Network-level Synchronization in Decentralized Social Ad-Hoc Networks

Matthew Dobson, Spyros Voulgaris, Maarten van Steen
Dept. of Computer Science
Vrije Universiteit Amsterdam, The Netherlands
{mc.dobson, spyros, steen}@cs.vu.nl

Abstract

Social ad-hoc networks can be composed of a large number of tiny, wireless, battery-powered nodes, and exhibit arbitrary mobility. We aim to enable the execution of distributed applications that depend on capturing network dynamics across large-scale social ad-hoc networks. Conservation of the nodes' fixed energy budget is the chief concern in all design decisions, and necessitates that wireless communication is kept to a minimum. In this paper we present GMAC[1]*: a gossiping TDMA MAC layer created to address these problems. We analyze its efficiency in* achieving *and* maintaining *a tight network-layer synchronization with respect to the active periods of node radios.*

Keywords: ad-hoc wireless networks, TDMA MAC layer, duty cycle synchronization

I. Introduction

At the intersection of wearable computing, wireless ad-hoc networks, and social networks, lies an area we have dubbed social ad-hoc networks. Such networks are composed of nodes carried or worn by people, use wireless communication, do not rely on existing communication infrastructures, and are battery powered. Unlike typical wireless ad-hoc networks, that focus on propagating collected data to a sink, social ad-hoc networks aim at capturing network dynamics at regular intervals.

A number of applications can benefit from capturing network dynamics. Consider, for instance, a (large) group of people at a conference or similar social event, each wearing a small unobtrusive electronic badge with a limited radio range. By simply measuring how often and for how long two badges are within range of each other, we can register social interaction and study the structure of the social network. Furthermore, by aggregating and disseminating data we can even stimulate social interaction, for instance by a social game where groups of people (e.g., students of the same department) increase their score by talking to members of other groups, and lose points when sticking among themselves. Finally, a family or group of friends attending a large social event may be informed when they come in close proximity to each other, helping them to stay in contact. Other applications easily come to mind, including peer-to-peer messaging, finding people with specific profiles, and crowd management, to name a few.

There are three main challenges at the MAC layer that must be overcome in order to implement large-scale social ad-hoc networking. The first, and primary concern for almost any mobile device, is *energy consumption*. As these mobile nodes are battery powered, careful and judicious use of a node's fixed energy budget is essential. The second problem we face is *scalability*. We require that our platform operates efficiently on a wide variety of network sizes, from a few tens of nodes (e.g., a birthday party or small restaurant) to thousands of nodes (e.g., a large sporting event or stadium rock concert). Our third major challenge is node *mobility*. In the social settings where our applications will be executing people are free to join and leave the network, and to move anywhere they please (i.e., make arbitrary changes in the network topology). These dynamic changes to network topology can wreak havoc on many algorithms (e.g., routing and leader election) which assume stable and symmetric connections between nodes.

As part of an overall solution, we propose GMAC: a gossip-based MAC protocol, with energy-awareness being a primary goal. GMAC is designed to run on highly constrained sensor nodes and it therefore has very low processing requirements and a small memory footprint. This serves to reduce energy consumption, but GMAC's chief mechanism for energy conservation is *duty cycling*. Duty cycling is a common technique from the area of wireless

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

978-1-4244-9142-1/10/$26.00

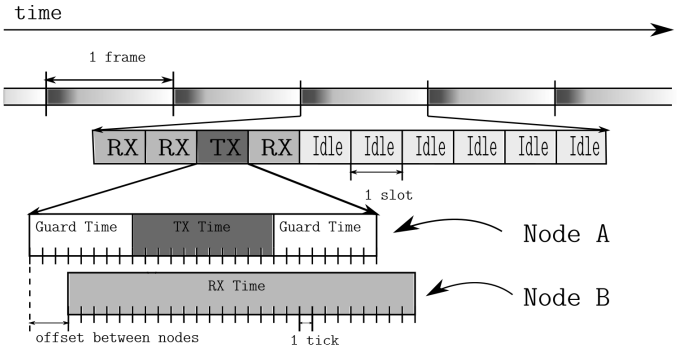


Fig. 1. GMAC's TDMA structure

sensor networks where individual nodes periodically turn on their wireless radio for only a short period of time to allow for communication. GMAC's communication is based on a periodic gossip paradigm, in which one period of gossip-based communication is known as a gossip *round*. During each gossip round, each node sends one *application* message and one *join* message. Application messages comprise a small GMAC header with synchronization and strategy information, and the application data. Join messages carry debug and/or statistical information from the GMAC core, and are also used to aid network synchronization.

Note that most existing MAC protocols, focusing on adapting radios' duty cycles to variable data propagation needs, are not appropriate for monitoring network dynamics at regular intervals. GMAC's periodic duty cycling provides an inherent solution to that, provided that nodes are properly synchronized with each other. The synchronization mechanism, which is what we concentrate on in this paper, is the glue that holds the entire network together. Properly synchronized, GMAC's gossip-based data dissemination will epidemically spread application messages throughout the network. Improperly synchronized, network-level packets will collide or go unheard entirely. By extensive simulation, we show that it is feasible for GMAC to dynamically establish and adaptively maintain a synchronized duty cycle for networks of very large scale, even in the presence of drifting clocks and diverse topologies.

II. System model / GMAC description

GMAC's core design considerations were predictable (and low) energy consumption and self-adaptability. In order to accomplish these goals, GMAC combines network-level gossiping with a low duty-cycle Time-Division Multiple Access protocol. TDMA protocols divide time into a repeated series of *frames* comprising a number of fixed-length *slots*, each of duration T_{slot}. They attempt to avoid message collisions by scheduling transmissions in unused slots. This scheduling mechanism is known as *slot allocation*, which GMAC provides in the form of scheduling **strategy** modules. For the slot allocation to be effective, nodes must be *synchronized*. That is, the nodes participating in this protocol must closely align their slots, minimizing the *offset* between all pairs of nodes. To this end, GMAC also supports the use of various **synchronization** modules. GMAC's TDMA structure is shown in Fig. 1.

GMAC maintains a fixed duty-cycle when in a synchronized state, and thus constant and predictable energy consumption. At compile-time GMAC computes the largest integer number of TDMA slots that fit within the desired frame time, $Slots_{frame} = \lfloor T_{frame}/T_{slot} \rfloor$. T_{slot} is based on the transmit time of a fixed-length GMAC packet and a *guard time*, which is padded to both ends of a single *TX time* to compensate for a possible synchronization offset between nodes, as shown in Fig. 1. The TX and guard times are measured at the granularity of the node's underlying hardware clock tick, T_{tick}[2], which determines the minimum possible synchronization adjustment. The transmit time is based on the radio hardware present on a node, specifically the total time for the radio to transition from standby mode to transmit mode ($T_{StandbyToTX}$) and the time to transmit a single packet ($\frac{PacketSize}{DataRate}$). Thus, $Ticks_{TX} = \lfloor (T_{StandbyToTX} + \frac{PacketSize}{DataRate})/T_{tick} \rfloor + 1$. The guard time is a compile-time constant, currently set to $300\mu s$, giving us $Ticks_{guard} = \lfloor 300\mu s/T_{tick} \rfloor$. This yields a final formula for the length of a TDMA slot: $T_{slot} = (Ticks_{TX} + 2 \times Ticks_{guard}) \times T_{tick}$. Finally, the GMAC's duty cycle is $Slots_{active}/Slots_{frame}$, where $Slots_{active}$ (the number of active slots in a frame) is a GMAC parameter.

A. Gossiping

GMAC addresses our issues of scalability and mobility by using a simple *gossip-based* communication paradigm [1]. Gossiping has had much success in peer-to-peer protocols for wired networks, partly due to its scalability and robustness in the face of network churn (changes in network membership or topology). For wireless networks, our basic model is that nodes periodically broadcast messages consisting of various *news items*. Other nodes within transmission range receive these messages and maintain a cache of recently heard news items. During each gossip period, or *round*[3], a node selects and sends a number of

[2]In the case of a 32,768Hz clock, $T_{tick} = 1s/32,768 \simeq 30.5\mu s$

[3]The term round and frame describe the same thing, a single period of GMAC communication. Round is used to refer to gossip, while frame is used at the MAC layer

news items from its cache, possibly including a freshly generated news item of its own. Similarly, a node that receives messages selects a number of news items that it heard during the round to store in its local cache. This simple protocol provides all nodes with an approximation of the true *global* view of all the news items in the network based purely on *local* data. The accuracy of this approximation depends on several factors, including the connectivity of the network, the size of a node's cache, and the algorithm a node uses to select which news items to send or store.

Network-level gossiping does not rely on routing, node IDs, or neighborhood maintenance. Still, it is highly fault tolerant, an ideal property in networks where both wireless links and individual nodes are unreliable. Such a simple gossiping protocol is sufficient for many ad-hoc networking operations, such as dissemination, aggregation, and even maintenance of routing tables, as in [2], in case they are needed.

B. Duty-Cycle Synchronization

GMAC's use of duty cycling makes proper node synchronization essential. If a node's frame is not synchronized with those of its neighbors, it is very likely that the node's broadcasts will go unheard. For example, with a typical duty-cycle of 1.5%, the odds of having the active periods of unsynchronized nodes overlap is quite low, meaning these nodes will not hear each other's messages. On the other hand, the better synchronized the nodes are, the more energy we can save. This is because GMAC is designed to be pessimistic, and currently uses a guard time that is quite high, approximately equal to the radio's TX time. A guard time that is too large will result in wasted energy as receivers wait with their radios on for transmitters to begin. On the other hand, a guard time that is too short will result in increased message collisions, as transmissions from adjacent slots begin to overlap. Shrinking this guard time as much as possible is one of the direct benefits of sharp synchronization.

Groups of nodes whose active periods overlap are said to form a *synchronized subnetwork*, or simply *subnetwork*. Synchronization of all nodes participating in the network then takes two forms: *maintaining synchronized subnetworks* and *joining separate subnetworks*. Because no two clocks are exactly identical, GMAC's synchronization module must maintain existing synchronized subnetworks by compensating for the inherent *clock drift* between any two nodes. GMAC operates in a completely decentralized manner, so there is a chance that nodes will form independently synchronized subnetworks or remain isolated. GMAC's join messages are responsible for joining these isolated nodes and subnetworks together.

The goal of the maintenance portion of GMAC's synchronization is to minimize the offset (see Fig. 1), $\Delta T_{i,j} = T_i - T_j$ between all pairs i, j of communicating nodes. GMAC divides the offset between nodes' notions of time into two parts: slot offset and phase offset. We define *slot* offset as the number of whole TDMA slots by which the two nodes differ, i.e. $\lfloor \Delta T_{i,j} / T_{slot} \rfloor$. We define *phase* offset as the number of clock ticks (intervals of T_{tick}) the two nodes are offset within one TDMA slot. So, the total offset between the nodes is the sum of the slot offset and phase offset between them.

Currently GMAC provides one synchronization module implementing a method known as the **median** algorithm, described in Alg. 1. The median algorithm uses the number of packets received during the current frame and the array of packets as input. The algorithm sorts the received packet entries by their offset from the local time, selects the median entry, and increases (or decreases) the number of slots for the next frame based on the timing data pertaining to that entry.

Algorithm 1 Median Synchronization

Require: $NumEntries > 0$, array $RxEntries$
$SortByOffset(RxEntries)$
$MedianEntry \leftarrow RxEntries[\frac{NumEntries}{2}]$
$TickCorrection \leftarrow MedianEntry.Offset/2$
$SlotCorrection \leftarrow TickCorrection \text{ div } Ticks_{slot}$
$PhaseCorrection \leftarrow TickCorrection \bmod Ticks_{slot}$
$NextFrameSlots \leftarrow Slots_{frame} - SlotCorrection$
$NextSlotTicks \leftarrow Ticks_{slot} - PhaseCorrection$

As GMAC nodes execute their synchronization algorithm in a completely decentralized manner, it is possible that separately synchronized subnetworks form (i.e., subnetworks partitioned in time, rather than space). In an attempt to join these separate subnetworks together to form a single network, GMAC nodes send out *join* messages in every TDMA frame. Each node randomly selects a slot during the inactive portion of the frame and transmits a message filled with synchronization and debugging information. Occasionally, a join message from $Subnetwork_A$ will be received during the active period of $Subnetwork_B$. Nodes receiving a join message will modify their local clocks in an attempt to synchronize to the sending node. Note that this join behavior is purely probabilistic, and it is not uncommon for disjoint subnetworks to exist, despite the nodes being in close physical proximity.

C. Slot Scheduling Strategy

GMAC's slot allocation algorithms are based on Slotted Aloha [3], and are used to share access to the wireless medium. At a high level, the main difference between

GMAC and Slotted Aloha is GMAC's use of duty cycling. Slotted Aloha is an *always-on* protocol, whereas GMAC turns the radio on only for a small percentage of the total slots in order to save energy. Slots in which the radio is powered up are known as *active* slots, in contrast to inactive, or *idle* slots, where the radio is powered down. At the moment, GMAC provides three different TDMA strategy algorithms. For the purposes of this paper, we restrict our investigation to the most basic strategy, known as **Simple TDMA**.

The Simple TDMA strategy takes as a parameter the number of active slots in a frame, denoted $Slots_{active}$. In each frame, the algorithm selects a random TX slot in the range $[1..Slots_{active}]$. This algorithm works well in low- to moderate-traffic networks. However, highly congested neighborhoods (i.e., where $\#Neighbors >> \#ActiveSlots$) will result in a large number of collisions, and thus significantly reduced throughput. The **Distributed TDMA** strategy attempts to remedy this problem by adapting the number of active slots based on the perceived number of neighbors. We leave the investigation of this and other strategies to our future work.

III. Experimental setup

We implemented our GMAC simulator using the MiXiM (http://mixim.sourceforge.net) extensions over OMNET++ (http://www.omnetpp.org). The OMNET++ platform is expressive, efficient, modular, and increasingly the de-facto simulation environment for mobile ad-hoc and sensor networks [4].

A. Simulated Nodes

As we are interested only in synchronization, we can ignore the difficulties of modeling sensors and actuators. The behavior of the GMAC layer is then driven solely by the interrupts from the internal clock and the radio. We chose to model our simulated clocks and radios in accordance with the hardware GMAC was designed to run on in the real world, MyriaNed nodes. These nodes have an Atmel ATXMega128 CPU with 8k of RAM, 128k of Flash memory, a Nordic nRF24L01+ wireless radio, and four colored LEDs, and several sensors. The internal timer is an oscillator designed to operate at 32,768Hz, and are specified to have a frequency offset $\leq \pm 20ppm$ (parts per million). This may seem small, but it means that after thirty seconds[4], two initially synchronized clocks may have drifted as much as $1.2ms$ (40 ticks) apart. That is, one of the nodes may have counted as many as one million and twenty ticks in this time interval, while another may have counted only nine-hundred ninety-nine thousand, nine-hundred and eighty ticks in that same interval. With $Slots_{active} = 8$ active slots of $Ticks_{slot} = 28$ ticks each, two nodes can become completely isolated from each other (i.e. their active periods do not overlap) in as little as 3 minutes. Compensating for this inherent divergence of separate clocks is the goal of any clock synchronization algorithm.

[4]one million time intervals of $T_{tick} = \frac{1}{Freq} \simeq 30\mu s$

TABLE I. Networks Investigated

Nodes	Dimensions	Spacing
16	$320m \times 320m$	80m Matrix, Random
64	$640m \times 640m$	
256	$1280m \times 1280m$	
1024	$2560m \times 2560m$	

We designed our own OMNET++ modules to represent the type of internal clocks described above. OMNET keeps track of the global *simulation time*, T_{sim}, while an individual node computes its own *local time*. A node bases this on its own clock's frequency offset (F_{offset}) and phase offset (P_{offset}) from the global simulation clock (provided as OMNET parameters): $T_i = (T_{sim} \times F_{offset_i}) + P_{offset_i}$. 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 the node's clock runs.

For the purposes of this paper, we have restricted our investigation to GMAC's most basic synchronization primitives: *median* synchronization maintenance and the *Simple TDMA* strategy, which were discussed in Section II.

B. Input parameters to investigate

As our primary interests lie in GMAC's synchronization and stability in large mobile networks, our investigations focus on the following parameters: clock drift, transmit power and starting topology. Due to space constraints, our investigation into GMAC's handling of mobility will be deferred to a future publication.

1) Clock Drift: Even clocks of the same specifications exhibit slight differences, as previously discussed. We investigate a range of *maximum* clock drift settings in our experiments. By selecting a maximum offset of O_{max}, the simulator will assign each simulated node's clock a random clock frequency multiplier in the range $[1 - O_{max}, 1 + O_{max}]$. This results in each node having a similar but slightly varied value for T_{tick}, causing our simulated clocks to slowly drift apart just as real clocks do.

2) Starting Topology: In this investigation, we examine two different network topologies: matrix topologies and

random topologies. In matrix topologies, N nodes are deployed in a $\sqrt{N} \times \sqrt{N}$ grid topology, where rows (and columns) and placed $80m$ apart. This will show us the behavior of the GMAC in a more 'friendly' setting, where the network is fully connected and there exist many paths between all nodes. In the random topologies, N nodes are deployed at random locations in the same area as the equivalent matrix topology (i.e., with the same number of nodes). Random topologies present more difficulty in synchronization as there will generally be some very highly connected areas and some less connected areas, and potentially even physically isolated nodes. These random topologies, however, are more representative of the type of topologies that will be encountered in the real world, and hence of more interest to our investigation. Since scalability is one of our primary interests, we investigate networks of various combinations of size and deployment pattern. A full list is given in Table I.

C. Observable metrics

When a simulated node begins a new round, it records the round number and global simulation time (T_{sim}). By later comparing the times that the nodes began each round, we can analyze the synchronization behavior of a simulated GMAC network.

IV. Simulation results

In this section, we present the results of our experiments simulating the GMAC layer in the OMNET++ environment.

We elected to analyze the GMAC's synchronization performance in two ways. First, we ran a series of experiments where all nodes were started in a *synchronized* state at the very beginning of the run ($P_{offset} = 0$). Second, we ran experiments where each node i was started at a randomly chosen time in the first 15 seconds ($0.0 \leq P_{offset} < 15.0$). The first set of experiments is designed to show how GMAC's synchronization maintenance, i.e., the median algorithm, performs in an already synchronized network. The second set will give us insight into the performance of GMAC's subnetwork joining mechanism, i.e., the join messages.

A. Synchronous Start

For this series of experiments, our primary metric for measuring the level of synchronization in the network is the standard deviation of the node start times. As mentioned earlier, each node reports the time it begins each round. For each round, we compute the standard deviation of all node's reported time for the start of that round. This tells us how tightly synchronized the network is at every round. This is a somewhat broad measurement, as it pertains to the whole network. That is, it cannot give us insight into which sections of the network are suffering from poor synchronization. Note that the best synchronization level we can realistically hope for is $\frac{T_{tick}}{2} \simeq 15\mu s$, or half the smallest adjustment that the GMAC can make in its effort to compensate for the drifting clocks. Consistent and stable synchronization to the level of a single clock-tick is quite good, though, and indicates that we could reduce T_{guard} significantly below its current value of $300\mu s$ in order to save energy.

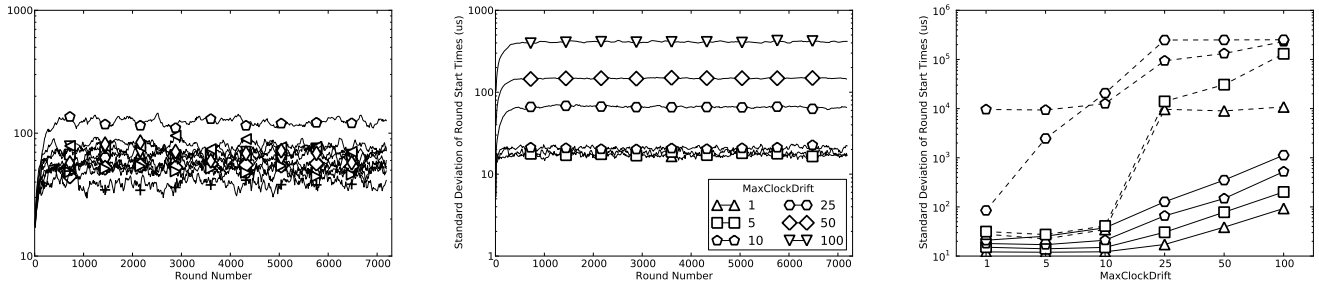
We begin by looking at Fig. 2a, depicting a set of individual simulated runs. Each line is the result of a single run on a 256-node matrix topology, with the standard deviation plotted per round. These simulations used the default transmission power of $20mW$ and *MaxClockDrift*=$25ppm$. Here we can see much similarity between individual runs. The median algorithm maintains a steady-state where it compensates for the clock drift between the nodes.

In Fig. 2b, each line shows the average standard deviation for each simulated round across the 10 experimental runs for each *MaxClockDrift* setting. We explore the effects of different clock drifts on the median algorithm, using the same topology and transmit power settings as above. Thus, the 10 lines from Fig. 2a comprise the single line in Fig. 2b at $25ppm$. Here we can see how the median algorithm copes with various clock settings. More clock drift between nodes leads to progressively looser synchronization. That is, as clocks drift apart faster and faster, the median cannot maintain the same level of synchronization between nodes. This can be seen by the increased standard deviation at higher drift settings.

Finally, in Fig. 2c we see the aggregation of a large amount of experimental data. Each data point represents the average standard deviation for the last half of the 10 runs of a particular parameter setting. We plot the clock drift settings along the x-axis and average converged standard deviation for the 10 runs of each *MaxClockDrift* value. Thus, the single points along the line {*Matrix, 256*} show the average value over the last 30 minutes of the 10 runs represented by the individual lines from Fig. 2b. The solid lines show *matrix* topologies, while the dashed lines show *random* topologies and, as expected, the random topologies show much more variability.

B. Asynchronous Start

For our second series of experiments, our metric for analyzing the joining of the nodes into a single synchronized network is the fraction of nodes *not* belonging to the largest synchronized subnetwork. When analyzing our logs, we group nodes into subnetworks as follows: we

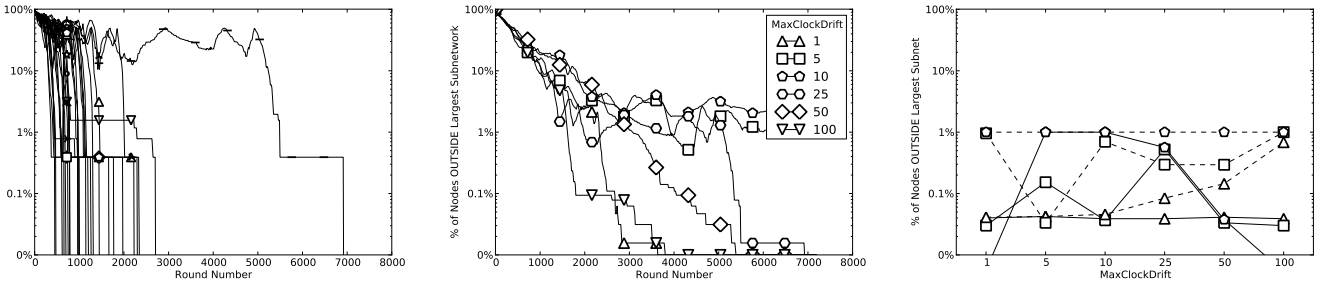


(a) 10 individual runs of same settings. *Topology*: 256-node Matrix, *MaxClockDrift*: 25ppm.

(b) Average over 10 runs for each *MaxClockDrift*. *Topology*: 256-node Matrix.

(c) Converged offset standard deviations for combinations of *MaxClockDrift* and *Topology*. From top to bottom: {1024, 256, 64, 16} nodes, {Random, Matrix} topologies.

Fig. 2. Variation in round start times, synchronous start



(a) 10 individual runs of same settings. *Topology*: 256-node Matrix, *MaxClockDrift*: 25ppm.

(b) Average over 10 runs for each *MaxClockDrift*. *Topology*: 256-node Matrix.

(c) Per-Parameter Per-Topology Average

Fig. 3. Percentage of unsynchronized nodes, asynchronous start

sort all the reported times for the start of a particular round. We iterate through the times, computing the offset between the previous and current time. When there is an offset between consecutive reported times greater than the subnetwork *threshold*, we decide that the current reported time marks the beginning of a new subnetwork. By separating the nodes into disjoint sets representing synchronized subnetworks at various thresholds, we can analyze how well GMAC's join mechanism is working. We use thresholds of $7000\mu s$, $850\mu s$, and $30\mu s$ which correspond to approximately the length of an active period, a TDMA slot, and a simulated clock tick, respectively. At each round, for each threshold, we calculate the percentage of all simulated nodes that are *outside* (i.e., not synchronized with) the largest subnetwork. As such, we should expect this metric to begin at or near 100% and fall towards 0%. In Fig. 3, we show the results of our asynchronous start experiments. In general, most runs behave as expected and rapidly converge to a single network. However, in some runs, we find that separate subnetworks can exist for a long time, to the point where they never form a single network during the entire simulated hour. We can see just such behavior in Fig. 3a, which depicts the results of a set of twenty-five 256-node matrix runs with a maximum clock drift of $\pm 25ppm$. In Fig. 3b, we show the average of 25 such runs for each *MaxClockDrift* setting. As previously explained, the behavior of the join mechanism is probablistic, and this can be clearly seen in the variability of the simulated results.

Finally, in Fig. 3c we plot (analogously to Fig. 2c) the average fraction of nodes outside the largest subnetwork over 25 runs for each setting, considering only the last half of the rounds of each run. The settings correspond to all combinations of clock drifts and topologies we experimented with. Clearly, in all of our simulations GMAC eventually achieved synchronization of at least 99% of the participating nodes, irrespectively of the topology and maximum clock drift.

V. Related work

In [5], the authors discuss why traditional synchronization mechanisms, like NTP, are unsuitable for ad-hoc sensor networks. In [6] the authors give an overview of the

problem of time synchronization in sensor networks and evaluate several important protocols. [7] provides a more recent survey of time synchronization protocols designed for sensor networks, however the most scalable protocol analyzed was run on networks up to 300 nodes, far below what is required here.

Many popular energy-aware MAC protocols, e.g. S-MAC [8], use carrier-sensing or ready-to-send/clear-to-send handshakes, which necessarily use more energy than a properly synchronized TDMA protocol that avoids collisions without these mechanisms. Both Timing-sync Protocol for Sensor Networks (TPSN [9]) and Flooding Time Synchronization Protocol (FTSP [10]) 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. Reference Broadcast Synchronization (RBS [11]) provides only *post-facto* synchronization and is thus infeasible in our setting. Post-facto synchronization is used to retroactively determine the time a past event occurred at a neighbor node (i.e., to come to consensus on what time an event sensed by several node took place), but is not designed to synchronize the current clock state of the nodes.

VI. Conclusions and Future Work

Our results indicate that the simple median synchronization mechanism provided by GMAC is capable of adaptively maintaining good clock synchronization, even at very large network sizes. Simulations show that the median algorithm is capable of compensating for clock frequency offsets far greater than can be expected from real clock components. Furthermore, simulations show the median algorithm can maintain synchronization in networks as large as 4096 nodes, and possibly beyond. This level of scalability is an absolute necessity in the setting of large social ad-hoc networks.

Though this synchronization maintenance algorithm is quite simple, in practice it turns out to be quite effective. One problem with the median algorithm is that it has no 'memory', or *state*. This means that the median algorithm can maintain synchronization only while it is in regular contact with other nodes. If a node becomes isolated (without neighbors) for an extended period of time, the node's clock will tend to drift away from the other nodes it was previously communicating with. It also seems that for smaller networks, GMAC keeps nodes tightly coupled. This indicates that there may be much energy to be saved by reducing the TDMA guard time.

Though GMAC's synchronization maintenance mechanism was shown to be scalable, its join mechanism did not prove as effective. As a node can only receive join messages during its (short) active period, separately synchronized subnetworks can persist for a long time. This is undesirable as subnetworks will be unable to communicate with each other, leading to significantly reduced utility of the network as a whole. We plan to investigate using a more *active* join mechanism, which periodically listens for other subnetworks and actively tries to deterministically join them. The frequency of this extended listening period could be adaptively based on the size of the network, neighborhood density, or other metrics.

If we can modify GMAC to efficiently join all reachable nodes into a single synchronized network, we will have a firm foundation on which to build scalable distributed applications running on massive, dynamic social ad-hoc networks.

References

[1] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic algorithms for replicated database maintenance," in *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*. ACM Press New York, NY, USA, 1987, pp. 1–12.

[2] K. Iwanicki and M. van Steen, "Multi-hop cluster hierarchy maintenance in wireless sensor networks: A case for gossip-based protocols," in *Proceedings of the Sixth European Conference on Wireless Sensor Networks (EWSN 2009)*. Cork, Ireland: Springer-Verlag LNCS 5432, February 2009, pp. 102–117. [Online]. Available: http://www.few.vu.nl/ iwanicki/publications/2009-02-EWSN/

[3] N. Abramson, "The throughput of packet broadcasting channels," *IEEE Transactions on Communications*, vol. 25, no. 1, pp. 117–128, 1977.

[4] E. Weingartner, H. vom Lehn, and K. Wehrle, "A performance comparison of recent network simulators," in *IEEE International Conference on Communications, 2009. ICC'09*, 2009, pp. 1–5.

[5] J. Elson and K. R
"omer, "Wireless sensor networks: A new regime for time synchronization," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 1, p. 154, 2003.

[6] F. Sivrikaya and B. Yener, "Time synchronization in sensor networks: A survey," *IEEE network*, vol. 18, no. 4, pp. 45–50, 2004.

[7] S. Rahamatkar, A. Agarwal, and N. Kumar, "Analysis and Comparative Study of Clock Synchronization Schemes in Wireless Sensor Networks," *Analysis*, vol. 2, no. 03, pp. 536–541, 2010.

[8] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient MAC protocol for wireless sensor networks," in *IEEE INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings*, 2002, pp. 1567–1576.

[9] S. Ganeriwal, R. Kumar, and M. Srivastava, "Timing-sync protocol for sensor networks," in *Proceedings of the 1st international conference on Embedded networked sensor systems*. ACM New York, NY, USA, 2003, pp. 138–149.

[10] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, "The flooding time synchronization protocol," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*. ACM, 2004, pp. 39–49.

[11] J. Elson and D. Estrin, "Time synchronization for wireless sensor networks," *Parallel and Distributed Processing Symposium, International*, vol. 3, p. 30186b, 2001.