Binary Consensus in Sensor Motes

Noor Al-Nakhala
Qatar University
Doha, Qatar
nalnakhala@qu.edu.qa

Ryan Riley
Qatar University
Doha, Qatar
ryan.riley@qu.edu.qa

Tarek M. Elfouly
Qatar University
Doha, Qatar
tarekfouly@qu.edu.qa

***Abstract*—In this work, we adapt the binary consensus algorithm for use in wireless sensor networks. Binary consensus is used to allow a collection of distributed entities to reach consensus regarding the answer to a binary question and the final decision is based on the majority opinion. Binary consensus can play a basic role in increasing the accuracy of detecting event occurrence. Existing work on the algorithm focuses on simulation of the algorithm in a purely theoretic sense. In this work, we modify the algorithm to function in wireless sensor networks by adding a method for nodes to determine who to communicate with as well as adding a heuristic for nodes to know when the algorithm has completed. We implement and test our algorithm in real wireless sensor motes and further support our results with a wireless mote simulator.**

***Index Terms*—Binary Consensus, TinyOS, Wireless Sensor Networks.**

I. Introduction

Algorithms for cooperative decision making have received significant attention in recent years from the theoretical computer science community. In these algorithms, a network of agents seeks to reach a decision and ensure that all nodes in the network know the final decision. One such algorithm in this area is binary consensus [1], [2]. Under binary consensus, the nodes in the network must simply agree on whether a statement is TRUE or FALSE. For example, a network of nodes capable of detecting natural gas could use binary consensus to answer the question "Is the amount of gas in the air greater than 10,000 ppm?" in order to help detect a gas leak in a gas processing center.

In the binary consensus problem, each node has an initial state of either 0 (false) or 1 (true), and the nodes should, in a distributed fashion, decide which one of these values is currently held by the majority of the nodes in the network.

The existing algorithm for binary consensus has two limitations. First, it doesn't specify how nodes find partners to run the algorithm with. Second, it doesn't provide a way for an individual node to determine when consensus has been reached. In order to implement the algorithm in a real distributed network, these limitations must be overcome.

Wireless sensor networks consisting of small, embedded devices (called motes) provide an excellent platform for binary consensus. Motes contain sensors that can be used collect data about their environments, and can communicate with each other wirelessly. Due to limitations regarding their size and power, sensor motes are computationally weak and should limit the number of packets they send.

In this paper, we propose a set of modifications to binary consensus that will allow it to operate in the context of wireless sensor motes having the limitations described above. Our modifications consist of changing how motes decide who to communicate with and also adding a heuristic to help motes estimate when consensus has been achieved. We have implemented our algorithm in a set of TinyOS based sensor motes and verified our algorithm functions both in hardware and in simulation.

II. Background

In this section we will give a brief overview of binary consensus and potential applications of it to wireless sensor networks (WSNs). Due to space constraints we do not provide an intensive background on WSNs, but instead assume the reader is familiar with the topic.

A. Binary Consensus

There are a variety of algorithms that are meant to allow a network of distributed nodes to reach consensus in a computation. In this work, we are specifically concerned with the problem of *binary consensus* [3], [4], [5], [6], where each node in the network holds one of two states and the algorithm allows all nodes to learn which state is held by the majority of nodes. There are many applications of such an algorithm, such as determining if the majority of sensors in a network have observed a certain event. Two strengths of binary consensus are that it is guaranteed to the correct conclusion [5], and that there is an upper-bound on the time to convergence [2].

Under binary consensus, nodes in the network start with their initial state and then update their state with each other based on an updating protocol. Convergence occurs when all nodes agree on the majority opinion. When two nodes communicate and run the updating protocol, they compare current states and then each assume a new state based on what they have seen. While the algorithm is running a node may be in one of four states, which can be described informally as:

1) 0 – The node believes the majority opinion is most likely false.
2) e_0 – The node believes the majority opinion might be false.
3) e_1 – The node believes the majority opinion might be true.

978-1-4673-2480-9/13/$31.00 ©2013 IEEE

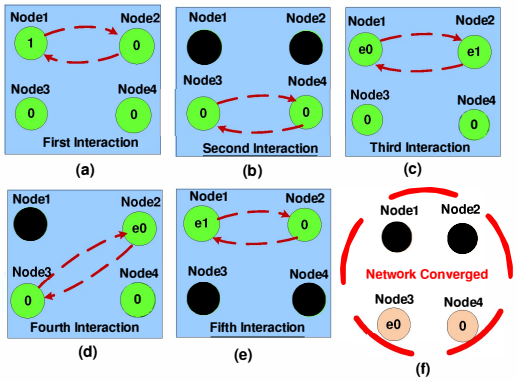


Fig. 1: Example of binary consensus algorithm functionality

4) 1 – The node believes the majority opinion is most likely true.

The updating protocol, as quoted from [2], is as follows:

> Each node is in one of four states: 0, e_0, e_1, and 1. The states satisfy the following order $0 < e_0 < e_1 < 1$. At each contact of a pair of nodes, their respective states x and y (without loss of generality) ordered such that $x \leq y$, are updated according to the following mapping $(x, y) \mapsto (x', y')$ defined by

$$\begin{aligned}
(0, e_0) &\rightarrow (e_0, 0) \\
(0, e_1) &\rightarrow (e_0, 0) \\
(0, 1) &\rightarrow (e_1, e_0) \\
(e_0, e_1) &\rightarrow (e_1, e_0) \\
(e_0, 1) &\rightarrow (1, e_1) \\
(e_1, 1) &\rightarrow (1, e_1) \\
(s, s) &\rightarrow (s, s), \text{for } s = 0, e_0, e_1, 1.
\end{aligned}$$

Convergence occurs when all nodes have states $\in \{0, e_0\}$ or $\in \{e_1, 1\}$. This means that if all nodes in the network have state 0 or e_0, then the network has converged and the majority of nodes initially held the value 0. Likewise, if all nodes in the network have state e_1 or 1, then the network has converged and the majority of nodes initially held the value 1.

The following example illustrates the functionality of the algorithm. Consider that there is a network with 4 nodes, 1, 2, 3 and 4 having initial states of (1; 0; 0; 0) respectively as shown in Fig. 1(a). The first interaction happens between nodes 1 and 2 and the state of node 1 becomes e_0 while the state of node 2 will be e_1. (This is according to the rules given above.) So the new sequence of states will be (e_0; e_1; 0; 0). Next, the second interaction is between nodes 3 and 4 as shown in Fig. 1(b); they communicate and nothing happens since they both hold the same state. Now, nodes 1 and 2 communicate again as depicted in Fig. 1(c), and their states are swapped leading to (e_1; e_0; 0; 0). Nodes 2 and 3 communicate as illustrated in Fig. 1(d) and also swap their states: (e_1; 0; e_0; 0). Finally, node 1 communicates with node 2 as shown in Fig. 1(e) leading to the converged states (0; e0; e0; 0) illustrated in Fig. 1(f). We consider this set of states converged because all nodes have value 0 or e_0. This means that the majority of nodes initially held state 0.

It is important to note that even though convergence has occurred, the nodes continue to communicate and exchange states. This is because individual nodes do not have global knowledge of the states of all others, and therefore cannot be certain whether convergence has occurred. Absolute certainty regarding convergence would require global knowledge.

TABLE I: Packets used during mote-to-mote communication. M1 and M2 are motes in the communication.

Packet	Payload	Description
P1	M1 State	M1 sends this packet to all motes in range.
P2	M2 State	M2 replies to M1 by sending this packet.
P3	-	M1 sends this packet to M2 in order to confirm that its state update was successful.

B. The Usage of Binary Consensus in Real World Applications

There are several applications in which the binary consensus algorithm may be used to accomplish a certain decision.

Consider a scenario when having a network of sensors capable of detecting the presence of an object using a camera. Then binary consensus algorithm can be applied to such a network in order to increase the accuracy of the final decision by taking the opinion of the majority motes. Security, military and hazardous locations are some examples of such scenario.

Another scenario where binary consensus can be used is in detecting gas leaks. For example, if there is a network consisting of gas sensors, and some sensors detect that there is gas leaking in one of the gas tanks, then binary consensus can be applied to such network to increase the accuracy of the final decision which based on the sensors' majority opinion.

III. Design and Implementation

While the binary consensus algorithm described in [2] and Section II-A provides a complete specification of how nodes should update their states, it leaves two important things unstated which are vital for implementing the algorithm in WSN. First, the algorithm does not discuss how individual nodes find a partner to update states with. Second, the algorithm does not provide a method for individual nodes to determine when convergence has occurred. In this section we will discuss modifications to the binary consensus algorithm that will allow us to provide both of these pieces of missing functionality.

A. Mote-to-Mote Communication

The motes that are part of a WSN do not, by default, have any awareness of the identities of any other motes in the network. Motes learn the identities of those around them by simply broadcasting and listening to messages. In this case, how does a mote determine who to communicate with and update its state? In our solution motes will randomly broadcast to their neighbors (other motes within range of receiving their wireless packets) in order to find partners.

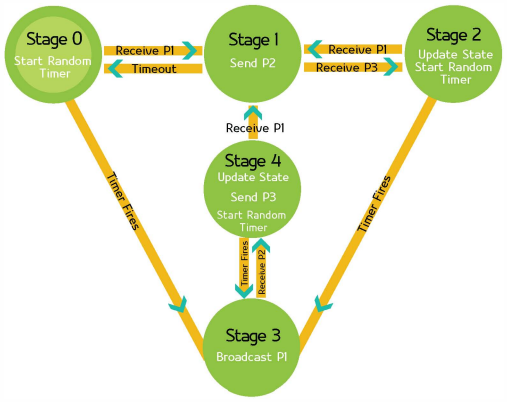


Fig. 2: Stage transition diagram of the communication algorithm

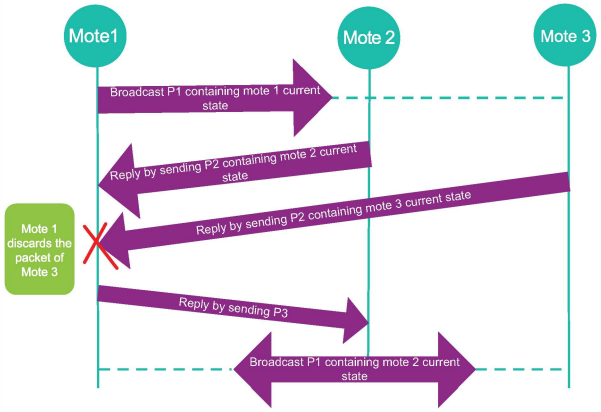


Fig. 3: Simple example of mote communication state updates

Fig. 2 illustrates a stage transition diagram of our communication algorithm[1]. Table I describes the types of packets sent and received during the algorithm. Our stages can be described as follows:

- *Stage 0*: After initialization, all motes start at Stage 0. During this stage, a mote will determine its initial state (0 or 1) and set a random timer that will decide when the mote will wake-up and broadcast information to its neighbors. If a mote is still in this stage when that timer fires, then it will transition to stage 3. If, instead, it receives a $P1$ packet from another mote, then it will transition to stage 1.
- *Stage 1*: After receiving $P1$, the mote will reply with a $P2$ packet containing its current state. This signifies to the sender that this mote is available to exchange state information. After sending $P2$ the mote will wait for a reply. During this time the mote will ignore any packets from other motes. After receiving a reply, the mote transitions to stage 2. If no reply is received after a suitable timeout, the mote returns to stage 0.
- *Stage 2*: When the mote receives $P3$, it will update its state using the rules previously described. At this stage both motes in the communication have updated their states. After this, the mote is free to communicate with another mote, and as such starts a timer and also waits for a potential $P1$ packet, just as in stage 0.
- *Stage 3*: In the event a mote has not been contacted by others, then eventually its own random timer will fire. In this case, the mote transitions to stage 3. After the timer fires, the mote will broadcast $P1$ and will wait to receive a packet of type $P2$. Once it receives it, it moves on to stage 4.
- *Stage 4*: After receiving $P2$, which contains the other mote's current state, the mote will update its state using the rules previously described. Next, it will send $P3$. After this, the mote is free to communicate with another mote, and as such starts a timer and also waits for a potential $P1$ packet, just as in stage 0.

[1]The astute reader will note that this is a state transition diagram with the word "stage" substituted for "state". This is to prevent confusion between the *state* of the mote (meaning $0, e_0, e_1,$ or 1) and the *stage* of the algorithm the mote is currently running.

Fig. 3 illustrates a simple example of how the motes communicate to update their states. Assume that we have 3 motes: 1, 2, and 3. All motes are initially in stage 0, waiting to either receive a packet or for their individual timers to fire. After a time, the timer on mote 1 fires and mote 1 broadcasts $P1$ to all its neighbors. Both mote 2 and mote 3 receive the broadcast. Mote 2 receives $P1$ first, and sends $P2$ in reply. Mote 1 receives the reply and updates its state accordingly. Shortly after that, mote 3 also sends $P2$, however since mote 1 received mote 2's reply first, it drops the reply of mote 3. Next, Mote 1 sends $P3$ to mote 2, who receives it and updates its state as well.

Note that we have not discussed packet loss in our example. Packets P2 and P3 are automatically acknowledged and resent if lost. We make use of the acknowledgement features built into the radio unit of our IRIS motes in order to accomplish this, and we leave the details out of our description of the protocol for the sake of clarity. $P1$ is not acknowledged because it is a broadcast packet.

B. Estimating Convergence

In standard binary consensus, nodes continue to run the algorithm even after convergence has occurred. This is because individual nodes have no way of knowing that the algorithm has converged. From an individual node's perspective, the algorithm does not have a stop condition.

In a wireless sensor network, this is unacceptable. In order to save power, it is vital that sensor motes know when to stop communicating. As such, we have designed a tunable heuristic that will allow motes to estimate when convergence occurs.

Whenever a mote updates its state, it also keeps track of the last N states that it has held. In the event that the mote has not significantly changed its guess for the last N state changes (meaning that all of the old states $\in \{0, e_0\}$ or $\in \{e_1, 1\}$) then the mote estimates that convergence may have occurred. In this situation the mote will disable the timer it uses to randomly wake-up and broadcast $P1$. In the event

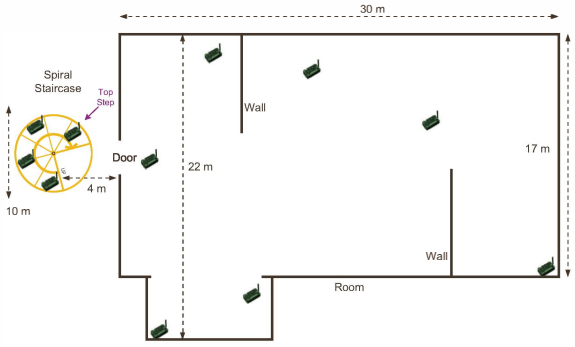


Fig. 4: Deployment configuration of IRIS motes

the network has actually converged, very quickly all motes will disable their timers and communication will cease. In the event the mote was incorrect; however, and the network has not converged then the mote is still able to respond to $P1$ packets it receives and participate. If, during one of these responses, it goes through a significant state change, it will reactivate its timer.

It is important to note that while this heuristic allows the algorithm to stop sending packets, it also removes the guarantee that convergence will occur correctly because there is a possibility that all motes could incorrectly assume convergence. This means that the value chosen for N is very important. If it is too low, then "false convergence" could occur. If it is too high, then needless packets are sent. The issue of choosing N is discussed further in Section IV.

IV. Experiments

We have implemented our algorithm in the IRIS family of sensor motes from the MEMSIC corporation [7], [8]. We used a development version of TinyOS (between versions 2.1.1 and 2.1.2). Our implementation required about 400 lines of nesC code, including appropriate comments.

We then tested our implementation both in hardware as well as in the TinyOS simulator TOSSIM. In this section we will discuss our testing methodologies and results.

A. Hardware

The binary consensus algorithm implementation was tested in 11 IRIS motes. The motes were placed indoors in a wide room as well as on an accompanying spiral staircase. Fig. 4 shows the layout of the motes deployment as well as the dimensions of the room where they were placed. The goal of the placement was to ensure that the network, while connected, was not fully-connected. Similar configurations containing fewer motes were also tested.

Five trials (signified T1 - T5 in the table) were performed for each configuration of motes. Each trial contained a different distribution of initial states (either 1 or 0) for the motes. In T1, initial states were distributed in such a way that if a mote had a value of 0 then its neighbor would have a value of 1. This configuration is close to optimal for the algorithm, as the mote will directly communicate with the neighbor motes that hold the opposite state and the majority value will be spread through the network faster. In T5, all 0s were concentrated to one side of the network while the 1s were concentrated to the other. This mimics a worst case scenario. In both cases, the number of 1s in the network is very close (within 1 or 2) to the number of 0s in the network. T2 - T4 made use of distributions that slowly shift from T1 to T5. (T1 is the "easiest" distribution, T2 is slightly more difficult, etc.)

TABLE II: Convergence time and N values for hardware motes

Motes	T1	T2	T3	T4	T5	AVG	N
5	33.5 s	36 s	39.5 s	40.5 s	41.8 s	38.3 s	5
7	45 s	57 s	62 s	63 s	81 s	63 s	7
11	92 s	98 s	100 s	116 s	146 s	110 s	10

In order to verify the correctness of the algorithm as well as to measure the time to convergence, one additional mote was programmed as a base station using the BaseStation15.4 application provided as part of TinyOS [9]. The mote programmed with this application was connected to a laptop via a serial link and was able to "sniff" all packets sent and received in the network.

The time to convergence was calculated manually by monitoring the packets sent and received from the serial readings received at the base station. The motes were turned on and a manual timer was started once the packets started to appear in the monitor. After reaching our heuristic for consensus, motes will stop initiating communication and eventually no packets will be sent over the network. We considered the network converged at the time the last packet was sent by any mote in the network. In addition, the correctness of convergence was verified.

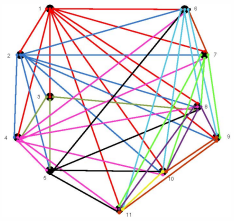
Beyond just convergence time, the value for the tunable convergence heuristic, N, was also experimented with. In our testing we observed that a suitable value for N was related to both the size of the network and the initial distribution of 1 and 0 states within the network. For each network configuration, a suitable N was experimentally chosen which ensured the network converged properly.

Table II shows the results of the hardware testing for 5, 7, and 11 motes. As would be expected, as the number of motes increases so does the convergence time. In addition, as the distribution of initial states becomes more difficult, the convergence time increases as well.

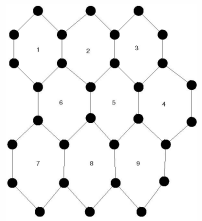
B. Simulation

In order to test our algorithm with a larger number of motes in other topologies, we also made use of the TinyOS SIMulator (TOSSIM) to gather additional results. TOSSIM simulates motes running the TinyOS platform, complete with network functionality and packet loss.

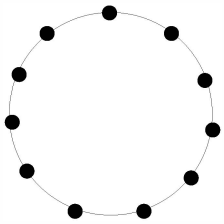
When simulating packet loss, TOSSIM takes as input a noise model as well as signal strength between motes. For our experiments we made use of the TOSSIM supplied *meyer-heavy* noise model originally derived from experiments done at the Meyer Library at Stanford University. The model includes



(a) Hardware topology



(b) Max 3 neighbors topology



(c) Ring topology

Fig. 5: Sample topologies simulated in TOSSIM

hardware noise floor readings and points of interference [10]. For the signal strength between motes we made a simplifying assumption of -55 dB for all connections.

We tested a variety of topologies. First, we replicated the topology of our hardware testbed used above for 5, 7 and 11 motes. We also simulated both max-3 neighbors and ring topologies for 5, 7, 11, 20 and 30 motes. Samples of the topologies can be found in Fig. 5.

The distribution of initial states for trials T1 - T5 follows the same pattern described for hardware testing. (T1 resembles the optimal initial state distribution of 1s and 0s while T5 resembles the worst case.)

The H-Sim section of Table III shows the hardware topology simulation results for 5, 7, and 11 motes. Recall that this topology matches that of the real hardware network described above, so we would expect similar results as in the hardware experiments. Comparing the results of the hardware and the simulation, we can see that the simulation convergence time is much larger than that of the real hardware. After investigation, we determined that this is due to the fact that the meyer-heavy noise profile results in a simulated network with much higher packet loss than found in our hardware test. This results in much higher convergence times due to the extensive retransmission that must occur for lost packets.

The Max-3 section of Table III shows the results for the max-3 neighbors topology. Fig. 5b shows a max-3 neighbors topology with various "areas" labeled. As would be expected, as the number of motes increases so does the convergence time as well as the required N. In T1, the distribution of $0's$ and $1's$ are spread uniformly across the network, and the convergence time is fast and the required N value is low. For T5, however, entire "cells" of motes (such as 1, 2, 3, etc. in the figure) are all assigned the same state, with adjacent cells having opposite assignments. In this case, the motes require more time to converge and have a higher N value.

TABLE III: Convergence time and N values for simulation experiments

	Motes	T1	T2	T3	T4	T5	AVG	N
H-Sim	5	55.4 s	67 s	69 s	74 s	80 s	69 s	7
	7	109 s	165 s	173 s	222 s	290 s	192 s	20
	11	180 s	220 s	317 s	453 s	530 s	340 s	25
Max-3	5	39 s	40 s	45 s	49 s	60 s	46.6 s	7
	7	76 s	95 s	110 s	110 s	113 s	100 s	10
	11	79 s	79 s	118 s	133 s	153 s	112 s	10
	20	108 s	195 s	227 s	240 s	353 s	201 s	15
	30	242 s	251 s	268 s	275 s	342 s	274 s	20
Ring	5	33 s	40 s	43 s	44 s	56 s	43.2 s	5
	7	43 s	49 s	59 s	66 s	110 s	83 s	5
	11	67 s	82 s	87 s	104 s	165 s	101 s	7
	20	113 s	180 s	246 s	293 s	480 s	262 s	20
	30	298 s	360 s	409 s	534 s	560 s	432 s	25

The Ring section of Table III shows the results for the ring topology. For small numbers of motes (5, 7 and 11) the convergence time as well as the N value are better than in max-3 neighbors. However, as the number of motes increases, the time of convergence as well as the value of N increase rapidly as well. In the worst case, reaching t =560 seconds for 30 motes. Much like the previous experiment, the initial states also impact convergence time. For example, with 20 motes, the difference in time between T1 (where all initial states were alternated among neighbors) and T5 (where the left and right halves of the rings had the initial states concentrated) was around 400%.

The simulation results also show that for small numbers of motes (5, 7 and 11) the convergence time as well as the N value is better in ring topology than in max-3 neighbors topology. For larger numbers of motes; however, this advantage is lost. In this case, max-3 neighbors converges faster and has a smaller N value than ring. This is due to the fact that as the ring becomes larger, it takes significantly more time for state changes to propagate completely around the ring.

V. Discussion

In this section we will discuss the complexity of our algorithm, potential improvements, and related work.

A. Complexity Analysis

The complexity of our algorithm varies depending on the topology of the network. In the case of a fully connected network, consider that the network consists of N motes. On average, each mote would send packets to $(N-1)$ other motes within the network. This makes the *average complexity* for this case:

$$O((N)(N-1)) \tag{1}$$

Which reduces to:

$$O(N^2) \tag{2}$$

B. Related Work

There is a plethora of related work on binary consensus [3], [4], [5], [6], [2], [11]. Mostefaoui et al. [11] proposed an algorithm in asynchronous systems with crash failures. In their

algorithm, every process runs a series of binary consensus subroutines sequentially to solve multivalued consensus. Binary consensus is deployed as distributed averaging on a network. The applications of this algorithm include coordination of autonomous agents, estimation, and distributed data fusion on ad-hoc or social networks. In [5], the algorithm is proven to converge to the correct solution with probability 1. In [2] the authors derive an upper-bound on the expected convergence time for a variety of network topologies, including complete graph, star, and Erdos-Renyi random graphs.

There is a large amount of existing work on routing protocols [12], [13], [14] in WSNs. In theory, these protocols could be used to create the effect of a fully connected topology and allow a different design to our algorithm. This paper is concerned with developing a binary consensus algorithm that functions without requiring the complexity of a full routing protocol. In future work, an alternative algorithm making use of a full routing protocol could be compared to this one in terms of energy efficiency and accuracy.

Most similar in concept to this work, Kenyeres et al. [15] performed a hardware implementation of the average consensus algorithm proposed in [16]. In average consensus nodes are attempting to converge on the average of all values held by nodes. They detect consensus by defining an accuracy parameter and declaring a counter that is increased whenever a node's value is changed. They assume that if the value of the node is changed in small intervals less than the defined accuracy parameter, or if the value is the same 3 times, then convergence has been achieved. Their work makes a crucial simplifying assumption that ours does not: They assume that the network topology is fully connected (every node can communicate directly with every other node). This assumption greatly simplifies their algorithm, but limits the size of the network it can support.

C. Limitations and Future Work

Our algorithm does note not guarantee a correct convergence. This is due to the fact that convergence is detected in our implementation based on the heuristic value N. Choosing an appropriate value of N reduces the probability of incorrect convergence. Our algorithm testing showed successful results of a correct convergence in both hardware and simulation, however care must be taken to choose a correct value of N when deploying this algorithm.

Future work in this area should include an analysis of this algorithm in the presence of an attacker, evaluation on a larger hardware testbed, a more thorough analysis of the effects of the N value, and applying the concepts from this algorithm to other consensus algorithms.

VI. Conclusion

In this work, we have adapted the binary consensus algorithm for use in wireless sensor networks by specifying how motes find partners to update state with as well as by adding a heuristic for individual nodes to determine convergence. We have evaluated our algorithm in hardware using 11 IRIS sensor motes and have further supported our results using the TOSSIM simulator for other topologies. The hardware as well as the simulation results show that the convergence speed depends on the topology type, the number of nodes present in the network, and the distribution of the initial 0 and 1 states. Our simulation results showed that the max-3 neighbor topology converges faster than the ring topology for networks with more than 11 nodes, while the ring topology converges faster when the number of nodes is less.

Acknowledgment

This publication was made possible by the support of the NPRP grant 09-1150-2-448 from the Qatar National Research Fund. The statements made herein are solely the responsibility of the authors.

References

[1] Y. Ruan and Y. Mostofi, "Binary consensus with soft information processing in cooperative networks," in *Proceedings of the IEEE Conference on Decision and Control (CDC 2008)*. IEEE, 2008, pp. 3613–3619.

[2] M. Draief and M. Vojnovic, "Convergence speed of binary interval consensus," in *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2010)*, San Diego California, March 15-19, 2010.

[3] A. Kashyap, T. Başar, and R. Srikant, "Quantized consensus," *Automatica*, vol. 43, no. 7, pp. 1192–1203, 2007.

[4] E. Perron, D. Vasudevan, and M. Vojnovic, "Using three states for binary consensus on complete graphs," in *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2009)*. IEEE, 2009, pp. 2527–2535.

[5] F. Benezit, P. Thiran, and M. Vetterli, "Interval consensus: from quantized gossip to voting," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2009)*. IEEE, 2009, pp. 3661–3664.

[6] A. Nedic, A. Olshevsky, A. Ozdaglar, and J. Tsitsiklis, "On distributed averaging algorithms and quantization effects," *IEEE Transactions on Automatic Control*, vol. 54, no. 11, pp. 2506–2517, 2009.

[7] CrossBow Technology Inc., "IRIS wireless measurement system datasheet," http://www.xbow.com/Products/Productpdffiles/Wirelesspdf/IRISDatasheet.pdf.

[8] Crossbow Technology Inc., "Professional kit for wireless sensor networks," http://www.xbow.com.

[9] TinyOS Community Wiki, "Mote-PC serial communication," http://docs.tinyos.net/tinywiki/index.php?title=Mote-PC_serial_communication_and_SerialForwarder&redirect=no.

[10] ——, "TOSSIM simulator," http://docs.tinyos.net/tinywiki/index.php/TOSSIM.

[11] A. Mostefaoui, M. Raynal, and F. Tronel, "From binary consensus to multivalued consensus in asynchronous message-passing systems." *Information Processing Letters*, vol. 73, no. 5-6, pp. 207–212, 2000.

[12] K. Akkaya and M. Younis, "A survey on routing protocols for wireless sensor networks," *Ad Hoc Networks*, vol. 3, no. 3, pp. 325 – 349, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870503000738

[13] L. J. Garca Villalba, A. L. Sandoval Orozco, A. Trivio Cabrera, and C. J. Barenco Abbas, "Routing protocols in wireless sensor networks," *Sensors*, vol. 9, no. 11, pp. 8399–8421, 2009. [Online]. Available: http://www.mdpi.com/1424-8220/9/11/8399

[14] S. Singh, M. Singh, and D. Singh, "Routing protocols in wireless sensor networks–a survey," *International Journal of Computer science and engineering Survey (IJCSES)*, vol. 1, no. 2, pp. 63–83, 2010.

[15] J. Kenyeres, M. Kenyeres, M. Rupp, and P. Farkas, "WSN implementation of the average consensus algorithm," in *Proceedings of the Wireless Conference 2011 - Sustainable Wireless Technologies (European Wireless)*. VDE, 2011, pp. 1–8.

[16] R. Olfati-Saber, J. Fax, and R. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215–233, 2007.