeMsgs+Listen} \rangle$, labeled ‘A+I+M’ and ‘A+I+ML’, is clearly significant. These two configurations perform almost identically and can synchronize all nodes in the simulated 32×32 grid almost five times faster than the configurations without merge messages. The best part about the merge messages is that they are essentially free. The only cost is the overhead of a two-byte merge offset with each application message.

In order to further evaluate the performance of the merge message optimization, we turn to our final scenario, *ClusterMerge*. In this scenario we simulate the merging of three sepa-

rate clusters in a 1024-node grid. Figure 8 shows the results of our three remaining configurations. The basic $\langle \text{Active+Ids} \rangle$ configuration performs quite well, taking only 500 rounds on average to converge to a single cluster. However, the configurations with merge messages perform even better and reach synchrony in less than 100 rounds.

We evaluate the effects of density and duty-cycle on synchronization, by returning again to our original scenario of an asynchronous network start. In Fig. 9a we examine the role of density on cluster merging by simulating each of the four transmit power settings described in Sec. V-C. The density of the network is determined by the nodes’ transmit power, so we can directly investigate the effects of network density on synchronization. Here we evaluate only the two most promising configurations: $\langle \text{Active+Ids+MergeMsgs} \rangle$ and $\langle \text{Active+Ids+MergeMsgs+Listen} \rangle$. The question we would like to answer is: does *listen-before-merge* offer any significant benefit in this scenario? The potentially large number of clusters present in the asynchronous start-up scenario provides the best opportunity for this modification to demonstrate its value. Unfortunately, at least in the case of varying network density, there is no clear advantage to performing a long listen before merging. Both configurations perform quite similarly, and as mentioned previously, the cost of performing these listening periods is prohibitively high without strong evidence of better performance. From the results of these simulations, it seems that network density has only a minor effect on the performance of the synchronization mechanism.

The results of our set of experiments extending GMAC’s frame time are shown in Figure 9b, and show how the nodes’ duty-cycle affects synchronization. Our default frame time to this point has been $\frac{1}{2}s$, giving a duty cycle of about 1.5%. We now simulate longer frame times with the same 8 slot active period yielding progressively lower duty cycles, with the lowest being about 0.15% at a frame length of 5s. Low-



(a) Experiments using 4 different transmit power settings, higher is more dense (b) Experiments using 4 different frame length settings, higher is lower duty-cycle

Fig. 9: The effects of density and frame length on *AsynchronousStart*

ering the duty cycle reduces the probability of detection, as discussed earlier, and that effect is clearly visible in our results. As the total frame time increases, the effects of clock drift are magnified as well, since nodes have less frequent opportunities to synchronize their clock with those of their neighbors. This behavior is clearly evident in our results, particularly at the highest frame length setting of 5s. Using such a low duty cycle drastically lowers the probability of detecting other clusters, though GMAC copes with this with frame lengths up to 2s. Furthermore, we again find no evidence that the *listen-before-merge* optimization is providing any noticeable performance benefit on top of that provided by the merge messages alone.

64×64 node grid. The performance advantage granted by the merge messages is again evident. Both configurations that include merge messages synchronize the entire network in an average of about 250 rounds, while the configuration without these messages takes approximately 2000.

VII. CONCLUSIONS AND FUTURE WORK

The main contribution of this paper is a thorough evaluation of several proposed methods of merging separately synchronized clusters. The results of our simulations show that the problem *is* solvable, and our solution can be used to achieve remarkably low duty-cycles, even with relatively inaccurate clocks. Our simulations have shown that GMAC is capable of synchronizing all nodes in a network so that they share a common active period, and doing so in a decentralized manner. Furthermore, simulations indicate that duty-cycles as low as $\frac{7ms}{2s} = 0.35\%$ are possible using these mechanisms.

All configurations using cluster IDs eventually synchronize the entire network, with the only difference being the time each configuration takes to do so. We demonstrated that while passive detection does consistently converge the network, it can take far longer than using active detection. Additionally, we demonstrated that the combination of active and passive detection can offer small performance benefit, but will generally not outweigh the additional energy cost. We have further shown that a simple notification message can drastically reduce the time for a network to reach a synchronized state, by as much as a factor of eight on our 4096-node topology. These two small modifications to GMAC's current behavior radically increase its suitability for large scale networks. The key insight is that as clusters build up, merge messages allow GMAC to leverage the inferior cluster's existing synchronization to rapidly merge *whole clusters*, not just individual nodes. Combined with a total ordering of clusters to solve the problem of which cluster to join, large and complicated networks can be synchronized in just a few minutes.

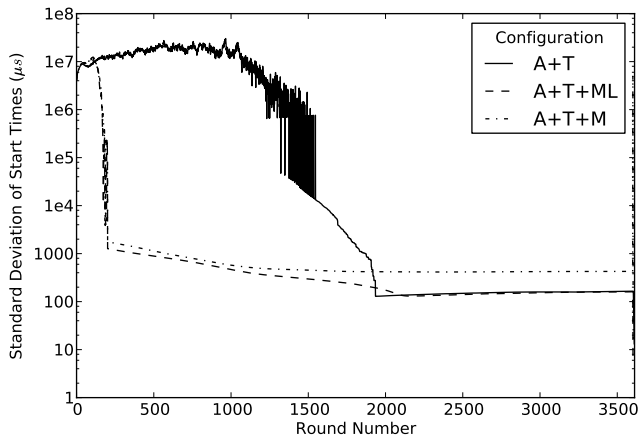


Fig. 10: Our largest topology, a 64×64 grid of 4096 nodes

Finally, we have one last set of simulations using an even larger topology to test the scalability of our best configuration, $\langle \text{Active+Ids+MergeMsgs} \rangle$, as well as $\langle \text{Active+Ids} \rangle$ and $\langle \text{Active+Ids+MergeMsgs+Listen} \rangle$. As shown in Figure 10, all these configurations can adequately handle the simulated

Our next step will be to verify the performance of these modifications by implementing them to run on our existing hardware. Additionally, we will investigate the effects of mobility on network synchronization. Mobility is always an issue in sensor networks, because the network topology is constantly changing. Even if not due to nodes *actually* moving, changing environmental conditions and failing nodes affect the network's topology in much the same way. Understanding the effect of mobility is therefore essential in order to evaluate this line of research. There are also many ways in which the methods described in this paper could still be improved. For example, nodes could be more adaptable to changing network conditions. Rather than sending a join message every round, nodes could keep track of the last time they heard a join message, in order to change the frequency of their join broadcasts. If a node in cluster A has not heard from any other cluster in a long time, it is likely that there are no unsynchronized nodes in its neighborhood and the active detection frequency could be reduced. As is often the case in sensor networks, there is a constant pressure to conserve as much energy as possible to prolong node lifetimes.

REFERENCES

- [1] M. Dobson, S. Voulgaris, and M. van Steen, "Network-level Synchronization in Decentralized Social Ad-Hoc Networks," in *ICPCA 2010*, 2010.
- [2] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient MAC protocol for wireless sensor networks," in *IEEE INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings*, 2002, pp. 1567–1576.
- [3] T. Van Dam and K. Langendoen, "An adaptive energy-efficient MAC protocol for wireless sensor networks," in *Proceedings of the 1st international conference on Embedded networked sensor systems*. ACM, 2003, pp. 171–180.
- [4] W. Ye, F. Silva, and J. Heidemann, "Ultra-low duty cycle MAC with scheduled channel polling," in *Proceedings of the 4th international conference on Embedded networked sensor systems*. ACM, 2006, p. 334.
- [5] M. Liu, T. Lai, and M. Liu, "Is Clock Synchronization Essential for Power Management in IEEE 802.11-Based Mobile Ad Hoc Networks?"
- [6] S. Mank, R. Karnapke, and J. Nolte, "An Adaptive TDMA based MAC Protocol for Mobile Wireless Sensor Networks," in *Proceedings of the 2007 International Conference on Sensor Technologies and Applications*. IEEE Computer Society, 2007, pp. 62–69.
- [7] —, "Mlmac—an adaptive tdma mac protocol for mobile wireless sensor networks," in *Ad-Hoc & Sensor Wireless Networks: An International Journal, Special Issue on 1st International Conference on Sensor Technologies and Applications*, 2008.
- [8] I. Cidon and M. Sidi, "Distributed assignment algorithms for multi-hop packet-radionetworks," in *IEEE INFOCOM'88. Networks: Evolution or Revolution. Proceedings. Seventh Annual Joint Conference of the IEEE Computer and Communications Societies*, 1988, pp. 1110–1118.
- [9] M. Arumugam and S. Kulkarni, "Self-stabilizing deterministic TDMA for sensor networks," *Distributed Computing and Internet Technology*, pp. 69–81, 2005.
- [10] S. Kulkarni and M. Arumugam, "SS-TDMA: A self-stabilizing MAC for sensor networks," *Sensor Network Operations*, 2006.
- [11] R. Tjoa, K. Chee, P. Sivaprasad, S. Rao, and J. Lim, "Clock drift reduction for relative time slot TDMA-based sensor networks," in *15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, 2004. PIMRC 2004*, vol. 2, 2004.
- [12] R. Pussente and V. Barbosa, "An algorithm for clock synchronization with the gradient property in sensor networks," *Journal of Parallel and Distributed Computing*, vol. 69, no. 3, pp. 261–265, 2009.