Decentralized Network-level Synchronization in Mobile Ad Hoc Networks

SPYROS VOULGARIS, VU University, The Netherlands and University of Patras, Greece
MATTHEW DOBSON, VU University, The Netherlands
MAARTEN VAN STEEN, University of Twente, The Netherlands

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 wireless mobile ad hoc networks that can potentially consist of thousands of nodes. More specifically, we are interested in operating the network with duty cycles that can be less than 1 percent of the total cycle time. We identify the fundamental issues that govern cluster merging, and provide a detailed comparison of various policies using extensive simulations based on a variety of mobility patterns. We propose a specific scheme that allows a 4,000-node network to stay synchronized with a duty cycle of approximately 0.7 percent. Our work is based on an existing, experimental MAC protocol that we use for real-world applications, and is validated in a real network of around 120 mobile nodes.

1. INTRODUCTION

Advances in electronics and embedded systems have led to wireless devices that are small, light, nonintrusive, and cheap. At present, we are capable of building and deploying very large networks consisting of tens of thousands of such devices, and we may expect to eventually see *smart dust* [Kahn et al. 1999] systems become reality.

A major concern of present and future wireless networks is their *lifetime duration* with energy being the main determining factor. Energy, in turn, is primarily dictated by the operation of a device's radio circuitry. For this reason, much research concentrates on reducing the time needed to keep the radio circuitry active, implying intermittently switching the radio on and off. The period during which a node's radio is on is known as its *active period*, which together with a subsequent *inactive period* forms a cyclic *round*. The ratio of the duration of the active period over the entire round is known as the *duty cycle*.

In order for several nodes to communicate, their active periods should be—at least partially—overlapping. In fact, to maximize the shared communication window and thus fully utilize the energy nodes spend on their radio circuits, their active periods should be synchronized as *accurately* as possible. In this paper, we aim at reaching such synchronization while demanding that all nodes are treated equally. In other words, we seek for fully symmetric autonomous synchronization without any node fulfilling a special role. Moreover, we aim at synchronizing the active periods of *all* nodes at the same time, and allow nodes to be mobile.

One can argue that having tens of thousands of wireless devices tightly and autonomously synchronized is not practical. First, it is much easier to use a number of fixed anchor points to which the other, mobile devices can synchronize. Second, instead of having tight synchronization of *all* devices, it is oftentimes better to deploy time-sharing scenarios so as to better utilize bandwidth. We agree. However, part of our research is motivated by *future* scenarios along the lines of aforementioned smart dust and so-called speckled computing. In such cases, nanoscale digital particles forming a very large ad hoc network are blended into our environment. We are already witnessing first steps toward such systems in entertainment scenarios where members of an audience wear wireless devices of which the leds are controlled to produce a light show [PixMob]. To us, it is important that we obtain a better understanding of the issues involved toward making such future networks.

When networks become very large, maintenance should be minimal. Autonomous synchronization without having to rely on specific nodes (such as fixed anchor points) is part of our design. For the same reason, having a *predictable* lifetime is important as it allows to anticipate maintenance efforts instead of having only reactive solutions. Also for these reasons, we strive for letting nodes have the same lifetime expectation so as to create a network in which all nodes can be treated symmetrically, again contributing to its maintainability.

The requirements of *long* and *predictable* lifetime duration have led to the GMAC[1] family of protocols. In GMAC, nodes use a very small duty cycle (less than 1%) and broadcast messages at fixed intervals, following a *gossip-based* communication model, from which GMAC gets its name. By ensuring constant energy use per time period, the lifetime of a network utilizing the GMAC protocol can be accurately determined before deployment. This allows the network owner to select appropriate battery hardware. The predictable lifetime of nodes also means that specific nodes (e.g., in an interesting location, proximal to some event) will not exhaust their energy supply prematurely simply due to being in a particular locale.

We are interested in enabling large-scale, mobile wireless sensor networks. Our current main use case is that of a *wearable* sensor network, in which each of the individual sensor nodes is worn or carried by a person. Real-world experiments involve gatherings of approximately 120 people. However, our solutions scale well to networks of thousands of nodes.

Ensuring that the active periods of nodes are synchronized is decomposed into two orthogonal subproblems:

— Once a group of nodes is synchronized, corrective actions are needed to alleviate effects of differences in clock rates that could easily lead to desynchronization.
— Distinct groups of synchronized nodes operating on *different* active periods should detect the existence of other such groups and merge such that each node operates on the same active period.

We have dealt with the first subproblem in [Dobson et al. 2010]. We have also partly explored the second problem in [Dobson et al. 2011], which we extend here to include an exploration of how node mobility affects synchronization while also considering much larger networks. This problem has not been addressed extensively by the research community. Although our solution is presented in the context of a specific MAC-layer protocol, the methodologies, principles, and algorithms we propose can be generalized to virtually any slotted MAC protocol with a very low duty cycle.

This paper's contribution is an extensive exploration of the design decisions, parameters, and configurations that govern the entire spectrum of network synchronization in mobile ad hoc networks. Namely, *achieving* and *maintaining* synchronization across all nodes, with duty cycles lower than 1%. We provide extensive quantitative insight through both simulations and a real-world experiment, as well as ample qualitative explanations of the protocol's behavior. This work constitutes a comprehensive concentration of knowledge and experience for scientists and engineers that wish to build, configure, and deploy large-scale wireless, static or mobile, ad hoc networks adhering to the gossiping paradigm.

The contents of the rest of the paper are as follows. We begin by describing the operation of the GMAC protocol that we have used for our research in Section 2. We then provide a thorough analysis of the aspects and methodologies of synchronization in Sections 3 and 4, including a discussion of the default behavior of GMAC and where it can be improved. We explain the measurements and metrics we use to evaluate our algorithms and provide a description of our simulation environment and experimental setup in Section 5. Following that, we present the results and evaluation of our simulations in Section 6. The results of one of our real-world experiments are discussed in Section 7. Section 8 contains a survey of related work and we come to conclusions in Section 9.

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

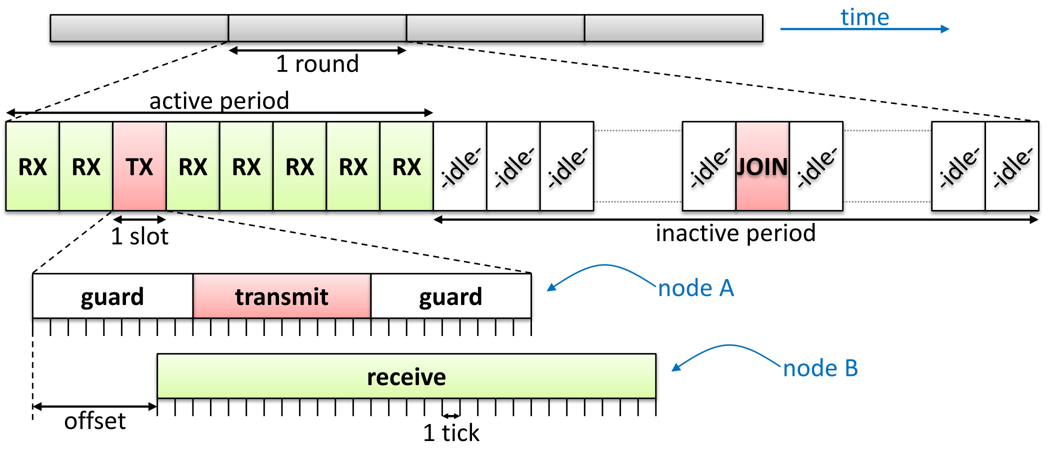


Fig. 1: GMAC's slot allocation and scheduling.

2. THE GMAC PROTOCOL

GMAC [Anemaet 2008] is a MAC protocol designed by Chess[2] to enable periodic, gossip-based communication, focusing on providing a long and predictable network lifetime. Although GMAC is a generic protocol that could run on any hardware platform, its official implementation is on Chess' *MyriaNed* platform, discussed in detail in Appendix A.

The selection of GMAC as the MAC protocol for our research stems from the protocol's nature, which makes it particularly challenging to maintain nodes synchronized in a single universal schedule in the face of node mobility. This is mainly due to GMAC's emphasis on an ultra-low duty cycle of less than 1%, and it is further aggravated by its completely decentralized mode of operation based on symmetric node behavior. Indeed, communication in an always-on MAC protocol, or when synchronization is governed by a central coordinator, is a nonproblem. However, such protocols would exhibit far lower lifetime, or more complex deployment and maintenance, respectively.

2.1. GMAC Overview

GMAC was inspired by the slotted Aloha protocol [Roberts 1975][Abramson 1977], but adds the notion of a fixed-length duty cycle. An illustration of a GMAC duty cycle is presented in Fig. 1. Primarily, GMAC operates by dividing time into a recurring series of *communication slots*, grouped together into *rounds* consisting of a fixed-number of slots. Some slots are considered *active* slots, during which nodes keep their radios activated in order to send and receive messages with their neighbors. Other slots are considered *inactive* slots, during which nodes keep their radios powered down in an effort to save energy. Of primary concern is the alignment of the active slots of all nodes, so that they can communicate effectively.

The length of a GMAC slot is the sum of the *transmit* time and *guard* time. In order to simplify things, the transmit and guard times are rounded up to an integer number of clock ticks. As stated above, the transmission time for a packet is about $300\mu s$, which is equivalent to 10 RTC clock ticks. A guard time of 18 ticks is used, although this can be significantly decreased. Thus, a single slot consists of 28 ticks and $T_{slot} = \frac{28}{32768}$ seconds, or about $850\mu sec$.

A *round* consists of s_{round} slots, having a duration of $T_{round} = s_{round} \cdot T_{slot}$ ticks. Scheduling many consecutive inactive slots allows the node to completely power down its radio (and

[2]http://www.chess.nl

even its CPU) in order to save a significant amount of energy. Exactly which slots are active and inactive is determined by GMAC's *strategy* module. Utilizing a low duty cycle, generally less than 1%, the synchronization of the active periods of neighboring nodes is crucial in order to maximize potential communication. The maintenance of synchronization between nodes executing the GMAC protocol is the responsibility of the *synchronization* module.

Fig. 1 depicts two types of transmission slots. The first, labeled "TX", is used to broadcast an application message, containing whatever data is provided by the application. The second, labeled "JOIN", is used to broadcast a `JOIN` message, containing synchronization information used by a recipient to discover (and potentially join) the sender's synchronized group. The use of `JOIN` messages is discussed further in Section 4.1.

It should become clear that, using exactly two message broadcast slots and a fixed number of message reception slots (in this case seven), the power consumption due to the MAC layer is fixed per time unit. Once a node has discovered an initial neighbor, the GMAC protocol never deviates from this rigid schedule.

2.2. Slot allocation strategy

For the purposes of this paper, we use a very simple slot allocation strategy module, called *RandomTX*. This strategy uses a fixed active period of $s_{active} = 8$ slots, followed by an inactive period lasting until the end of the round. In our experiments we use a round time of $1s$, which yields $s_{round} = 1170$ slots per round. With 8 active slots, we are left with $s_{inactive} = 1162$ slots, and a duty cycle of $\tau = \frac{8}{1170} = 0.68\%$. By using a fixed number of active slots per round a node's power consumption is constant and can be determined before deployment. It is this constant upper bound on energy use that allows the lifetime duration of a GMAC network to be computed.

Note that due to *RandomTX*'s use of a fixed number of active slots, the local density at each node will strongly affect how much useful communication can take place. If a node has significantly more than s_{active} neighbors, there will be many collisions as many senders will randomly select the same slot. There are more advanced slot selection strategies that attempt to compensate for this by adjusting the number of active slots based on estimates of the node's neighborhood size (Swarm-MAC and Distributed GMAC in [Anemaet 2008]). However, strategies of this type introduce more complexity because synchronized neighboring nodes still may not communicate as expected if they use different active period durations. Due to this, for the purpose of this paper we restrict our investigation to the simple *RandomTX* strategy, with the understanding that at very high node densities communication may be significantly hampered.

2.3. Synchronization

The inter-node synchronization is the focus of this paper, and is implemented in two distinct parts. First, by including the sender's current slot number within the round in every message, a receiving node can always determine the offset between its own notion of time and that of the sending node. By comparing the offsets of a number of different sending nodes, the receiving node can make an adjustment to the length of its round so that it starts the *next* round in synchrony with its neighbors. This maintenance of existing synchronization is provided by GMAC's *synchronization* module (discussed further in Section 3.2). The second part of inter-node synchronization, the merging of separately synchronized groups, is discussed in detail in Section 4.

We aim at synchronizing all participating nodes to a single common active period for one reason: to allow seamless node communication, even under high mobility. Our focus is on large-scale mobile networks, and our main use case is a wearable-sensor network. In this scenario there are no fixed areas in which nodes will operate, so infrastructure nodes (e.g., gateways, sinks) are not assumed. As nodes can freely move about, they should be able to immediately communicate with their new physical neighbors. If the participating nodes do not share a common active period, they will have to discover other nodes in their new location, which they may occupy only temporarily. Furthermore, if all or most of the nodes are mobile, most hierarchical or cluster-based methods of synchronization will be constantly readjusting. Synchronized nodes, on the other hand, can simply communicate with their neighbors regardless of their previous location.

2.4. Application Model

GMAC, as the **G**ossiping MAC, is designed with the assumption that the applications utilizing this MAC layer will communicate using a gossip-based mechanism. This is not strictly necessary, but it does serve to explain some of the design decisions made while developing the protocol. For example, GMAC does not provide any routing features, nor does it try to pass messages to a specific *sink* node. Because nodes use gossip to disseminate information, GMAC assumes that important information will be readily rebroadcast, while unimportant information will die out. If the operator of a sensor network running GMAC wants to inspect or modify the operation of that network, there are several options. Primarily, one can make use of passive *sniffer* nodes. These nodes do not actively participate in the network, but simply operate in a receive-only mode. The received messages sent by the *active* nodes can be interpreted on the sniffer itself, or potentially forwarded over the Internet for further processing.

GMAC provides a very simple API for application developers, primarily consisting of three callback functions:

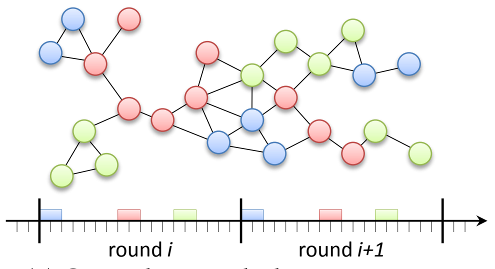
— **appInit** called at initialization time, when the application should initialize all relevant data.
— **appReceiveMsg** called for each received message, allows an application to parse the received message and take any appropriate action.
— **appPrepareMsg** called at the end of a round, allows an application to generate a message that will be broadcast during the next round.

Note that it is assumed that an application will generate a message for broadcast *every* round. Nodes rely on regular communication with their neighbors in order to remain synchronized, so sending one message per round ensures nodes will regularly receive messages. Because of the gossip-based nature of communication, this message may be composed of data generated by the sending node, data received and determined to be worthy of rebroadcast by the sending node, or a combination of both. Again, GMAC assumes that important data will eventually be spread through the network as nodes rebroadcast and disseminate the most “interesting” data they observe.

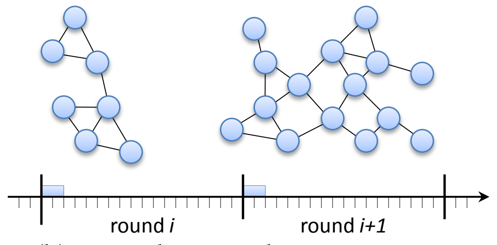
This also explains why the *RandomTX* strategy does not attempt to build a collision-free schedule. In networks of appropriate density some messages will be lost to collisions, but many will be received correctly. Since nodes broadcast a message in every round, a single lost message will not present a problem. Important messages will eventually be received by one or more neighboring nodes, where they can be further rebroadcast and disseminated throughout the network. If the density is above some minimum threshold (which we will explore in Section 6) and below some maximum threshold (based on the number of active communication slots, s_{active}), communication can proceed without excessive message collisions.

2.5. Hardware Considerations

As GMAC has been developed around the MyriaNed platform, we discuss here to what extent it has been tailored to that specific platform and in what ways. A detailed description of the MyriaNed hardware platform can be found in Appendix A.



(a) One subnet with three syncgroups



(b) Two subnets with one syncgroup

Fig. 2: The edges between the nodes represent wireless links. Nodes that are connected via (a chain of) wireless links are known as a *subnet*. The coloring of a node represents the group of nodes with which its active period overlaps, that is, its *syncgroup*. The axis at the bottom represents time and is divided into rounds of length T_{round} by the large numbered markers. The smaller markers denote individual communication slots. Finally, the shaded bars atop the time-axis represent the duration of the active periods of the similarly-shaded syncgroups.

The characteristics of the radio play the most important role in shaping the protocol. The radio used on MyriaNed nodes was originally intended for use in wireless keyboards, mice, etc., and is generally limited. More specifically, it has neither collision detection nor collision avoidance. This can make adaptive duty cycles tricky, as we cannot distinguish between channel over-saturation and under-utilization. In the former packets are lost due to collisions, in the latter there are simply no packets.

A second limitation set by the radio is that packets carry payloads of precisely 32 bytes. This constitutes a limitation mostly at the application layer.

Another point has to do with time accuracy. MyriaNed nodes are limited to a 32KHz oscillator for timekeeping, which limits how tight GMAC synchronization can be.

Finally, at a more general note, by using custom-made hardware we did not have a chance to benefit from the debugging and refinement that more “stable” or “common” platforms typically receive.

3. ESTABLISHING AND MAINTAINING SYNCHRONIZED GROUPS

Duty cycling makes synchronization of nodes’ active periods essential. Nodes whose active periods do not overlap cannot communicate with each other, effectively partitioning the network into *temporaly* disconnected components, called *syncgroups* (Fig. 2a). Nodes that are not within each other’s transmission range cannot communicate with each other either, partitioning the network into *spatially* disconnected components, called *subnets* (Fig. 2b).

The existence of separate subnets and their evolution in the face of mobility is beyond our control, depending exclusively on nodes’ physical locations. With respect to node synchronization, though, our aim is to synchronize *all* nodes at a single syncgroup irrespectively of their physical subnet (Fig. 2b), so that when two subnets meet, their nodes are readily able to communicate.

Ensuring node synchronization can be divided into three specific targets:

(1) Establish synchronization to an existing syncgroup at node startup.
(2) Maintain synchronization in an already synchronized group, negating the effect of clock drifts.
(3) Detect and merge two or more separately synchronized groups when they meet.

Targets (1) and (2) are discussed in the following two sections, 3.1 and 3.2, respectively. Target (3), being the most challenging, is discussed separately in Section 4.

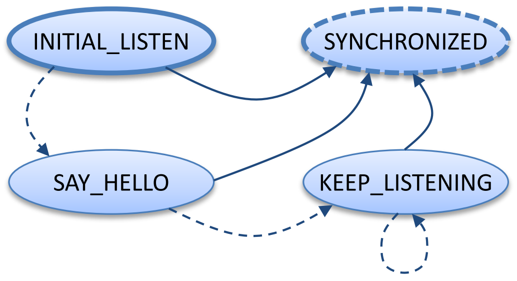


Fig. 3: The finite state machine describing the operation of GMAC when a node starts up. The `INITIAL_LISTEN` is the normal starting state, while `SYNCHRONIZED` is the goal state. A node will follow the dashed arrows if it receives no messages in a round, and the solid arrows if it does receive a message.

3.1. Establishing synchronized groups

In order to communicate, a node must first discover its neighbors. Nodes executing the GMAC protocol follow a simple finite state machine when initialized, shown in Fig. 3.

Nodes normally begin in the `INITIAL_LISTEN` state. In this state, a node i's first round will be spent with its radio active and continuously listening for a randomly-chosen number of slots $s_{0,i}$, with $s_{round} < s_{0,i} \leq 2 \times s_{round}$, where s_{round} is defined to be the number of slots in a round (Section 2). If the node hears a message while in `INITIAL_LISTEN`, it will immediately deactivate its radio, calculate a timer adjustment to align the start of its next round with that of the sender, move to the `SYNCHRONIZED` state, and then sleep until the computed start of its next round. If, however, a node in `INITIAL_LISTEN` reaches the end of its round without hearing a message, it will move to the `SAY_HELLO` state. In that state, a node will broadcast a specially-tagged "hello" message in the first slot of the round. After broadcasting the message, the node will switch its radio back to receive mode and enter the `KEEP_LISTENING` state. A node's behavior in `KEEP_LISTENING` is similar to that of a node in `INITIAL_LISTEN`. The differences are that in the `KEEP_LISTENING` state, a node i will maintain the standard number of slots in each round r (e.g., $s_{r,i} = s_{round}$), and will remain in that state indefinitely in the absence of receiving a message. A node in the `KEEP_LISTENING` state reacts to received messages in the same manner as described for `INITIAL_LISTEN`, aligning its next round with the sender and transitioning to `SYNCHRONIZED`. Upon reaching the `SYNCHRONIZED` state, a node executes the normal active/inactive duty cycle behavior, as discussed previously and shown in Fig. 1.

The reader will notice that in the above initialization behavior a node can spend a significant amount of energy in the `INITIAL_LISTEN` and `KEEP_LISTENING` states, as its radio will be active 100% of the time. This design decision runs contrary to GMAC's general philosophy of very low and constant power use. However, this decision was made to boost a node's initial discovery of a syncgroup, under the fundamental assumption that nodes will not be started in isolation.

3.2. Maintaining synchronized groups

Establishing synchronization across a set of nodes is necessary but not sufficient for a stable network. In the absence of local corrections, node clocks will drift apart rendering the network desynchronized.

In [Dobson et al. 2010] we showed that the *median algorithm* (Alg. 1) is capable of maintaining tight synchronization within connected networks of static grids of nodes. The algorithm is quite simple, yet very efficient. In each round, a node adjusts its clock to stay in closer synchronization with the set of neighboring nodes from which it successfully re-

Algorithm 1: The *Median* Algorithm

```
Median(numRxEntries > 0, array rxEntries, gain > 0)
    // sort received messages by offset (in ticks) from local time
    SortByOffset(rxEntries);
    // select the median entry
    medianEntry = rxEntries[numRxEntries/2];
    // scale the correction by gain = 0.5
    phaseError = medianEntry.tickOffset × gain;
    // adjust the length of the current round based on phaseError
    AdjustRoundLength(phaseError);
```

ceived a message. More specifically, at the end of its active period, a node considers the timing offsets of all messages it received in that round, and adjusts its clock *half way* toward the *median* offset. By always selecting the median timing offset, the algorithm ignores outlying data points and focuses on those in the center, effectively forcing the fastest and slowest nodes in the network to match those. Note that the gain of 0.5 serves in dampening the adjustments and in preventing oscillations of the synchronization maintenance behavior. We emphasize that it is crucial for scalability that the algorithm is based completely on *local decisions*.

It is worth noting here that GMAC was originally designed for static deployments on the scale of dozens to hundreds of nodes. As mentioned earlier, we are interested in networks at least an order of magnitude larger, so the sheer size of the networks we investigate may pose problems. Furthermore, although GMAC is designed to operate in a purely peer-to-peer manner without any dependencies on specific peers, it remains to be seen how mobility will affect its operation. With the introduction of dynamic topologies, good network-wide synchronization becomes even more important. In a static network, a node will always have the same neighbors, so tight local synchronization (i.e., among direct neighbors) with much looser global synchronization is perfectly acceptable. However, if a node can suddenly move away from its local synchronized group and position itself anywhere else, strong network-wide synchronization will be required.

4. MERGING SYNCHRONIZED GROUPS

While maintaining synchronization within syncgroups is fundamental, we must also ensure that GMAC can merge together separate syncgroups to form a single, cohesive network. The situation is illustrated in Fig. 2. When all syncgroups have been merged together and all nodes share a common active period, we say that the network has *converged*. Note that we reject solutions that attempt to "bridge" separate syncgroups by requiring nodes in the overlap to execute multiple active periods per round. Such behavior creates an asymmetric energy burden for gateway nodes, jeopardizing our goal of a predictable network lifetime. Furthermore, the premature failure of gateway nodes tends to be more detrimental than the failure of normal nodes, since it is precisely these gateway nodes that connect the otherwise isolated groups.

In our previous work, [Dobson et al. 2010], we demonstrated that the group-merging behavior of GMAC was sufficient to achieve convergence for small networks, but that GMAC struggled to consistently converge larger networks. A major contribution of this paper is a thorough analysis of various methods of merging large networks composed of groups of desynchronized mobile nodes. This problem of syncgroup merging can be further broken down into three subproblems: *detection*, *decision*, and *notification*, which we discuss in the following sections.

4.1. Detection

Before separately synchronized groups can be merged, they must first become aware of each other. We distinguish two methods of detection. In an *active* method, nodes *broadcast* a `JOIN` message during the inactive portion of their round, allowing other nodes using a different active period to detect the sending node's syncgroup. Note that `JOIN` messages are always transmitted during the sender's inactive period, but can be received only during another node's active period. In a *passive* method, nodes *listen* during the inactive portion of their duty cycle to detect *application* messages from nodes in other syncgroups.

4.1.1. Active detection. The effectiveness of active detection is mainly determined by the duty cycle of the network, τ. If $T_{active} > T_{inactive}$, then $\tau > 50\%$ (i.e., nodes are active for more than half of each round). This implies that the active periods of any two nodes overlap to some degree, therefore all nodes belong by default to a single syncgroup.

For duty cycles below 50%, we can compute that the probability p that a message transmitted during one group's inactive period will be received during another group's active period (ignoring collisions), is equal to $p = \frac{T_{active}}{T_{inactive}}$. By definition, the duty cycle is $\tau = \frac{T_{active}}{T_{active}+T_{inactive}}$, from which we derive $p = \frac{\tau}{1-\tau}$. Thus, the detection probability quickly becomes very low when τ is very small, which is exactly the case for the type of networks we are interested in. Nodes using a duty cycle of $\tau = 1\%$ can expect a detection probability of $p = 1.01\%$.

4.1.2. Passive detection. Passive detection offers a trade-off of increased energy consumption for faster detection. For example, a node could virtually guarantee detection of any other node in its range if it continued to listen for the entire duration of a round. However, this obviously defeats the original purpose of duty cycling, and would rapidly deplete the node's battery. We could apply the duty cycling method to passive listening by instructing nodes to listen to some percentage, p_l, of the inactive period, reducing energy consumption but also effectiveness. Note that this can be implemented as listening for an additional $p_l \times s_{inactive}$ slots every round or by listening to the entire round (an additional $s_{inactive}$ slots) with probability p_l. We chose to implement the latter method because listening to the entire inactive period eliminates the possibility that a node will fail to detect an unsynchronized neighbor due to listening to the "wrong" portion of its inactive period. Still, we will want to keep p_l as low as is practical, because the higher p_l is, the more energy is spent listening.

We have implemented both types of detection in order to allow for a comparison of the effectiveness of the two methods. The implementation of active detection sends one `JOIN` message per round, as described in Section 2. In our implementation of passive detection a node listens to the whole inactive portion of its round with probability p_l. In order to have a fair comparison between active and passive detection, we would like to spend approximately the same amount of energy in both cases. Sending a `JOIN` message costs the energy required to wake up the node, turn on the node's radio, broadcast a message, and turn off the node's radio again. On a node's Nordic nRF24L01+ radio (see Appendix A), this costs about the same as two active receive slots. Therefore, we tuned our implementation to listen for the equivalent of two (inactive) slots per round, and since $s_{inactive} = 1162$ slots, we set $p_l = \frac{2}{1162} \approx 0.17\%$.

Active detection does have a decided advantage over passive detection. A whole *set* of neighboring nodes may detect the existence of another syncgroup at once, by a *single* message broadcast by one node of that group, provided the `JOIN` message hits the active period of the neighbors. In the case of passive detection, each node would have to *individually* detect the presence of the foreign syncgroup, by paying the price of keeping its radio in receive mode during its inactive period. The disadvantage, however, of active detection is an increased chance of collisions, as the `JOIN` messages sent from one group may collide with each other, or with *application messages* belonging to other groups. Both active and passive detection schemes will be heavily affected by the density of the network, in particular, the number of neighbors, or *degree*, of participating nodes. In addition, mobility will influence the effectiveness of both detection techniques as well. When a node n from one syncgroup detects a node m from another, there is a chance that, due to mobility, node m will no longer be in node n's range during the next round, leaving n trying to merge into a syncgroup that has no members in its vicinity.

4.1.3. Listen-before-Merge. In addition, we implemented a modified version of passive detection designed to augment both detection methods. This technique is based upon the notion of *superior* and *inferior* syncgroups, which is explained below, in Section 4.2. Normally a node will merge immediately upon discovering (either via normal active or passive detection) a superior syncgroup. However, in very large networks where many syncgroups may be collocated, before joining a superior syncgroup that was discovered in the current round, it may make sense to listen for the whole inactive period in search of an even better syncgroup, in order to merge directly to the best syncgroup of the vicinity. This technique is called *listen before merge*. Technically it violates our principle of stable energy use, but the asymmetry induced by it is expected to be negligible as merges would be happening sporadically.

4.1.4. Targeted `JOIN` messages. We have also devised an improvement for active detection, called *targeted `JOIN` messages*. Normally, if a node in a superior syncgroup, A, detects (hears a `JOIN` message from) a node in an inferior syncgroup, B, it will simply ignore this message, assuming that the sender will eventually detect the existence of syncgroup A. Since, as discussed above, the detection probability is quite low and detection events are relatively rare, we should try to take advantage of A's detection of syncgroup B even if A is superior to B and therefore the node will not decide to merge. We can do this by allowing the detecting node in A to try to *target* syncgroup B's active period with its next `JOIN` message. Using the timing details from the sender's message, the receiver can determine an offset between the two syncgroups and, thus, can estimate when B's next active period will begin. By sending the next `JOIN` message in a slot that has a strong possibility of overlapping with B's active period instead of in a random slot, the node from syncgroup A greatly increases the chance of a neighboring node from syncgroup B to detect it.

4.2. Decision

Regardless of how detection happens, once a node from group B is aware of another group A, it must decide whether it should merge into A or if it should stay in B. Nodes cannot merge unconditionally, because otherwise the whole network may never converge as nodes merge back and forth between multiple groups. Our goal is to ensure that all the nodes converge into a single (possibly multi-hop) syncgroup, so that we should try to minimize the amount of time and energy spent on reaching a converged state.

The decision algorithm should implement a relation $\succ$ that provides a *total ordering* of the set of existing synchronized groups of nodes. That is, the decision relation $A \succ B$ determines whether group A is superior to group B. Thus, when a node in B receives a `JOIN` message from a node in A, it should merge into group A if and only if $A \succ B$. The relation $\succ$ should provide the following three properties:

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

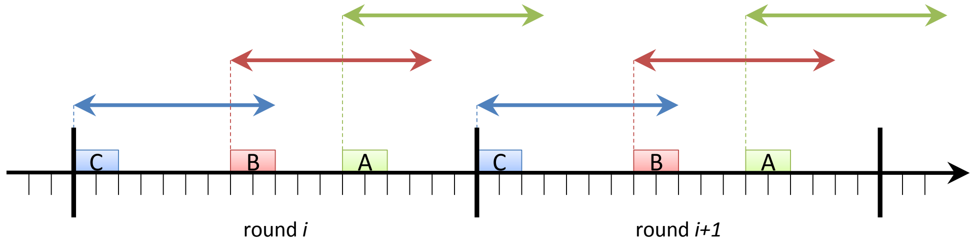


Fig. 4: A graphical representation of the problem in the timing-based decision mechanism called $\overset{t}{\succ}$. The axis at the bottom represents time in the same fashion as Fig. 2. The additional colored lines above the axis show the span of time where the `JOIN` messages from the associated syncgroup will be respected.

In order to promote convergence in the merging behavior, we propose enforcing a deterministic ordering of syncgroups. In static networks, nodes should eventually detect all other nodes/groups within their radio range. Provided that the network is connected, all nodes will eventually become aware of all other synchronized groups in the network. Because of the total ordering of these groups, nodes can always deterministically select the *best* group when making a merge decision. In theory, this should lead to network convergence as all nodes eventually merge into the best group.

Note that taking mobility into account does not directly affect the logic of deciding whether or not to merge. Nevertheless, there are effects that we need to consider, specifically on the design of the decision relation.

4.2.1. Timing-based decision relation, $\overset{t}{\succ}$. GMAC's default decision relation uses a heuristic mechanism to decide when a node should merge into a newly discovered group: if a received `JOIN` message was sent during the first half of the sender's round, then it is accepted as valid, otherwise it is discarded. Note that while sending `JOIN` messages in the second half of a node's round that will only be discarded seems wasteful, the messages contain debugging information so they are sent regardless of whether or not they will be considered valid. This timing-based relation, $\overset{t}{\succ}$, is meant to ensure antisymmetry (Property 1) for the decision relation. This is because, for any two groups, 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. Furthermore, $\overset{t}{\succ}$ provides totality (Property 2), because the two groups cannot be desynchronized by more than half a round, implying the active period of one overlaps with the first half of the other's round. However, $\overset{t}{\succ}$ does not provide transitivity (Property 3). If more than two groups exist in each other's range, there can be a 'loop', i.e., where nodes can merge from A to B to C and then back to A. In the best case, one of the groups in the loop can be eliminated if all of its nodes merge into the other groups.

For example, if groups B and C could get all of the nodes in A to merge into their respective groups before A can get any nodes from C to merge into it, then the loop would have been removed. In the worst case, these loops can persist forever, leading to a network that never converges. A visual example of this effect is seen in Fig. 4. Nodes in group A will accept `JOIN` messages from group B, because the first half of the group B's round overlaps with the active period of group A (i.e., $B \overset{t}{\succ} A$). Similarly, nodes in group B will react to `JOIN` messages from group C. Finally, due to the timing of their active periods, nodes in group C will react to join messages only from nodes in A. It is thus seen that the three groups form a loop, allowing for a node to merge from A to B in one round, from B to C in a subsequent round, only to later merge back into group A, ad infinitum. We will provide a solution to this problem below.

4.2.2. Cluster-based decision relation, $\overset{\mathcal{C}}{\succ}$. Ideally, we would like the group with fewer nodes to always merge into a group 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 group, we would still need a method of breaking ties between groups of equal size. Such a tie-breaker method can also serve as the primary criterion for the group-merge decision. This may lead to suboptimal merge operations, i.e., forcing many nodes to resynchronize to match a few. Deterministic convergence, however, is more important than optimality, particularly because in a stable network the occurrence of merge operations can be assumed to be infrequent. That said, the effects of mobile nodes on network convergence will be a chief aspect of our evaluation.

We propose to solve the convergence problem using a relation, $\overset{\mathcal{C}}{\succ}$, based on *cluster tags*. A cluster tag is simply an identifier used by a group of nodes that share a common active period, i.e., a syncgroup. Nodes that have the same cluster tag are said to be in the same *cluster*. Note, however, that two nodes in the same syncgroup need not be in the same cluster, e.g., because the best cluster tag has not propagated throughout the entire syncgroup yet. Similarly, two nodes in the same cluster need not be in the same syncgroup, e.g., because they were synchronized but have drifted apart.

In our solution, a cluster tag C is composed of two integers, an *ID* and an *epoch*, and written $C = \{C.id,\ C.epoch\}$. We assume that all nodes have a unique identifier and, upon starting, nodes initially use their own unique identifier and an epoch of 0 for their cluster tag. We will discuss the use and function of epochs below. Nodes in the `INITIAL_LISTEN`, `SAY_HELLO`, or `KEEP_LISTENING` states (described in Section 3.1) ignore cluster tags, and will respond to any received message by synchronizing with its sender. Once in the state `SYNCHRONIZED`, a node will strictly respect the ordering of cluster tags by following the protocol described here.

Using $\overset{\mathcal{C}}{\succ}$ a node can make a deterministic decision with regards to merging. A node can compare its own cluster tag, A, with a received tag, B, using this relation: $B \overset{\mathcal{C}}{\succ} A$ if $B.id > A.id$ (again, ignoring epochs for now). A synchronized node will always adopt a superior cluster tag received from other nodes in the *same* syncgroup. That is, if a node with cluster tag A receives an application message with cluster tag B during its active period (signifying that the sender's active period overlaps with its own) and $B \overset{\mathcal{C}}{\succ} A$, it should discard its old tag and adopt the tag B. Similarly, if this node in A detects a node with cluster tag C from a *different* syncgroup (either by hearing a `JOIN` message during its active period, or by overhearing any message while listening during its inactive period), it can simply compare its own cluster tag to that of the other node. If $A \overset{\mathcal{C}}{\succ} C$, its own ID is higher it can ignore the message. However, if $C \overset{\mathcal{C}}{\succ} A$, the other group's cluster ID is higher and the node can deterministically decide that it should merge into the other cluster and react accordingly. By assuring that the nodes in a syncgroup with a superior cluster tag never merge into a syncgroup with an inferior cluster tag, we can eliminate the looping problem in the $\overset{t}{\succ}$ decision relation. The relation $\overset{\mathcal{C}}{\succ}$ provides for all three properties, including the transitivity missing from GMAC's default relation, $\overset{t}{\succ}$. Without loops, all other syncgroups should eventually merge into the group with the best cluster tag.

4.2.3. Cluster-based decision relation with epochs, $\overset{\mathcal{E}}{\succ}$. However, node mobility will complicate matters. One can imagine that the situation presented in Fig. 2a has been resolved and all nodes have merged into the same syncgroup, sharing a common cluster tag as well. After some time, due to mobility, the nodes may have changed their locations and are now physically separated into two different subnets as shown in Fig. 2b. It is likely that the median clock frequencies in the two subnets are different. Thus over time, the physically separated nodes, though once synchronized, will slowly drift apart. The nodes in each subnet should continue to maintain synchronization within their subnet, however, a problem can later arise because both of these groups have the same cluster tag but will no longer be in the same syncgroup. If, later in the scenario, mobility brings the two subnets into contact again, nodes in each syncgroup will ignore `JOIN` messages from the other syncgroup, since neither possesses a *superior* cluster tag. This scenario is what we call a *cluster split*.

Our solution to this problem is to introduce *epochs*. Epochs are given precedence over IDs in implementing the $>$ relationship, so a tag with a higher epoch is always superior to a tag with a lower epoch. This essentially invalidates tags with lower (older) epochs when a higher (newer) epoch is created. We call this modified relation $\overset{e}{\succ}$: $A \overset{e}{\succ} B$ if and only if $(\boldsymbol{A.epoch} > \boldsymbol{B.epoch})$ OR $((\boldsymbol{A.epoch} == \boldsymbol{B.epoch})$ AND $(A.id > B.id))$. Using the epoch counter in its cluster tag, a node can increase the weight of its cluster tag by incrementing its epoch when detecting a cluster split. That is, when a node X with cluster tag $A = \{a, e\}$ detects another node Y also claiming cluster tag A but with a different active period, node X generates a new cluster tag $A' = \{a', e + 1\}$. This new tag will contain a randomly generated ID, a', and an epoch counter one higher than that of the old tag, $e + 1$. The higher epoch makes this new cluster tag superior and ensures that the node's synchronized neighbors will adopt and disseminate this new tag. Nodes generate a new random ID in order to prevent the epoch counter running up to infinity without resolving the split in the case that nodes from two syncgroups experiencing a split of cluster tag $B = \{b, e\}$ independently and simultaneously detect each other. If they did not generate a new ID, each node would increment the epoch of its cluster tag but keep the same cluster ID, resulting in both nodes having the same tag, $B' = \{b, e+1\}$. This process could repeat indefinitely, causing the epoch counter to count upwards to infinity without resolving the cluster split.

Note that the generation of a new cluster tag, B', does not cost any extra energy or computation for the synchronized neighbors of this node. Neighboring nodes that still have the cluster tag B will (eventually) receive a message from the node with tag B'. When this occurs, the receiving node will simply compare B and B', determine that B' is superior, and adopt this new tag. Because the receiver is synchronized with the sender (that is, their active periods overlap allowing them to exchange messages) the receiver will not need to make a large resynchronization, but only to adopt the new tag. If *many* nodes detect a cluster split at the same time, there may (temporarily) exist a variety of cluster tags: B, B', B'', etc. The best of these will quickly dominate and cause the elimination of the others, however.

4.3. Notification

By default GMAC does not use any notification of discovered groups. Nodes that decide to change groups just silently merge. That is, they leave their old group by adjusting the length of their current round to align their next round with their new group. In situations with large groups, it can take very long to reach complete synchronization.

4.3.1. Merge messages. Once a node has decided that it must merge into a new group, it should notify its own group of the merge. Though not strictly necessary, notification of the node's decision to switch from group B to group A can be rapidly propagated through group B (leveraging the group's existing synchronization), saving the need for repeated detections of group A. Because the probability of detection is proportional to the duty cycle, the networks we investigate will have very low detection probabilities. Propagating a notification of detections will reduce the number of detection events necessary to synchronize the entire network.

In order to add notification functionality to GMAC, 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 group. After discovering a group with a better cluster tag, a node can record the time difference, or offset, between its own group and the one with the superior cluster tag. Then, rather than immediately merging into the new group, it can stay synchronized to its current group for one more round, in order to communicate with its neighbors and inform them about the new superior group. By sending this merge offset along with its message in the following round, its current neighbors can be made aware of both the existence and the offset of this superior group *without* the need to detect it on their own. This notification should greatly reduce the time and energy spent on detection.

Again, when taking mobility into account, we expect that the main effect on merge notification will be the same as for detection. Namely, that a node may receive a notification of a merge decision and adjust its next round to synchronize with the new group, only to find that the member(s) of that group are no longer within communication range. How drastic this effect is will depend on the density of the particular scenario as well as the speed of the nodes in question. When considering node density, the positive effect from the merge messages should be proportional to the density of the network. That is, the denser the network, the greater the number of nodes that can be notified of a merge detection/decision. We will examine both of these issues experimentally later in the paper.

5. SIMULATION SETUP

Before presenting the actual evaluation results (Section 6), we provide preliminary information on the GMAC configurations tested, the topologies used, the metrics considered, as well as the simulator itself.

5.1. GMAC Configurations

In order to facilitate discussion of GMAC's behavior with various alternatives switched on or off, we will analyze several specific combinations of behaviors, called *configurations*. Table I presents a comprehensive list of our configurations.

— **Active:** The default GMAC behavior, as described in Sections 2, 3, and 4.
— **Passive:** A configuration using passive detection, rather than active, with p_l =0.17%, as described in Section 4.1.2.
— **Active+Cluster:** Active detection augmented by cluster tags (relation $\overset{c}{\succ}$, Section 4.2.2) in order to make consistent merge decisions. The only cost of cluster tags is an additional 3 bytes in the message.
— **Active+Cluster+Notify:** The same as Active+Cluster, but nodes do not immediately merge into a newly discovered group, rather they wait one round in order to send a merge message (Section 4.3.1) to their neighbors. This notification adds a cost of an additional 2 bytes in the message.
— **Active+Cluster+Listen:** The same as Active+Cluster, but nodes do not immediately merge into a newly discovered group, rather they listen for a whole round in order to discover the best cluster tag in range (Section 4.1.3). The cost of listen-before-merge is the energy of up to one full round in receive mode each time a node detects a new cluster.
— **Active+Cluster+Notify+Target:** This configuration combines active detection, cluster tags, merge messages, and targeted join messages. The idea of targeting join messages

Table I: GMAC configurations investigated

Name	Abbreviation	Synchronization Aspect	Section
Active Detection	Active	Detection	4.1.1
Passive Detection	Passive	Detection	4.1.2
Listen before Merge	Listen	Detection	4.1.3
Target Join Messages	Target	Detection	4.1.4
Cluster Tags	Cluster	Decision	4.2.2
Notify on Merge	Notify	Notification	4.3.1

Table II: Network sizes investigated

Nodes	Area (m^2)	Node Density (node/m^2)
100	$\approx 100,000\ (316m \times 316m)$	≈ 0.001
500	$\approx 500,000\ (707m \times 707m)$	≈ 0.001
1000	$1,000,000\ (1000m \times 1000m)$	0.001
4000	$4,000,000\ (2000m \times 2000m)$	0.001

was introduced in Section 4.1.4. The targeted join message has no additional cost, since the join message would have been sent anyway.

5.2. Topology

To better assess the strengths and weaknesses of the various configurations, we investigate the effect of topology on group merging. In particular, we look at network size and node mobility. We investigate a number of mobile scenarios, created using the BonnMotion [Aschenbruck et al. 2010] framework. We selected three (*Gauss-Markov*, *Random Walk*, and *Reference-Point Group Mobility*) of the fourteen mobility models provided by BonnMotion and generated traces of several sizes, shown in Table II (for detailed parameters, see Table V in Appendix B). Notice that in all cases we ensure that the scenarios have an average node density of one node per thousand square meters ($\frac{1node}{1000m^2}$), in order to increase comparability between results from various models. With the same goal, we also use similar movement speeds (i.e., a maximum of $5\frac{m}{s}$, unless otherwise stated) and other parameters across all three scenarios.

With respect to density, in particular, we vary it by varying the transmission power for all nodes in the network on a per-run basis, effectively varying the average number of neighbors per node. We use the term *transmission density* to denote the average number of neighbors per node. Table VI in Appendix B lists the simulated transmission powers we use, along with the associated transmission range, transmission area, and transmission density for each.

5.3. Simulator

OMNeT++ ([Varga and Hornig 2008], [Weingartner et al. 2009]) is an open-source discrete event simulator, and the MiXiM extensions ([Köpke et al. 2008]) provide a framework for wireless and mobile networking simulations. The OMNET++ platform is expressive, efficient, modular, and the de facto simulation environment for mobile ad hoc and sensor networks, while MiXiM provides support for mobility and wireless network protocols.

In addition to modules implementing the GMAC protocol itself, we also designed our own OMNeT++ modules to represent the clocks found in our sensor nodes. OMNeT++ keeps track of the global simulation time, t, while the clock module for an individual node i computes the local time, t_i. A node's local time is based on its own clock's frequency offset (f_i) and phase offset (p_i), provided as OMNeT++ simulation parameters. Thus, i can compute $t_i = (t \times f_i) + p_i$. 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 offset determines how much faster or slower than simulation time a node's clock runs. Unless otherwise specified, the clock at each node, i, will use a random frequency multiplier $0.99998 < f_i < 1.00002$, i.e., ± 20 parts per million. It is the differences in this local clock frequency offset that cause the simulated nodes to want to drift apart.

One weakness of this model is that we do not account for potential temperature differences experienced by nodes throughout the network. Ambient temperature is one of the largest influences of variability in electronic timing hardware. Nevertheless, such temperature-based variability in the clocks should not significantly affect the median algorithm, since it is insensitive to a node's historical clock rate. That is, the median algorithm considers only the instantaneous timing offset amongst its neighbors, so the fact that its own (or its neighbor's) clock was running at a slightly different rate a minute ago should not affect the functionality of the algorithm. This is demonstrated in Section 7 by the operation of GMAC in a conference venue with multiple rooms of (presumably) different ambient temperatures.

5.4. Performance Metrics

Nodes report two important pieces of data at the beginning of each new round, r. Each simulated node i logs the global simulation time, $t_{r,i}$, it began round r, as well as its current cluster tag, $c_{r,i}$. Using the logged timing data, we can see not only which nodes *are synchronized* to which other nodes (i.e., whether their active periods overlap), but how *tightly* they are synchronized (i.e., how *much* their active periods overlap). Through the recorded cluster tags, we can determine which nodes *think they are synchronized* with which other nodes. In addition to time and cluster tag, each node i records its (x, y) position at the start of the round, $x_{r,i}$ and $y_{r,i}$. Nodes also log a number of packet-level statistics, like the number of sent and received packets (both application and join packets), number of collided packets, and the number of attenuated packets (those lost due to weak/distant transmission signals).

Given these measurements, we derive the following three metrics to quantitatively evaluate both the default GMAC protocol, as well as our suggested modifications to it:

— **Local synchrony:** In order to get a measurement of local synchronization, we compute the standard deviation of start times amongst the direct (1-hop) neighbors of each node. That is, for each logged round, r, and for each simulated node, i, we compute the set of neighbor nodes (including i itself), $N_{r,i}$, and then compute the standard deviation of the round start times within that set, $\left\{t_{r,j} : j \in N_{r,i}\right\}$. This gives us a measure of the variability in the synchronization for each local neighborhood. By averaging this result across all nodes in the simulation, we can see how network-wide synchronization progresses during a simulated run. We denote this *local* standard deviation as λ_r.
As λ_r is measured in seconds, it is important to put it in perspective relative to a simulated node's "hardware" timing. As with the real hardware (see Appendix A), a simulated broadcast takes about $300\mu s$, a full communication slot lasts about $850\mu s$, and a full active period (8 slots) has a duration of approximately $7000\mu s$. Nodes can potentially communicate if their active periods overlap even slightly, but we would prefer their active periods to be perfectly aligned. We consider a local neighborhood to be synchronized if the nodes are offset by at most one communication slot. As such, we choose $\lambda_r \leq 300\mu s$ to be the point where we consider a network to be *tightly* synchronized. If the standard deviation is $300\mu s$, then 99.7% of the contributing round start times are within three standard deviations (i.e., $900\mu s$), or approximately the duration of one communication slot. We select $\lambda_r \leq 2000\mu s$ to be the point where we consider a network to be *loosely* synchronized. Again, a standard deviation of $2000\mu s$ implies that almost all node neighborhoods are synchronized to within $6000\mu s$.
Note that this measurement is more meaningful in static scenarios, where a node will have persistent neighbors, than in mobile scenarios, where a node's neighborhood will be constantly changing. Thus, the more mobile the nodes, the more we need to focus on tight global synchronization.

— **Global synchrony** In order to evaluate the degree of synchronization across the entire network, we compute the *global* standard deviation, σ_r, of reported start times for round r across all nodes. We compute σ_r as above with λ_r, but $N_{r,i}$ is always the set of *all* nodes. As such, when σ_r is also within the bounds described for λ_r, then we can be confident that the network is synchronized at a global level. This is extremely important in the mobile scenarios, where tight local synchronization but a lack of global synchronization could pose a problem.

— **Network synchronization** It is useful to be able to condense an entire simulated run down to just a single metric in order to easily compare the effect of different parameters on the execution. We consider one such metric to be the number of nodes that are mutually synchronized. As we study networks of different sizes, it makes sense to compute the *percentage of mutually synchronized nodes*. For the purposes of this metric, we consider a set of nodes to be synchronized when the difference between the largest and smallest reported times for round f is less than $\delta = 12ms$. Here we take δ to be a hard limit on the maximum timing offset within a group of synchronized nodes. This value represents 1.7 active periods, indicating that there should be at least *some* overlap between the active periods of all nodes in this range. To compute this percentage, we sort all of the nodes' logged timestamps for round f, find the largest cluster of results that falls within the window of length δ, and divide the size of this cluster by the total number of simulated nodes.

6. EVALUATION

Our evaluation of GMAC synchronization and our proposed improvements follows the order in which they have been described throughout the paper. That is, it is split up in *stability* and *convergence* analysis.

In stability analysis (Section 6.1), we explore GMAC's capability at *maintaining synchronization*. More specifically, we investigate synchronization performance with respect to five factors: (i) node density, (ii) active period length, (iii) node velocity, (iv) network diameter, and (v) broadcast channel reliability.

In convergence analysis (Sections 6.2-6.4) we explore GMAC's capability to *merge* sync-groups, through the proposed techniques for (i) detection and (ii) decision. All techniques are combined to demonstrate their performance in large networks of 4000 nodes (Section 6.5).

All results are generated by simulating each GMAC configuration eight times, each iteration with a different random seed. The lines plotted represent an average across these eight repetitions of the same configuration.

6.1. Maintenance

We evaluate how well GMAC maintains synchronization in static and mobile scenarios by simulating a network that is initially perfectly synchronized. That is, all nodes begin execution at the same time in the `SYNCHRONIZED` state (Section 3.1).

We assess the level of synchronization based on the local and global synchrony metrics, as defined in Section 5.4. More specifically, we record and plot the standard deviation of round start times, λ_r and σ_r, at the beginning of each round. As the nodes are initially synchronized, all graphs start with $\lambda_0 = \sigma_0 = 0$. Two dashed horizontal lines at $y = 2000\mu s$ and at $y = 300\mu s$ have been added on all graphs, to indicate the thresholds explained previously.

6.1.1. Effect of node density.

In Fig. 5 the upper set of graphs presents the local standard deviation λ_r, while the lower part presents the global standard deviation σ_r. In the left-most and middle graphs we see the performance of the median algorithm on randomly-deployed static topologies of 100 and 1000 nodes, respectively, while the right-most graphs depict the performance on a 1000-node mobile scenario.

In the static random deployment of 100 nodes we can immediately see that node density has a direct effect on synchronization maintenance. Low-density topologies (lighter lines) are far more susceptible to clock drifts, failing to preserve an initially perfect synchroniza-

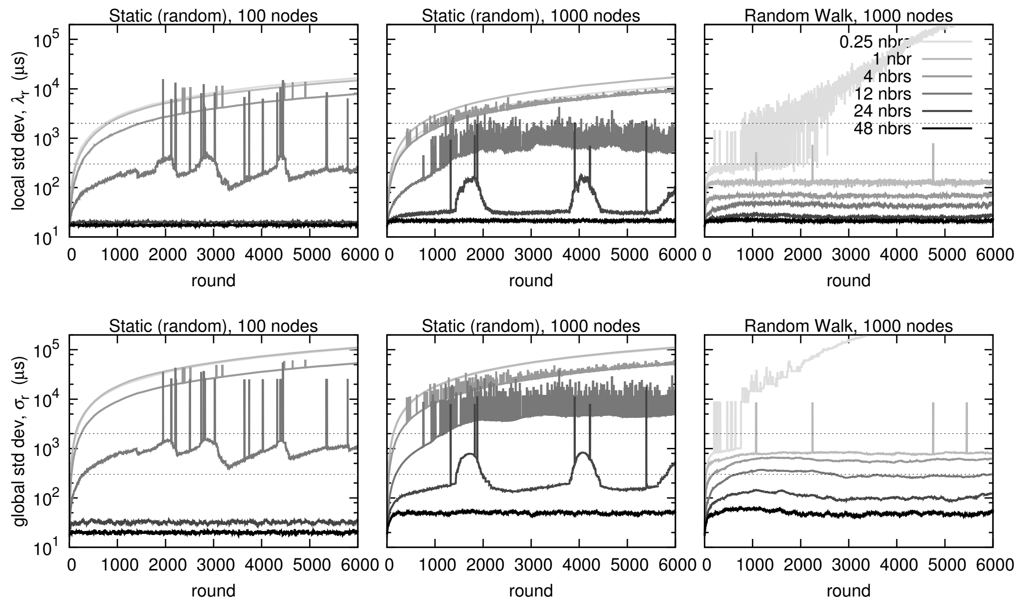


Fig. 5: Maintenance performance on several static and mobility (random walk) scenarios. From lighter to darker lines: 0.25, 1, 4, 12, 24, and 48 average neighbors per node.

tion, contrary to high-density topologies (darker lines) that manage to keep nodes tightly synchronized.

To explain this recall that GMAC compensates for clock drift by leveraging exchanged packets to continuously readjust node clocks through the median algorithm (Section 3.2). In low-density topologies the network is split into several disjoint subnets, effectively suppressing the median algorithm, forcing the network to gradually dissolve into disjoint sync-groups as node clocks drift apart. This is not an issue for high-density topologies.

Note that at a middleground density of 12 neighbors per node on average, the network is very tenuously connected resulting in loose synchronization right on the borderline of what we consider acceptable. As 12 neighbors per node is only an *average* value, there are nodes with a very small number of neighbors, which stay barely synchronized to the rest. Averaging data with very different orders of magnitude (due to weakly-connected nodes'

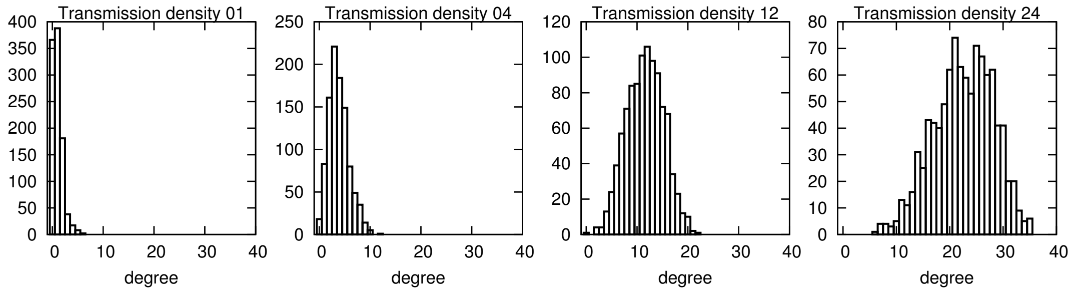


Fig. 6: Degree distributions for the static topology of 1000 nodes.

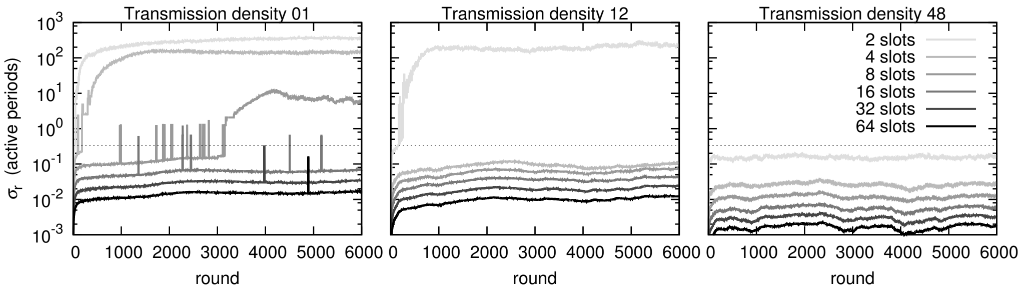


Fig. 7: Effect of active period length on synchronization maintenance. Random walk, 1000 nodes. From lighter to darker lines: active period of 2, 4, 8, 16, 32, and 64 slots.

sensitivity on random variations in timing, slot selection, etc.) across different runs, results in the spikes observed on the respective plots.

In the middle graphs, we show the results of testing our protocols on a larger static random deployment, this time with 1000 nodes. Increasing the scale of the network by an order of magnitude does not change the results by much. In fact, they look remarkably similar. In the larger network, however, a density of 12 is no longer sufficient for GMAC to maintain synchronization. The larger network size means that there can be more and larger sparse regions in the topology, balanced by regions of higher density elsewhere. Fig. 6 shows the degree distributions of this 1000-node static random deployment. For a transmission density of 12, there are a few nodes with degrees below four, notably one node without neighbors.

We can also see from the results that, even at a transmission density of 24 nodes, GMAC struggles to keep all nodes tightly synchronized. The periodic synchronization behavior at this density appears to be caused by a group of nodes weakly connected to the main component. A few "unfortunate" scheduling decisions leading to message collisions can allow these nodes to drift out of sync with the rest of the network, apparent as bumps in the plot. In both 100- and 1000-node topologies, it is only the transmission density that has any significant effect on the results.

Finally we look at GMAC's behavior in a network of mobile nodes in the right-most plots of Fig. 5. The difference in the performance between static and mobile networks is striking. It is clear that GMAC achieves far superior results on a dynamic network topology compared to static topologies. The median algorithm maintains the initial synchronization for all tested transmission densities, with the exception of the lowest setting of 0.25. Allowing the nodes to move has an effect similar to increasing the density, because it increases the number of neighbors a node will see in a given duration, effectively giving the median algorithm a chance to repair clock drifts.

As seen in all subplots of Fig. 5, the differences between σ_r and λ_r are the magnitude of the deviation, not the behavior over time. From the above results, we can clearly see that the global standard deviation follows the same pattern as the local standard deviation. This is expected, and for this reason, we present only the global standard deviation (that is, σ_r) from now on.

6.1.2. Active period length. As mentioned in Section 2.2, GMAC's default slot allocation strategy defines an active period of 8 slots. For scenarios where the anticipated node density is high, though, nodes may have to be configured with more active slots to prevent excessive collisions. Although sporadic successful transmissions are usually sufficient to maintain synchronization, excessive collisions may severely hinder an application's data transfer. It is, therefore, interesting to explore the effect of the active period length on synchronization stability.

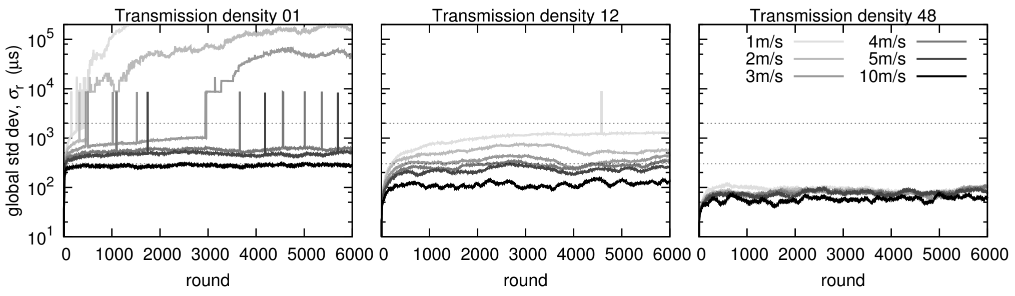


Fig. 8: Effect of node velocity on synchronization maintenance in 1000-node network with 8 active slots. From lighter to darker lines: 1, 2, 3, 4, 5, and 10m/s node speed.

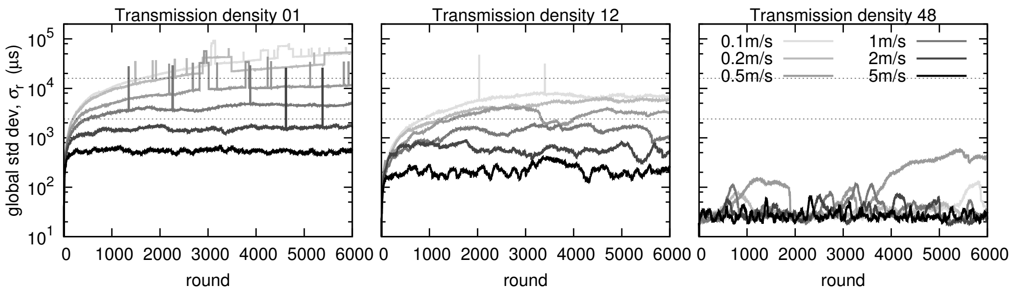


Fig. 9: Effect of node velocity on synchronization maintenance in 100-node network with 64 active slots. From lighter to darker lines: 0.1, 0.2, 0.5, 1, 2, and 5m/s node speed.

We carried out experiments for the random walk scenario of 1000 nodes. Fig. 7 presents the global standard deviation, σ_r, for active periods ranging from 2 to 64 slots. Note that we use 1 *active period* as the y-axis unit, rather than $1\mu s$, as the synchronization metric σ_r is meaningful only as a fraction of the active period. As explained in Section 5.4, a standard deviation equal to one third of the active period means that 99.7% of the nodes are loosely synchronized within an active period's time, that is, their active periods overlap at least a bit. Thus, we plot a dashed horizontal line at $y = 1/3 \times \textbf{active period}$, to visually distinguish loosely synchronized networks (below that line) to nonsynchronized ones.

As intuition suggests, the longer the active period (darker lines) the easier it is for nodes to stay in sync. This is a trend we consistently observe for all transmission densities we considered, namely 1, 12, and 48 neighbors per node on average. As a result, low active period configurations fail to maintain synchronization in sparse topologies.

Given these results, we adopt the default active period of 8 slots for the rest of our evaluation, resting assured that any higher value will only perform better with respect to stability.

6.1.3. Effect of node velocity. As node mobility has an evident (positive) impact on node synchronization, we inquire the specific effect of node speed to it. In the set of experiments presented in Fig. 8 we explore six different mobility scenarios, where all nodes move continuously at a fixed speed in random directions, based on the random walk mobility model. Nodes' speed is constant in each experiment, and ranges from 1 to 5m/s, as well as 10m/s. The three plots correspond to transmission densitites of 1, 12, and 48 average neighbors per node.

As expected, higher-velocity scenarios (darker lines) have a clear advantage over lower-velocity ones. As explained earlier, mobility allows nodes to contact a more diverse set of other nodes, compensating for a sparse topology. Essentially, we are witnessing a clear correlation between mobility and density, in the sense that both have a positive effect on maintaining synchronization. It is not the number of neighbors a node has at a given moment that determines synchronization, but rather the number of *contacts* a node can make per time unit, irrespectively of whether this is due to network density or node mobility.

Fig. 9 illustrates the effect of node velocity for a different network setting, which better approximates our real experiment presented in Section 7. First, we are dealing with a smaller network of 100 nodes. Second, GMAC is configured with an active period of 64 slots. Hence, the two dashed horizontal lines have been placed at $y = 16000\mu s$ and at $y = 2400\mu s$ to indicate synchronization thresholds for this longer active period. Finally, node speeds are *not* constant, but range from zero to a maximum speed, the maximum speed being 0.1, 0.2, 0.5, 1, 2, and 5m/s. The lower speeds better approximate a human crowd in a conference event. The plots illustrate the precise same effect as that of Fig. 8: the higher the node speed the easier the maintenance of synchronization.

6.1.4. Effect of network diameter. In this set of experiments we explore the effect of network diameter on maintaining synchronization. To limit the number of parameters, we focus on a network of 1000 nodes following the random walk mobility model. To allow for a fair comparison, we fix the playground area's size for all experiments, but we fluctuate its aspect ratio from 1:1 to 100:1. Table III lists the aspect ratios investigated and their associated dimensions. Each playground setting is annotated with the estimated diameter for each transmission density considered, computed based on the playground's diagonal length and the respective transmission range.

Table III: Playground aspect ratios investigated

Aspect ratio	Dimensions (m^2)	Approx. diameter ($hops$)		
		TX dens. 4	TX dens. 12	TX dens. 24
1 : 1	$1000m \times 1000m$	39	22	16
2 : 1	$1414m \times 707m$	44	25	18
5 : 1	$2236m \times 447m$	63	36	26
10 : 1	$3162m \times 316m$	89	51	36
50 : 1	$7071m \times 141m$	198	114	80
100 : 1	$10000m \times 100m$	280	161	114

We notice a clear effect of the network diameter on node synchronization. The longer the diameter (higher aspect ratio – darker lines), the worse the synchronization. This behavior comes as no surprise, as propagating (and healing) the effect of diverging clock drifts is expected to be harder when more hops have to be crossed.

6.1.5. Effect of broadcasting reliability. Finally, we explore synchronization stability in the face of unreliable communication. We reran the 1000-node random walk experiments, but this time inducing a uniform random packet drop rate of up to 80%.

Fig. 11 presents the results of these experiments, demonstrating the striking resilience of the median algorithm (Alg. 1) on packet loss. Although our induced packet drop rate reaches values as high as 80%, we observe that this has hardly any noticeable effect on synchronization stability, both for sparse and dense topologies. This is attributed to the fact that keeping nodes in sync requires only *occasional* communication among them to even out their clock drifts.

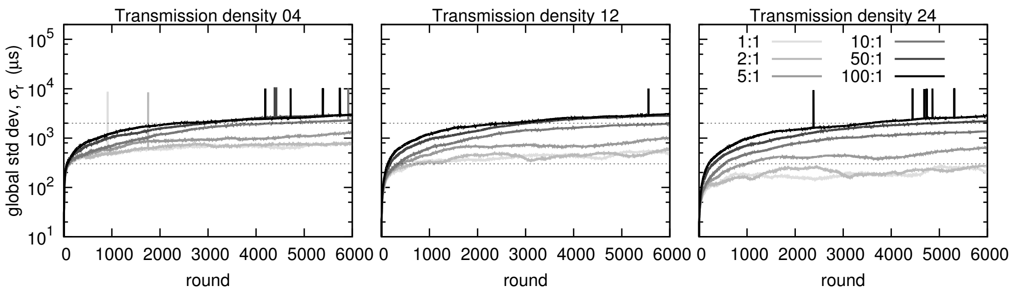


Fig. 10: Effect of network diameter on synchronization maintenance on a mobile (random walk) network of 1000 nodes. From lighter to darker lines: 1:1, 1:2, 1:5, 1:10, 1:50, 1:100 playground aspect ratio.

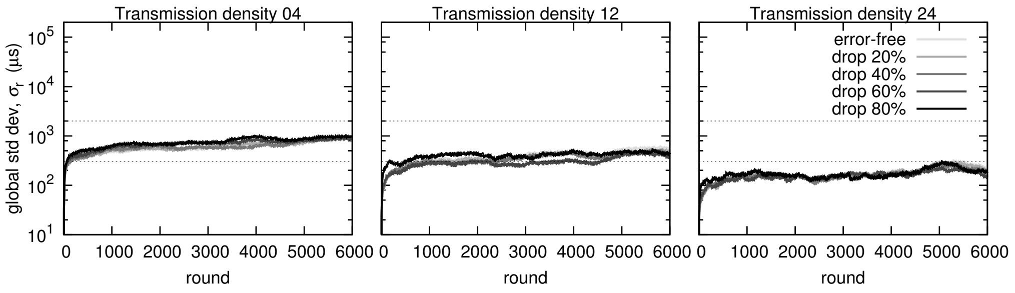


Fig. 11: Effect of *average broadcasting reliability* on synchronization maintenance on a mobile (random walk) network of 1000 nodes. From lighter to darker lines: 0%, 20%, 40%, 60%, and 80% packet drop rate.

On a side note, this also sheds some light on why our scenarios of high density still manage to keep in tight synchronzation, despite the increased number of collisions.

6.2. Detection

As we have chosen complete network convergence as our goal, we should ensure that detection of other synchronized groups happens quickly, but also with as little energy expenditure as possible. Here we evaluate both the active and passive methods of detection by artificially creating two already synchronized groups. We are interested in only two syncgroups because we want to eliminate the possibility of nondeterminism in GMAC's timing-based decision relation, $\overset{t}{\succ}$ (Section 4.2), obscuring our evaluation of the detection mechanism. The first syncgroup is composed of a single node, node 0, while the second syncgroup is composed of the rest of the nodes in the simulated network. Both groups start, independently, in the `SYNCHRONIZED` state. The large group begins executing at $t = 0.5s$, while node 0 begins at a random time $0s < t \leq 1s$. We choose a random start time for the singleton group in order to vary which group is superior according to $\overset{t}{\succ}$. In runs where node 0 starts up *before* the synchronized group, the large group will be forced to merge with node 0. In contrast, in runs where node 0 starts *after* the other nodes, node 0 must detect the other group and merge with it. As above, we perform our experiments using the same variety of static and mobile topologies, and at several different transmission power settings. Our main metric to evaluate detection will again be the standard deviation of round start times, σ_r.

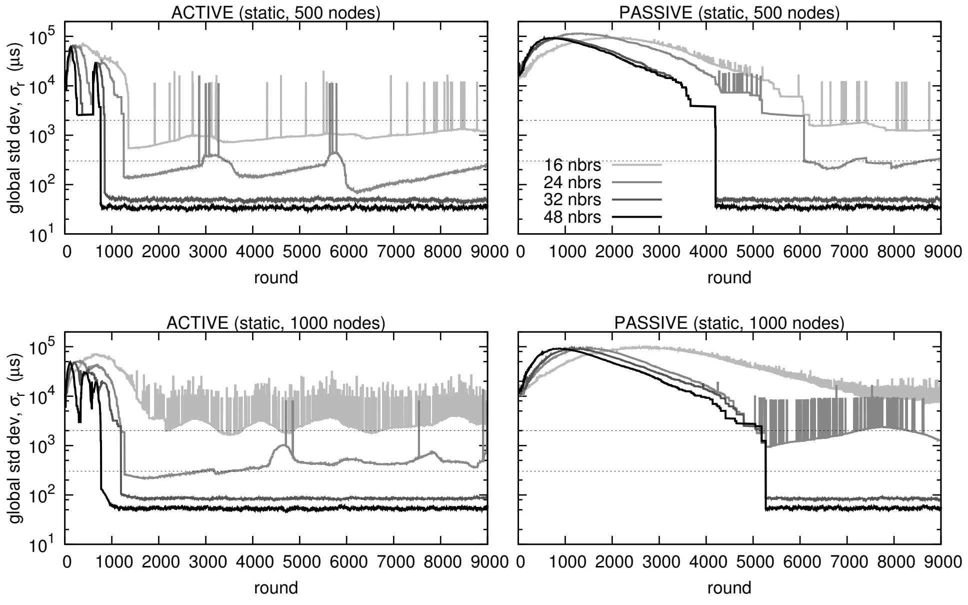


Fig. 12: Evaluating GMAC's detection mechanisms in static networks. From lighter to darker lines: transmission density of 16, 24, 32, and 48 neighbors per node on average.

Note that we carefully implemented active and passive detection to have (nearly) identical energy costs, so that we can compare the two purely on performance.

The two GMAC configurations we will examine here are Active and Passive. Both configurations succeed in synchronizing the 100-node topologies, both static and mobile, so we do not present those results here. We instead look at the 500-node and 1000-node networks. In Fig. 12, we present the results of the static topologies. We see on the left side that using active detection quickly leads to tight synchronization at the three highest density settings, but can achieve only loose synchronization at the lowest investigated density. Clearly density is a determining factor in not only whether synchronization will succeed or not, but also in what bounds can be achieved. Passive detection, on the other hand, takes several times longer to converge, while it does not reach equally good synchronization for the two least dense scenarios.

As in our evaluation of synchronization maintenance, detection performs significantly better too in the case of mobile topologies, seen in Fig. 13. As nodes move around the simulated area, they directly exchange messages with a much larger number of nodes than in a static scenario. This allows synchronization information to propagate via physical movement as well as by radio communication, and lends a strong performance benefit. Furthermore, the effects of transmission density are less pronounced in the mobile scenarios. Using active detection (left side of Fig. 13), GMAC synchronizes both the 500- and 1000-node networks within about 1000 rounds. Passive detection (right side of Fig. 13), while also successful, takes four to six times as long to synchronize the same network. Finally, mobility has also remedied the strange passive detection behavior where higher density topologies can take longer to synchronize than lower density ones.

Note that the observed detection performance appears to be very poor even in the best case of Active configurations, taking around 1000 rounds (~16 minutes) to reach synchronization. This is indeed the default GMAC behavior, due to the fact that each node is es-

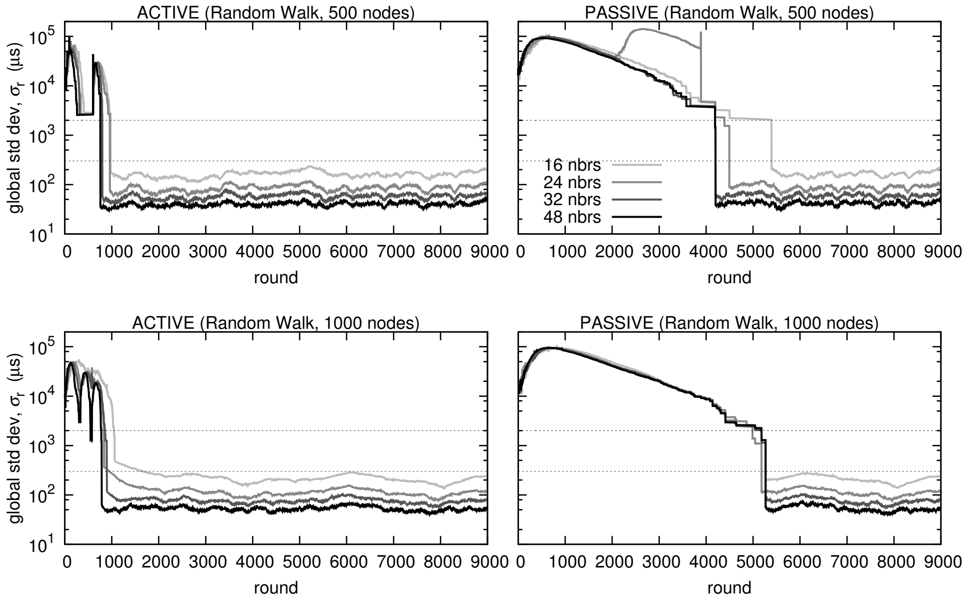


Fig. 13: Evaluating GMAC's detection mechanisms in mobile networks. From lighter to darker lines: transmission density of 16, 24, 32, and 48 neighbors per node on average.

sentially left to discover other syncgroups and merge with them on its own. Our proposed improvements leverage—among other points—the synchronization of existing syncgroups to speed merges up, bringing them down by one to two orders of magnitude to ~1 minute, as demonstrated in the following sections.

6.3. Decision

To examine the *decision* aspect of merging, we study networks under an asynchronous initialization. In these simulations, we create the conditions for a chaotic network start. All nodes start unsynchronized and must initially *detect* their neighbors in order to form local syncgroups. The better syncgroups, determined by the relation $\succ$, will continue to grow in size as nodes discover neighbors in superior syncgroups and *decide* to merge with them. By measuring what percentage of the nodes have synchronized (to a common active period) as the simulation progresses, we can see the effects of deterministic decisions made by the nodes. For each round, we count the percentage of synchronized nodes as described in Section 5.4. The GMAC configurations we will examine here are Active and Active+Cluster.

As we are interested in networks of large scale, we focus our evaluation of decision mechanisms on mobile 1000-node topologies. In Fig. 14 we compare side-by-side GMAC's performance based on the default timing-based decision relation $\overset{t}{\succ}$ (left), and the improved cluster tag-based relation $\overset{c}{\succ}$ (right). It is evident that the timing-based algorithm fails to converge, as nodes cycle between different syncgroups. The cluster tag-based algorithm, however, quickly converges the entire network to a single active period at all but the lowest transmission densities in Gauss-Markov topologies.

The failure of the reference-point group mobility trace to maintain the whole network synchronized in the long run is attributed to the fact that this mobility trace lends itself to physically isolated groups of nodes.

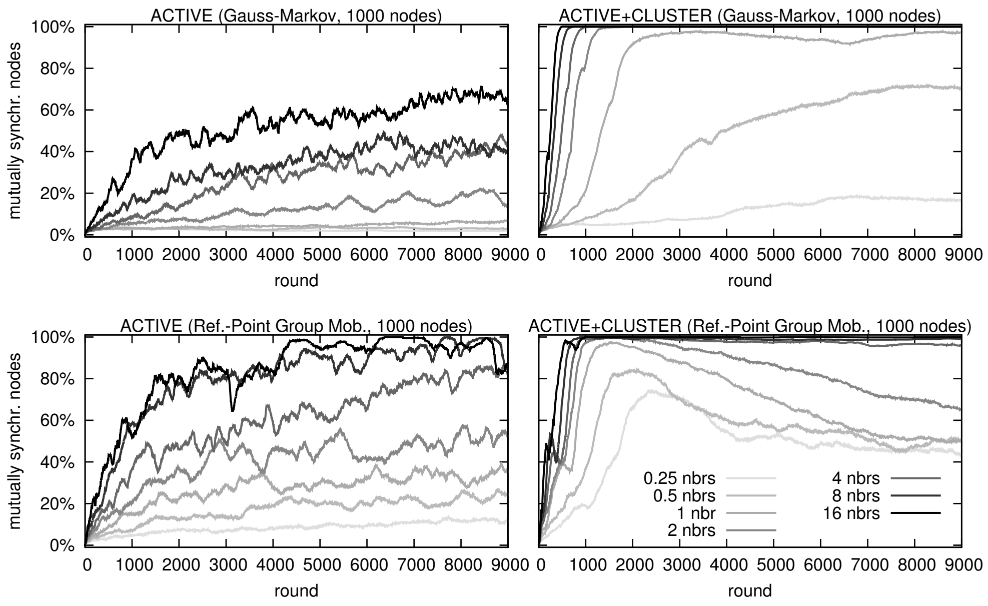


Fig. 14: Evaluating GMAC's decision mechanisms in mobile networks.

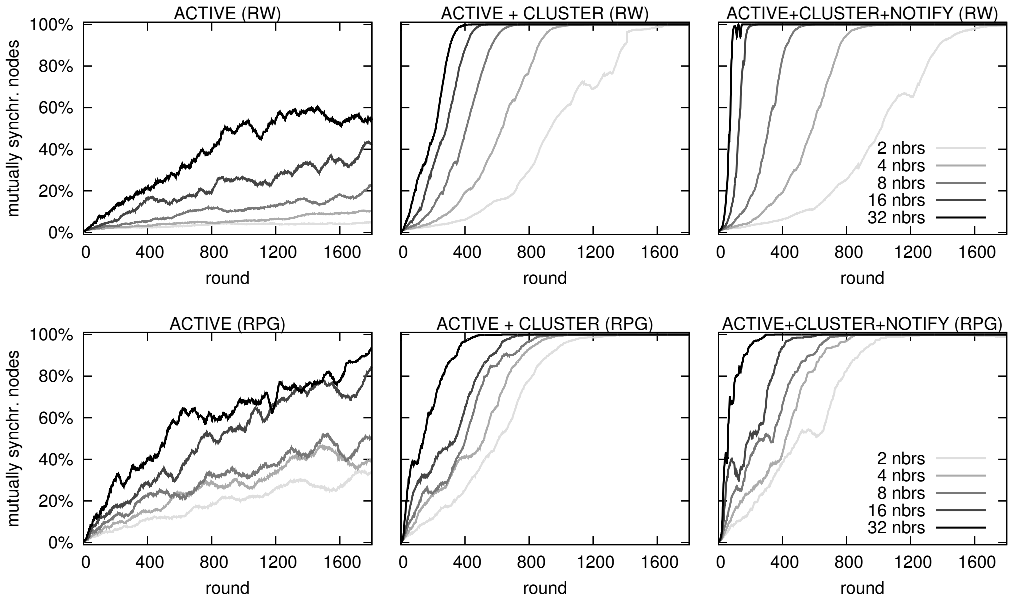


Fig. 15: Evaluating notification mechanisms in mobile networks.

To examine the *notification* aspect of merging, we will again make use of asynchronous initialization scenarios. We look at simulations similar to those as above, but this time with an eye to the performance of the merge messages optimization.

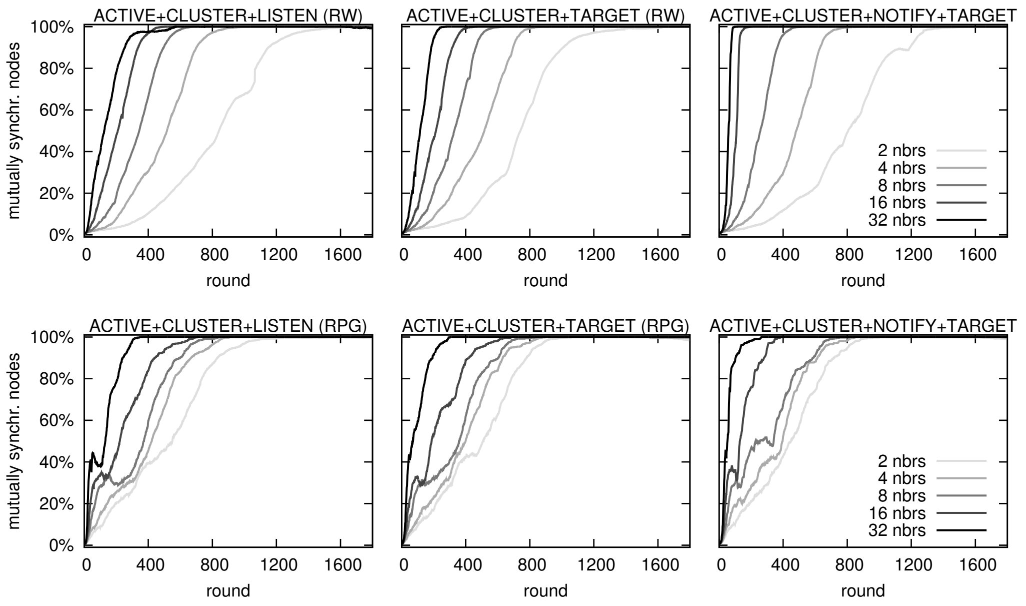


Fig. 16: A look at the performance of further proposed improvements to GMAC's detection behavior, 1000-node mobile networks

In Fig. 15, we see the performance of Active (for reference), Active+Cluster (without merge messages), and Active+Cluster+Notify (with merge messages). The top graphs depict the random walk mobility pattern we have seen throughout this section. The addition of merge messages (top right plot) allows the network to converge in an average of just over 100 rounds for a density of 32 neighbors per node, while the configuration without merge messages (top middle) takes more than three times as long. It is interesting to note that, with merge messages, the percentage synchronized line becomes almost vertical at high densities. This is because the merge messages greatly reduce the chance of nodes being "left behind" as the other nodes in their group detect a better group and merge into it. This effect is seen in the much more gradual slope of the results without any notification method on the left. The effect is also strongly tied to a node's transmission range, as the results for lower densities reflect little improvement. The bottom graphs show the behavior on nodes following the reference-point group mobility pattern. This mobility trace keeps nodes in tight physical groups that move in reference to a common point. The effect on the synchronization behavior is clear, as this pattern restricts interactions between nodes to mainly those in the same reference point group. Only when these groups cross paths are there opportunities for synchronization information to pass between them. The results for this mobility model also show a strong correlation with the network density, as the effect of the merge messages is more pronounced at high density. There is a cost to these merge messages, however, of an additional two bytes in the MAC message header, to store the offset of the new syncgroup.

6.4. Detection revisited

Two of our detection optimizations, listen-before-merge and targeted join messages, should be far more effective if combined with cluster tags. For that reason, we revisit the issue of detection here, this time, however, using an asynchronous start in order to allow for more syncgroups to be established. We study the behavior of ACTIVE+CLUSTER+LISTEN, ACTIVE+CLUSTER+TARGET, ACTIVE+CLUSTER+NOTIFY+TARGET to see whether we can improve syncgroup detection even further.

The performance of these configurations is depicted in Fig. 16, which can be seen as an extension to Fig. 15 presented above. We find the results of the listen-before-merge optimization (left plots) to be disappointing, especially considering the energy cost of this behavior. Listening for an entire round costs about the same as sending 600 join messages, quite expensive indeed. The performance is comparable, but inferior, to that provided by targeted join messages. However, the targeting does not imply any additional radio time and thus no additional energy cost, making it the superior choice.

Targeted join messages (middle plots – Section 4.1.4), however, offer a notable performance benefit, particularly at high transmission density. The reason for this is that this optimization balances out the asymmetric decision behavior. That is, nodes would normally ignore messages from inferior clusters. Targeting join messages allows us to effectively double the detection probability by making the process symmetric. As we have seen throughout this section, the effect of the optimizations is limited by the density of the network. The better connected the network topology, the less benefit afforded by the targeting. The results for the reference point group mobility (lower plots) also show a stronger performance increase at high density when using the targeting behavior. However, the performance benefit quickly diminishes or disappears entirely at lower densities.

The right plots of Fig. 16 show the performance of the combination of merge messages with target joins. This is clearly the optimal of our proposed configurations, therefore it constitutes the recommended configuration for deploying sensor networks based on GMAC.

6.5. Larger networks

As we have emphasized several times, our chief interest is scalability. As sensor nodes continue to fall in price, very large-scale networks will become economically feasible. Thus, we end the discussion of our simulated results focusing in that direction. The only significant difference between these experiments and those described earlier is in the number of simulated nodes. We still look at the same mobility patterns, but here we observe the behavior of 4000 nodes. We will again look at an asynchronous initialization, since this type of scenario presents a worst-case for network-level synchronization. Because running and processing these simulations are quite demanding, we look at only three transmission density settings (2, 8, and 32) and two GMAC configurations (ACTIVE+CLUSTER and ACTIVE+CLUSTER+NOTIFY+TARGET).

In Fig. 17, we present results for all three mobile topologies. We can see that the results for the Gauss-Markov mobility pattern (top) and those for random walk (middle) are very similar. Both of these mobility patterns lead to network topologies with relatively uniform node density. This can be seen in the results, as the percentage of synchronized nodes follows a smooth line. The results of the reference point group mobility pattern (bottom) present a more bumpy and irregular pattern. As mentioned previously, this type of mobility manifests a much less uniform node density as groups of nodes move together around their common reference points. Nevertheless, both tested configurations manage to converge all three simulated deployments at transmission densities 8 and 32. The uniformity of the Gauss-Markov and random walk topologies provide the best setting for the ACTIVE+CLUSTER+NOTIFY+TARGET configuration to outperform the simpler ACTIVE+CLUSTER configuration. The left-hand graphs show the behavior of ACTIVE+CLUSTER, which reaches 100% synchronization in about 500 rounds at density 32 and 1000 rounds at density 8, on both topologies. The right-hand side shows that the addition of merge messages and join message targeting reduces the time required to about 80 and 400 rounds, respectively. Similar, though less dramatic, improvement is seen with the reference point group mobility pattern as well, but GMAC's performance is limited by the lower connectivity of the net-

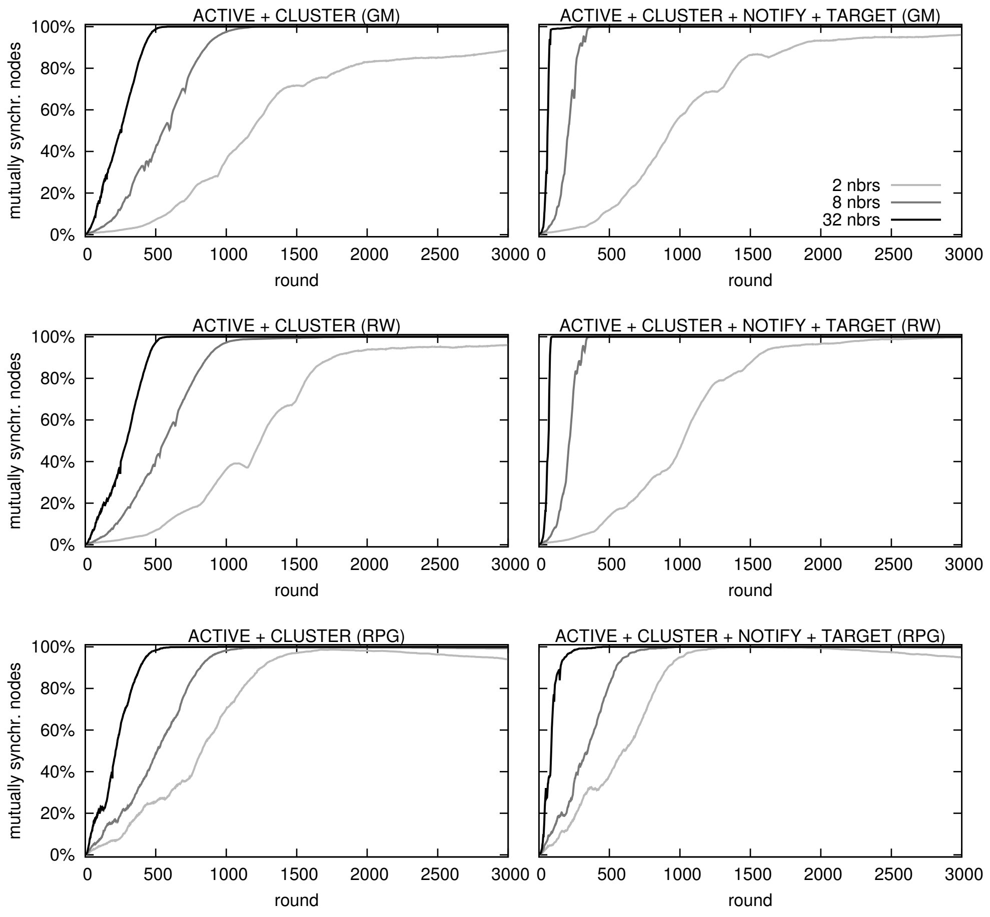


Fig. 17: GMAC at very large scale: 4000-node mobile networks.

work. Finally, we can see that our proposed improvements make little if any difference at the lowest examined transmission density, regardless of the topology.

7. REAL-WORLD EXPERIMENT

In the course of our research, we have performed a number of real-world experiments. As our primary goal is to design a set of networking protocols suitable for large-scale mobile wireless sensor networks, real-world testing and experimentation are essential. Because we are interested in highly scalable solutions, we must necessarily test our algorithms with hundreds of nodes. Ideally we would be able to perform experiments with thousands of nodes, but practicalities and costs of running experiments with thousands of people makes such an approach at this stage infeasible. As such, we have performed much of our investigation via simulation. Part of the challenge of performing real-world experiments in the domain of sensor networks lies in the effort involved in planning, preparing and executing experiments involving many individual wireless sensor nodes. Each device needs to be independently charged, programmed and tested. After the experiment, we must re-collect all the nodes, properly shut them down, and fetch the recorded log from each node individually. The entire process is time consuming, cannot be automated, and is often plagued by unforeseen problems, as discussed in [Langendoen et al. 2006].

Nevertheless, in October of 2012, we executed an experiment at the ICT Open conference which took place in the World Trade Center in Rotterdam, The Netherlands. The conference lasted six hours and drew about 220 attendees, approximately 120 of which volunteered to participate in our experiment. Each of these volunteers was issued an electronic badge node, consisting of a MyriaNed device, a plastic case, and a lanyard to allow it to be worn around the neck. The participants were to wear the device for the duration of the conference, while we provided real-time visualization of the topology of the network and, afterward, an analysis of the "social mingling" of the participants. The goals of this experiment were to perform a large-scale experiment successfully and visibly, to demonstrate our work to the Dutch research community, and, most importantly, to capture accurate timestamps for analysis of real-world synchronization.

In this section we will describe the nodes, equipment, and software we used to perform this experiment, explain the measurements we took during the experiment, describe the preparation and setup of the venue, and finally present the results we obtained.

7.1. Nodes and application

In this experiment we used MyriaNed V3 devices (see Appendix A) primarily as *active* nodes (sometimes called **badge** nodes) in the network. An active node is worn by an attendee of one of our experiments and is a full participant in the peer-to-peer GMAC network. The secondary use of the MyriaNed nodes is as *passive* nodes, or **sniffer** nodes. A passive node does not fully participate in the network and never broadcasts any messages. The passive nodes are used to observe, in real-time, the message traffic generated by the active nodes. The data from the sniffer nodes can be routed to a central PC in order to visualize the behavior of the network of badge nodes.

7.1.1. Active Nodes. The active nodes in this experiment executed the GMAC protocol using the **ACTIVE+CLUSTER+NOTIFY+TARGET** configuration as described in Section 5.1. The only difference is that the active nodes used a longer active period, $s_{active} = 64$. This was chosen because the transmission range of the nodes is about 15 meters and so some smaller conference rooms could result in 1-hop neighborhoods, too large for only 8 active slots. Using this extended active period reduces the energy efficiency of the nodes, but greatly reduces the likelihood of a failed experiment due to too many collisions.

For this experiment, we designed a new application, called *NeighborReport*:

— The application takes advantage of GMAC's gossiping protocol to share neighborhood data items throughout the network.
— Each neighborhood data item includes the creator's node ID (C), the round number in which the neighbor nodes were observed (r), and up to five node IDs that were neighbors of C in round r. Node IDs are stored as 1-byte integers and round numbers as 2-byte integers, so one neighborhood data item requires eight bytes.
— A node will include both a data item representing a sample of its own neighborhood for the previous round and a recent data item from its cache.

The inclusion of a second data item serves to increase the speed at which data items spread through the network. It also allows the sniffers to gain insight into what is happening in parts of the network that are not directly within their range. Data items created by nodes far away can be carried into the range of a sniffer either by node mobility or multi-hop propagation.

7.1.2. Sniffers. In order to inspect the operation of the network in real-time, we utilize a second network of passive nodes. Each sniffer node is controlled by a host computer via USB. The host computer logs the data and forwards it over Ethernet to a central PC that runs

Table IV: GMAC packet data

Name	Type/Size	Description
transmit slot	12-bit int	slot index selected for transmission
magic number	20-bit int	constant magic number for error detection
mac-level CRC	2-byte int	cyclic redundancy check for error detection
cluster tag ID	2-byte int	sending node's cluster tag id
cluster tag epoch	1-byte int	sending node's cluster tag $epoch$
Round number	2-byte int	sending node's round number
merge tag ID	2-byte int	
merge tag epoch	1-byte int	*iff non-zero*: notification of superior cluster (id, $epoch$) with specified slot offset
merge offset	2-byte int	
app data	16 bytes	application data
TOTAL	32 bytes	

a visualization application. By observing the stream of gossiped data items, the visualizer can reconstruct and display the topology of the network. In this experiment, we had only three sniffer nodes, resulting in coverage of approximately only 30% of the main hall of the conference venue.

In addition to the MyriaNed nodes that we use as sniffers, we also employ a raw radio-frequency sniffer device, called an *RF sniffer* for short. These devices use an FPGA and high-speed processor to monitor a specified frequency range. All radio-frequency data in this band is captured, and the device attempts to parse packets from the incoming stream of data. This is accomplished by matching the known header in GMAC packets, as well as checking each message's packet-level CRC to make sure the message is correct. The most important aspect of this device is its high-precision clock, which allows the device to timestamp received packets with a granularity of $1\mu s$ (about 30 times finer-grained than the resolution of the clocks on the V3 nodes).

7.2. Measurements

We captured the measurements presented below from data recorded by the RF sniffer described above. The RF sniffer allows us to capture messages sent by the active nodes, along with a high resolution timestamp indicating the time the message was received. The basic packet format used by the active nodes is shown in Table IV. RF sniffer timestamps are recorded as the number of microseconds that have elapsed since the device was started. The RF sniffer device is connected via USB to a host PC that runs software designed to interpret and parse the raw radio frequency data and log the parsed packet and timestamp to disk. Much of this processing will eventually happen on the RF sniffer node's internal FPGA, as the software for this device matures.

Each message contains the sending node's transmission slot number. Because a node always broadcasts its message a fixed number of clock ticks after the beginning of its transmission slot, we can compute the exact time the sending node started that round by subtracting $9 + (28 \times TX\_SLOT)$ V3 clock ticks (approximately $30\mu s$ each) from the packet reception timestamp, where 9 ticks correspond to the first slot's initial guard time, and 28 ticks are the duration of an entire slot (see Fig. 1). We call this calculated time the node's *slot-0 time*, and it is the real-world equivalent of the timestamp logged by the nodes in the simulator. By comparing the slot-0 times of all nodes from which the sniffer logged a message on a per-round basis, we can evaluate how the network synchronization proceeds throughout the experiment.

7.3. Setup

In this experiment, the layout of the venue prohibited us from achieving full coverage with our sniffer nodes. The lobby of the World Trade Center is a wide, open-air area with extremely high ceilings and very few locations to setup the sniffer nodes and laptops to run them. As an alternative, we used a demonstration area for placing our sniffers, which proved to be convenient as this area was also used during the lunch and other breaks.

As with most conferences, the day was broken up into presentation sessions interrupted by short break periods. This scenario provides a difficult test-case for any synchronization mechanism, since groups of nodes will be physically separated and unable to communicate for large portions of the day. During these periods of separation, groups of nodes in the same room will tend to stay synchronized with each other, but may drift apart from the other nodes in different rooms. When the participants (and the nodes) come together again during the break periods, the synchronization protocols must allow a node to detect other unsynchronized nodes in its vicinity, decide whether to merge with them or to wait for them to merge with it, and finally to notify its own neighbors of any decision. From an experimental perspective, a multi-session conference is therefore an ideal scenario for testing synchronization mechanisms.

7.4. Results

Although we were not able to convince a majority of the conference's attendees to wear one of our electronic badges, we considered the experiment to be a success. We had about 120 active nodes participating in a wearable sensor network, which was our second largest experiment to-date. Most importantly, we were able to capture synchronization data during periods of socialization and mingling, showing us how GMAC performs in real-world scenarios with high mobility.

We present an example of the timing data recorded during the ICT Open experiment in Fig. 18. Along the x-axis we show the round number, grouping received messages based on the round number in which they were sent. For each round number, we compute statistics about the set of messages received during that round, just as with our simulated networks. Because the nodes have a limited transmit range, a sniffer is able to capture only those packets broadcast nearby, rather than all messages as in a simulator. In the bottom plot, we show the number of logged messages for each round number. In the top plot, we show the standard deviation of the slot-0 times of the senders, in microseconds. This is essentially the σ_r measurement from Section 5.4. The dashed horizontal lines correspond to the synchronization thresholds explained in Section 5.4, adjusted to reflect a 64-slot active period. The results presented here represent the entirety of the conference event. As can be seen in the received packet counts, there were periods with very few nodes in the main hall (e.g., during the breakout sessions) and periods with many nodes in range of the RF sniffer (e.g., coffee breaks between presentations). In this graph, we do not present rounds in which zero or one packet was received, as we cannot compute a standard deviation. Nevertheless, there were over 14000 rounds with two or more messages logged by the RF sniffer, covering almost four of the six hour event.

These results are extremely positive, demonstrating that synchronization is working properly. The results from this experiment indicate that nodes are generally synchronized to within a standard deviation of $< 10,000\mu s$ (about 12 communication slots), almost all of the nodes should be synchronized to within $30,000\mu s$ (about 35 slots). Since we used an active period of 64 slots ($T_{active} \approx 54,000\mu s$) for this experiment, this strongly indicates

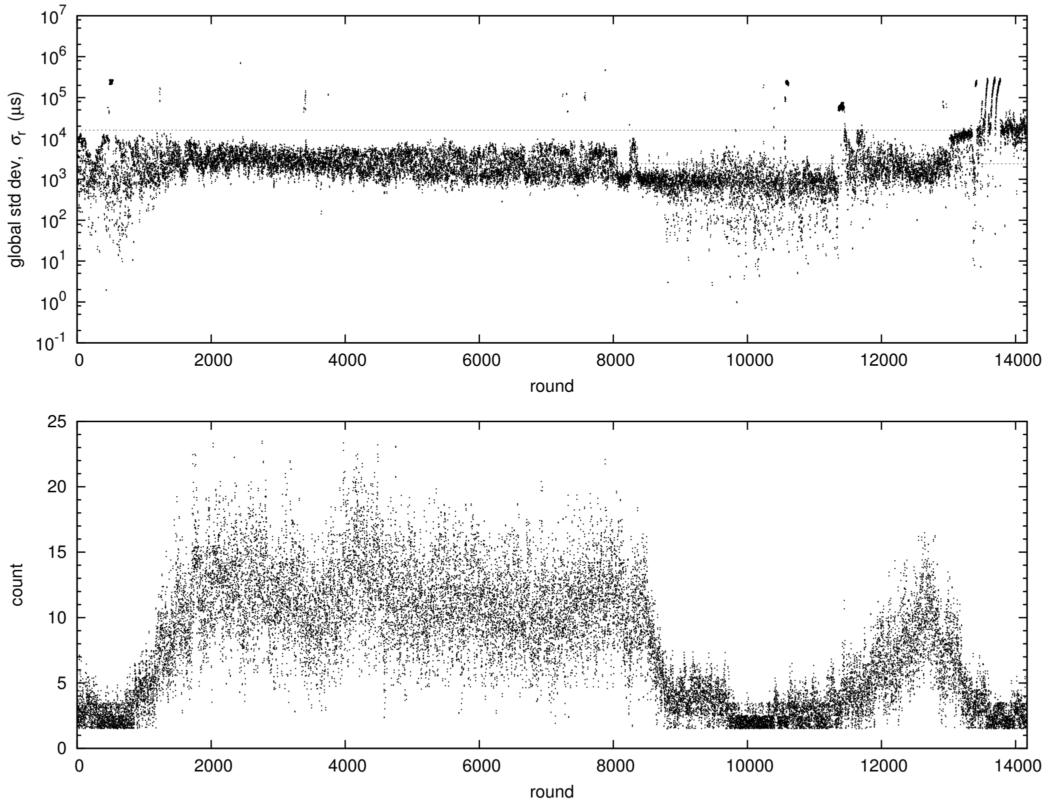


Fig. 18: Timing results from the ICT Open experiment

that the active periods of all nodes overlap to a large degree, which in turn means communication amongst all participating nodes should be possible.

A second positive outcome of these results is that they validate our simulations, as they are in accordance with our simulated experiments for a small network of slowly-moving nodes with an active period of 64 slots, presented in Fig. 9. Despite inevitable precision errors in timestamping (by the RF sniffer), and approximated modeling of radio propagation, interference, reflections, etc., our simulations closely reflect the real-world experiment measurements. This confirmation of the accuracy of our many simulated experiments is a very important step in this research.

Finally, it should be noted that there are some outlying data points, indicating rounds where all nodes were merging after being separated. The results show that these periods are brief and nodes then return to global synchronization, just as seen in our simulations. We also are quite confident in these results because the synchronization appears to be most stable and reliable in the periods where there are many observed packets per round, representing ten to twenty percent of the total nodes in the network.

8. RELATED WORK

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.

Nevertheless, there is a wide range of work related to ours, which we subsequently present organized in the following main families.

8.1. Synchronous, contention-based protocols

In this group, protocols split time in discrete synchronous slots, and let nodes compete for each slot, without imposing any specific schedule.

There are two classic MAC-level protocols that are particularly relevant for our discussion. S-MAC [Ye et al. 2002], one of the main representatives of *slotted* access protocols, divides time in fixed-length slots of 1-3s and uses a 300ms active period (equivalent to a duty cycle of 10% to 30%), during which nodes compete for the channel using carrier-sensing to avoid collisions. T-MAC [Van Dam and Langendoen 2003] improves upon S-MAC by adding adaptivity to traffic. Active nodes time out if they hear no traffic during a brief contention phase at the beginning of each slot, drastically reducing energy use in idle networks. In both S-MAC and T-MAC, when a new node joins a network it listens for at least the duration of a whole frame 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 goes against our goal of fixed energy consumption and predictable lifetime. Most importantly, both protocols do not address 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, it constitutes a major shortcoming.

SCP-MAC [Ye et al. 2006] 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 authors implicitly assume a set of nodes that operate with tight synchronization. Thus, the issue of merging independently synchronized "virtual clusters" to a common schedule is not tackled in SCP-MAC. It is precisely on this problem that the bulk of this paper is focused. As mentioned previously, the existence of multiple separately synchronized clusters can be detrimental to internode communication, particularly in the presence of a high degree of node mobility.

8.2. Asynchronous protocols

These protocols follow an entirely asynchronous approach, without requiring clock synchronization or time slots, while they still strive for a very low duty cycle.

The authors of [Cattani et al. 2014] take a radically different approach to keeping duty cycles low, by introducing SOFA, a reactive MAC protocol that circumvents the need for synchronization. Nodes wake up periodically, yet asynchronously, and *listen* for a tiny fraction of time. To send a message, a sender transmits repetitive advertisements, until a node that wakes up as a result of its standard periodic operation responds with an acknowledgment. Then the sender unicasts the message to that node. An interesting property of SOFA is that it favors dense networks, as the sender's expected advertising time is shorter. Note that the sender has no control over who the receiver will be, essentially implementing *anycast* semantics, applicable in gossiping protocols.

8.3. Centralized TDMA

These protocols split time in discrete slots and rely on a master node to impose a TDMA schedule that *deterministically* assigned slots to specific nodes to avoid contention.

Mobile LMAC, presented in [Mank et al. 2007] and [Mank et al. 2008], removes assumptions about static topologies and uses gateway nodes to bootstrap synchronization. The proposed merge protocol comes close to ours with respect to making a decision concerning 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 [Arumugam and Kulkarni 2005] an algorithm is presented that deterministically establishes a TDMA schedule by a gateway node circulating a token. However, no attention is paid to merging clusters and keeping them synchronized, as nodes are assumed to be de facto synchronized.

WirelessHART [Song et al. 2008] is an industry-backed TDMA-based wireless mesh networking technology for real-time process control. Like GMAC, it is based on fixed TX slots, and achieves network-wide synchronization, which however is achieved through a master-slave protocol relying on ACK responses. The most notable difference is that its operation is based on a central network manager. WirelessHART is a fully-fledged wireless sensor framework, featuring (in addition to node synchronization) reliable streaming, large data transfer, channel hopping, and security. A number of WirelessHART-based industrial products have been developed, by companies such as DustNetworks (with their SmartMesh product line), Pepperl+Fuchs, Endress+Hauser, Nivis, etc.

8.4. Distributed TDMA

These protocols aim at a similar deterministic schedule as the protocols of the previous group, however they achieve it by means of a decentralized algorithm that involves all participating nodes.

A method of *desynchronizing* the active periods of participating nodes is presented in [Degesys et al. 2007]. With the presentation of DESYNC, the authors present a novel approach to scheduling the active periods of participating nodes. By spacing the active period of each of N neighbors equally within a total period of T, the algorithm will dynamically ensure that each node gets an uncontested active period of T/N for communication. DESYNC is not well suited to our needs for several reasons. First is that the algorithm assumes all participating nodes are arranged in a one-hop network, while we focus on multi-hop topologies. Second, the convergence time of the algorithm takes $O(N^2)$ rounds, significantly longer than our algorithms and certainly too slow for a network of $N = 1000$ nodes. Finally, though the authors discuss the possibility of nodes entering/leaving the synchronized group, mobility is not specifically addressed.

The protocol presented in [Ma et al. 2009] focuses on sleep scheduling and tries to schedule contiguous active slots. Their work depends on a *fixed* topology for their algorithm to converge and to be efficient, and is, thus, not appropriate for the mobile scenarios we target.

The algorithm proposed in [Cidon and Sidi 1988], allows a multihop network of N nodes to dynamically agree on a conflict-free TDMA schedule. However, it requires $O(|N|)$ slots per round, which renders it inappropriate for large values of N, i.e., scenarios involving thousands of nodes.

In SS-TDMA [Kulkarni and Arumugam 2006] the authors propose 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.

In DISCO [Dutta and Culler 2008] nodes pick a pair of prime numbers such that the sum of their reciprocals is equal to the desired radio duty cycle. Each node increments a local counter with a globally-fixed period. If a node's local counter value is divisible by either of its primes, then the node turns on its radio for one period. This protocol ensures that two nodes will have some overlapping radio on-time within a bounded number of periods, even if nodes independently set their own duty cycle. Once a neighbor is discovered, and its wakeup schedule known, rendezvous is just a matter of being awake during the neighbor's next wakeup period, for synchronous rendezvous, or during an overlapping wake period, for asynchronous rendezvous. As such, the main difference from GMAC is that it sets a provable upper bound on how long discovery can take, unlike GMAC in which it can theoretically take indefinitely.

8.5. The 802.11 family

These protocols aim at improving synchronization in multi-hop 802.11 networks.

In [Liu et al. 2005], the authors describe a method for merging clusters in multi-hop 802.11 ad hoc networks, in contrast to the more common solution of bridging the clusters. Their method is based exclusively on the passive listening method (extensively described in Section 4.1). There are no details on the merge process itself, presumably nodes simply "jump" to their new schedule during the merge (i.e., without notifying their neighbors). As seen in Section 6.2, our experiments indicate that active detection (used by GMAC) consistently outperforms passive detection, assuming equal amounts of energy are spent in both cases.

8.6. Clock synchronization in the face of clock drifts

The focus on these protocols is on maintaining clock synchronization in the face of drifting clocks.

The issue of clock synchronization in the face of clock drifts is addressed in SMART [Tjoa et al. 2004]. Although this paper is an inspiration for the synchronization algorithm adopted in GMAC, it does not deal with the orthogonal problem of merging clusters with non-overlapping schedules, neither does it consider duty cycling.

In [Mirollo and Strogatz 1990], the authors present a method of establishing and maintaining internode synchronization inspired by pulse-coupled biological oscillators. This paper is an inspiration for our own work, and experimenting with such techniques is planned as part of our future work. Biologically-inspired algorithms, such as this one, fit well into the theme of our research because it utilizes *local* decision making, maintains very little local *state*, is not dependent on specific *anchor* or *head* nodes, and is extremely scalable. Because the GMAC was built to have a modular synchronization component, we aim to experiment with replacing the current median algorithm with something similar to this pulse-coupled oscillator technique. In our investigations thus far, the *maintenance* of synchronization has been less problematic than the *merging* of synchronized clusters, and thus has not yet received as much attention.

The authors of [Li and Rus 2006] present three different methods for global clock synchronization. The first, or "all-node-based synchronization" requires the nodes to establish a cycle (path) through the network that passes through each node at least once, which is obviously impractical (if not impossible) for large-scale mobile networks. Their second version, known as the "cluster synchronization algorithm" operates like the first, but requires that the nodes in the network first elect cluster-heads which are then synchronized amongst each other, before the cluster-heads are finally used to synchronize the individual nodes within each cluster. Once again, node mobility renders this method difficult to use as we can no longer assume that a node is still within range of its cluster head by the time synchronization messages are exchanged. Their third algorithm is called "synchronous diffusion" and maintains synchronization in a manner similar to the GMAC median algorithm, but nodes synchronize to the *mean* value of their neighbors rather than the *median* value. Their simulated results are also similar to ours, showing 1000 nodes synchronizing within a few hundred rounds. An important distinction with our own work is that this paper does not focus on dynamic network topologies, though they state that their algorithm can adapt to node mobility.

8.7. Application-level synchronization

Protocols in this group address clock synchronization on sensor networks at the application layer, that is, the capability of nodes to communicate is orthogonal to their synchronization state. Consequently, duty cycling as well as detection and merging of different clusters are not applicable them.

Timing-sync Protocol for Sensor Networks (TPSN [Ganeriwal et al. 2003]) and Flooding Time Synchronization Protocol (FTSP [Maróti et al. 2004]) rely on creating a spanning tree over the whole network, stemming from a globally elected *root* node. The cost of leader election and tree building make such solutions unsuitable for high diameter and/or mobile networks.

The cost of synchronization is confined in [Pussente and Barbosa 2009] by piggybacking synchronization information on existing application traffic and by proposing a completely distributed solution, free from the expenses of centralized coordination. Nevertheless, this solution assumes a reliable communication channel, which is unrealistic in general and particularly when duty cycling is in place.

Reference Broadcast Synchronization (RBS [Elson and Estrin 2001]) offers single-hop synchronization to within a few microseconds (tighter bounds than we achieve here) in addition to multi-hop synchronization with degraded accuracy. Furthermore, RBS has been implemented on a number of different hardware and radio platforms. An interesting question is how RBS would perform in the large-scale mobile networks we target. Participating nodes must maintain state (timing data) on broadcasting nodes, which is later exchanged with other nodes in the network.

8.8. Miscellaneous

The authors of [Schmid et al. 2010] take duty cycling a step further. In addition to duty cycling the radio chip, they also duty cycle the high-frequency (and therefore, power hungry) clock itself, using a second clock of much lower frequency. However, their protocol dictates that one node be a "reference" node, which has access to a high precision time source (e.g., GPS). Our protocol assumes no such reference node, and operates in a completely decentralized manner.

Quorum, described in [Tseng et al. 2002], does not use a synchronized sleep schedule like GMAC, and nodes view time as a series of n^2 beacon windows. Using a matrix nodes select a set of O(1/n) beacon windows in which to transmit, and their protocol guarantees that another node will be awake and listening during at least one of those intervals.

9. CONCLUSIONS AND FUTURE WORK

The main contribution of this paper is a thorough examination of MAC-layer synchronization in ultra-low duty cycle, large-scale, mobile (and static) networks. In order to fully evaluate the problem, we examined the two distinct subproblems of network-level synchronization through numerous simulations and a large-scale mobile deployment. The results of these various experiments show that both problems *are* solvable, and our methods can be used to achieve remarkably low duty cycles, even with relatively inaccurate clocks. In addition, GMAC's design ensures energy is used at a fixed rate, which allows for very accurate predictions of network lifetime. And, perhaps most importantly, we have demonstrated that GMAC is not only capable of synchronizing *all nodes* in a network so that they share a common active period, but also of doing so in a completely decentralized manner. Removing the need for special "cluster-head" nodes makes planning and deploying a sensor network simpler and less costly.

We investigated the first aspect of MAC-layer synchronization, maintenance of existing synchronization between groups of nodes, by simulating networks of initially-synchronized nodes. We varied the simulated transmission range in order to see the effects of neighbor density on synchronization maintenance. We found that the simple *median* algorithm was able to maintain synchronization in a mobile 1000-node network with average densities as low as one node per transmission area. Static networks require a much higher transmission area (about 16 nodes per transmission area) in order to maintain synchronization, due to the network being partitioned into multiple subnets at lower levels. The results show that mobility can be helpful in sparse networks by routinely introducing new neighbors, which matches well with GMAC's decentralized, gossip-based nature. With regard to synchronization maintenance, we can conclude that the median algorithm is both simple and functional, but still has room for improvement. The idea of making timing adjustments during periods of isolation based on an approximation of a node's clock frequency offset could be a strong addition to the protocol, but an inaccurate method of approximation is likely to be worse than doing nothing at all.

The second aspect of network synchronization, merging separately synchronized groups of nodes, has been split into three orthogonal subproblems. We investigate each of the three subproblems (detection, decision, and notification) using different mobility patterns and starting conditions. We looked at two methods of detection, active and passive, and active detection outperformed passive detection by a significant margin in all scenarios. We attribute this to active detection's ability to detect multiple neighbors with a single broadcast, as described in 4.1. 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 and jeopardizes our goal of lifetime predictability. Performance is even further increased with the technique of *targeted join messages*, effectively doubling the detection rate. Regarding merge decisions, the use of *cluster tags* strongly improves the chances of eventual synchronization. We have demonstrated their efficacy in mobile networks ranging from 100 to 4000 nodes, showing the importance of deterministic merging decisions. Finally, our proposal of using a header field for notifying neighbors of local merge decisions can drastically reduce the time for a network to reach a synchronized state, by as much as a factor of eight on our 4000-node topology. These small modifications to GMAC's current behavior radically increase its suitability for large-scale mobile networks. The key insight is that as synchronized groups build up, the merge messages allow GMAC to leverage an inferior group's existing synchronization to rapidly merge *whole syncgroups*, not just individual nodes. Combined with the total ordering provided by cluster tags to solve the problem of which group to merge with, large and complicated networks can be synchronized in just a few minutes.

In the end, the most clear conclusions of our simulations is that the achievable level of synchronization is dictated by two things: density and mobility. The underlying issue is propagating synchronization information throughout the entire network. With very low transmission densities, a node will have infrequent communication with other nodes which will greatly limit the opportunities for disseminating synchronization data. Conversely, high transmission density means that this data can move a great distance during each round. Similarly, mobile scenarios generally facilitate network synchronization because the required information can be propagated faster. This occurs because nodes physically move around, carrying their data with them. Both of these factors, density and mobility, are aspects of the overall network topology. It is the topology that will determine whether full network synchronization will succeed, our goal is to ensure that individual network subnets remain synchronized and discover each other as rapidly as possible when they physically connect. To that end, active detection with targeted join messages, cluster tag-based decision and notification via merge messages results in *significantly* faster synchronization convergence.

The direction of this research was inspired by the failure of network synchronization during our earliest real-world deployments. However, our most recent deployments (e.g., the ICT Open experiment presented in Section 7) indicate that the conclusions of our investigations using simulation are applicable to the physical nodes and do indeed have the potential to work in large-scale wearable sensor networks.

There remain many interesting avenues to pursue. An investigation of significantly lower duty cycles is possible. For example, in applications that are not latency-sensitive we could increase the duration of a round. Simulations we have run show that $T_{round} \geq 5s$ are achievable, reducing the duty cycle to $\tau \leq 0.15\%$. By reducing transmission guard times, improving synchronization maintenance and using more accurate clocks, we could possibly achieve duty cycles less than 0.01%. Finally, creating a hybrid protocol using GMAC's active period as a mechanism to schedule *application-specific active periods* during GMAC's inactive period would allow for execution of a wider range of applications which require lower latency, higher bandwidth, or both.

Acknowledgment

The authors would like to thank Konrad Iwanicki (University of Warsaw) and Frits van der Wateren (Chess) for their invaluable input and assistance in the creation of this paper.

REFERENCES

Abramson, N. 1977. The throughput of packet broadcasting channels. *IEEE Transactions on Communications 25,* 1, 117–128.

Anemaet, P. 2008. Distributed G-MAC: A Flexible MAC Protocol for Servicing Gossip Algorithms. M.S. thesis, TU Delft.

Arumugam, M. and Kulkarni, S. 2005. Self-stabilizing deterministic TDMA for sensor networks. In *Proceedings of the 2nd International Conference on Distributed Computing and Internet Technology (ICDCIT)*. Springer, 69–81.

Aschenbruck, N., Ernst, R., Gerhards-Padilla, E., and Schwamborn, M. 2010. Bonnmotion: a mobility scenario generation and analysis tool. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 51.

Cattani, M., Zuniga, M., Woehrle, M., and Langendoen, K. 2014. Sofa: Communication in extreme wireless sensor networks. In *Wireless Sensor Networks*, B. Krishnamachari, A. Murphy, and N. Trigoni, Eds. Lecture Notes in Computer Science Series, vol. 8354. Springer International Publishing, 100–115.

Cidon, I. and Sidi, M. 1988. Distributed assignment algorithms for multi-hop packet-radio networks. In *Proceedings of the 7th Annual Joint Conference of the IEEE Computer and Communcations Societies (INFOCOM) - Networks: Evolution or Revolution?* 1110–1118.

Degesys, J., Rose, I., Patel, A., and Nagpal, R. 2007. Desync: self-organizing desynchronization and tdma on wireless sensor networks. In *Proceedings of the 6th international conference on Information processing in sensor networks*. ACM, 11–20.

Dobson, M., Voulgaris, S., and van Steen, M. 2010. Network-level synchronization in decentralized social ad-hoc networks. In *5th International Conference on Pervasive Computing and Applications (ICPCA)*. IEEE, 206–212.

Dobson, M., Voulgaris, S., and van Steen, M. 2011. Merging ultra-low duty cycle networks. *41st International Conference on Dependable Systems and Networks (DSN)*, 538–549.

Dutta, P. and Culler, D. 2008. Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*. SenSys '08. ACM, New York, NY, USA, 71–84.

Elson, J. and Estrin, D. 2001. Time synchronization for wireless sensor networks. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium*. 1965–1970.

Ganeriwal, S., Kumar, R., and Srivastava, M. 2003. Timing-sync protocol for sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*. ACM New York, NY, USA, 138–149.

Kahn, J. M., Katz, R. H., and Pister, K. S. J. 1999. Next century challenges: Mobile networking for &ldquo;smart dust&rdquo;. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*. MobiCom '99. ACM, New York, NY, USA, 271–278.

Köpke, A., Swigulski, M., Wessel, K., Willkomm, D., Haneveld, P., Parker, T., Visser, O., Lichte, H., and Valentin, S. 2008. Simulating wireless and mobile networks in omnet++ the mixim vision. In *Proceedings of the 1st inter-*

national conference on Simulation tools and techniques for communications, networks and systems & workshops. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 71.

Kulkarni, S. and Arumugam, M. 2006. SS-TDMA: A self-stabilizing MAC for sensor networks. In *Sensor Network Operations*. IEEE Press, Chapter 4, 186–218.

Langendoen, K., Baggio, A., and Visser, O. 2006. Murphy Loves Potatoes: Experiences from a Pilot Sensor Network Deployment in Precision Agriculture. In *14th Int. Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*.

Li, Q. and Rus, D. 2006. Global clock synchronization in sensor networks. *Computers, IEEE Transactions on 55,* 2, 214–226.

Liu, M., Lai, T., and Liu, M. 2005. Is clock synchronization essential for power management in IEEE 802.11-based mobile ad hoc networks? In *Proceedings from the Second IEEE International Conference on Mobile Ad Hoc and Sensor Systems*.

Ma, J., Lou, W., Wu, Y., Li, X.-Y., and Chen, G. 2009. Energy efficient TDMA sleep scheduling in wireless sensor networks. In *IEEE INFOCOM 2009*. 630–638.

Mank, S., Karnapke, R., and Nolte, J. 2007. 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, 62–69.

Mank, S., Karnapke, R., and Nolte, J. 2008. 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*.

Maróti, M., Kusy, B., Simon, G., and Lédeczi, Á. 2004. The flooding time synchronization protocol. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*. ACM, 39–49.

Mirollo, R. and Strogatz, S. 1990. Synchronization of pulse-coupled biological oscillators. *SIAM Journal on Applied Mathematics 50,* 6, 1645–1662.

PixMob. `http://www.pixmob.com`.

Pussente, R. and Barbosa, V. 2009. An algorithm for clock synchronization with the gradient property in sensor networks. *Journal of Parallel and Distributed Computing 69,* 3, 261–265.

Roberts, L. G. 1975. Aloha packet system with and without slots and capture. *ACM SIGCOMM Computer Communication Review 5,* 2, 28–42.

Schmid, T., Dutta, P., and Srivastava, M. B. 2010. High-resolution, low-power time synchronization an oxymoron no more. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*. IPSN '10. ACM, New York, NY, USA, 151–161.

Song, J., Han, S., Mok, A., Chen, D., Lucas, M., and Nixon, M. 2008. WirelessHART: Applying wireless technology in real-time industrial process control. In *IEEE Real-Time and Embedded Technology and Applications Symposium, 2008. RTAS '08*. 377–386.

Tjoa, R., Chee, K., Sivaprasad, P., Rao, S., and Lim, J. 2004. Clock drift reduction for relative time slot TDMA-based sensor networks. In *15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*. Vol. 2.

Tseng, Y.-C., Hsu, C.-S., and Hsieh, T.-Y. 2002. Power-saving protocols for IEEE 802.11-based multi-hop ad hoc networks. In *IEEE INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings*. Vol. 1. 200–209 vol.1.

Van Dam, T. and Langendoen, K. 2003. An adaptive energy-efficient MAC protocol for wireless sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*. ACM, 171–180.

Varga, A. and Hornig, R. 2008. An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 60.

Weingartner, E., vom Lehn, H., and Wehrle, K. 2009. A performance comparison of recent network simulators. In *IEEE International Conference on Communications (ICC)*. 1–5.

Ye, W., Heidemann, J., and Estrin, D. 2002. An energy-efficient MAC protocol for wireless sensor networks. In *21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*. IEEE, 1567–1576.

Ye, W., Silva, F., and Heidemann, J. 2006. Ultra-low duty cycle MAC with scheduled channel polling. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*. ACM, 321–334.

APPENDIX

A. MYRIANED HARDWARE

The MyriaNed *V3* nodes we use were designed to be small, lightweight and reliable. They feature a Nordic nRF24L01+ radio chipset and an Atmel ATMega128 processor. The V3 nodes are also intended to be flexible, with optional sensors and even the possibility of augmenting their functionality using daughter boards via an edge-connector.

The nRF24L01+ radio communicates at either 1 or 2 megabit per second (Mbps) in the unregulated 2.4GHz spectrum, sharing the same frequencies used by WiFi and Bluetooth devices. The Nordic chipset provides for a fixed-size 32-byte MAC packet, with additional physical-layer headers (including a 16-bit CRC) being added automatically by the radio itself. At the 2 megabit setting, a physical packet broadcast takes approximately $300\mu s$. The nRF24L01+ does not have the ability to perform channel sensing or collision detection, nor does it provide any received signal-strength indicator (RSSI). This chip was chosen in spite of these limitations because it has extremely low power usage while powered off. Nodes running GMAC generally operate with the radio disabled for more than 95% of the time, so a high leakage current (the power used by the chip even when it is disabled) can significantly increase the overall power consumption of the device. The nRF24L01+ provides four operational modes: power-down, standby, receive and transmit. As mentioned, the power usage in the power-down state is extremely low at $900nA$, while in standby mode the radio consumes $26\mu A$. In receive mode at 2 Mbps it uses $13.5mA$, and it consumes $11.3mA$ in transmit mode.

For time keeping, the V3 nodes use a 32kHz real-time clock, or RTC. A single clock *tick* of this clock represents the smallest synchronization adjustment these nodes can make, and has a duration of $\frac{1s}{32768} \approx 30\mu s$. An external clock is required to wake the CPU up from deep sleep states, as its own internal clock is disabled while it is asleep. The ability to completely shut down the CPU during long idle periods is also essential to the low-power operation of the MyriaNed nodes. These clocks (actually, oscillators), like any clocks, have limited precision and will tend to *drift* in relation to another such clock over time. These devices carry a specification of $\pm 20ppm$ (parts per million) from the factory. This variability in individual clocks is exactly what GMAC tries to compensate for with its synchronization mechanisms, described in Sections 3 and 4.

In our experiments, these V3 nodes are almost always used in conjunction with *SED* (**S**torage of **E**nergy and **D**ata) modules. The SED module can operate only when attached to a host V3 node via its edge connector, since the module has no micro-controller of its own. A SED module is composed of a $\frac{1}{2}$ AA battery, two 2MB flash memory chips, and an on-off switch. The addition of the SED module makes the V3 node fully functional as an experimental node, giving it self-contained power and stable storage for logging.

B. BONNMOTION MOBILITY TRACE PARAMETERS

For the sake of experiment reproducibility, we provide the following two tables listing the BonnMotion [Aschenbruck et al. 2010] parameters we used. For the exact meaning and use of these parameters, please refer to the BonnMotion documentation[3].

Table V lists the complete suite of parameters used in generating the mobility traces for the simulations presented in this paper. All traces were generated using the BonnMotion suite, discussed in Section 5.2.

Table VI lists the transmission powers used in our simulations, along with their respective transmission range, transmission area, and transmission density (i.e., average number of neighbors per node).

[3]http://bonnmotion.net/

Table V: BonnMotion Parameters

model	GaussMarkov	RandomWalk	RPGM
randomSeed	1299022770517	1299023102669	1299023208481
x (m)	1000.0	1000.0	1000.0
y (m)	1000.0	1000.0	1000.0
duration (s)	10800.0	10800.0	10800.0
nn (nodes)	1000	1000	1000
circular	false	false	false
maxspeed (m/s)	5.0	5.0	5.0
minspeed (m/s)	N/A	0.1	0.1
maxpause (s)	N/A	60.0	60.0
updateFrequency (s)	2.5	N/A	N/A
angleStdDev (rad)	0.39269908169	N/A	N/A
speedStdDev (m/s)	0.5	N/A	N/A
bounce	true	N/A	N/A
initGauss	false	N/A	N/A
uniformSpeed	true	N/A	N/A
mode	N/A	t	N/A
modeDelta	N/A	60.0	N/A
groupsize_E	N/A	N/A	12.0
groupsize_S	N/A	N/A	2.0
pGroupChange	N/A	N/A	0.1
maxdist	N/A	N/A	25.0

Table VI: Transmission Power settings

TX Power (mW)	TX Range (m)	TX Area (m^2)	TX Density (nodes)
0.005764	8.92	250.0	0.25
0.016303	12.62	500.0	0.5
0.046111	17.84	1000.0	1
0.084712	21.85	1499.9	1.5
0.130423	25.23	1999.9	2
0.182271	28.21	2499.9	2.5
0.239602	30.90	2999.8	3
0.368891	35.68	3999.9	4
1.043381	50.46	7999.8	8
1.916814	61.80	11999.7	12
5.421568	87.40	23999.5	24
15.334513	123.61	47999.4	48