Computer Networks 79 (2015) 30–38

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Distributed algorithms in wireless sensor networks: An approach for applying binary consensus in a real testbed

Noor Al-Nakhala *, Ryan Riley, Tarek Elfouly

Qatar University, Department of Computer Science and Engineering, Doha, Qatar

ARTICLE INFO

Article history:
Received 6 February 2014
Received in revised form 15 October 2014
Accepted 17 December 2014
Available online 7 January 2015

Keywords:
Binary consensus
TinyOS
Wireless sensor networks

ABSTRACT

In this work, we realize 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 binary consensus algorithm focuses on simulation of the algorithm in a purely theoretical sense. We fill the gap between the theoretical work and real hardware implementation by modifying the algorithm to function in wireless sensor networks. This is achieved 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. Our implementation is asynchronous and based on random communication. In this work, we expand our previous implementation to test it on 139 hardware testbed. Moreover, we are able to minimize the convergence time achieving ultimate results. Our implementation show successful results and all the motes are able to converge to the expected value in very short time.

1. Introduction

Algorithms for cooperative decision making have received significant attention in recent years. In these algorithms, a network of agents seeks to reach a decision cooperatively and ensure that all nodes in the network know the final decision. The consensus problem comes when the agents should agree on a certain value. 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 measuring temperature could use binary consensus to answer the question “Is the temperature greater than 80 °C?” in order to help detect a fire in a building.

In the binary consensus problem, each node has an initial state of either 0 or 1, and the nodes should decide which one of these values are correctly held by the majority of the nodes in the network.

The existing algorithm for binary consensus has two limitations. First, it does not specify how nodes find partners to run the algorithm with. Second, it does not 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 to collect data about their environments, and can communicate with each other wirelessly [3]. Due to limitations regarding their size and power, sensor motes are computationally weak and should limit the number of packets they send. When data is transmitted by the sensor

* Corresponding author.
E-mail addresses: nalnakhala@qu.edu.qa (N. Al-Nakhala), ryan.riley@qu.edu.qa (R. Riley), tarekfouly@qu.edu.qa (T. Elfouly).

http://dx.doi.org/10.1016/j.comnet.2014.12.011
1389-1286/

motes in the network, more energy is consumed in the process of transmission than the process of computation [4].

Previous research focused only in the theoretical side of the binary consensus without going further to implement this algorithm in real sensor motes. The binary consensus algorithm can be effectively used in wireless sensor motes in numerous fields to increase the accuracy of detecting certain decisions or events.

To the best of our knowledge, there is no exiting work that deals with implementing binary consensus algorithm in real world scenario. Our goal in this work is to study binary consensus algorithm further by implementing it in real world implementation.

In [5] we implemented and tested our algorithm in real wireless sensor motes by applying it in 11 sensor motes, and further support our results with more motes and topologies in a wireless mote simulator. Our previous results of convergence time showed that convergence speed depends on the following factors: the topology, the number of motes present in the network and the distribution of the initial 0 and 1 states. 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. Moreover, we tested our implementation in a large sensor network consists of 139 motes.

2. Background

In this section we will give a brief overview of binary consensus and potential applications of it to wireless sensor networks (WSNs). We assume the reader is familiar with WSNs, and focus on binary consensus here.

2.1. 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* [6,7], 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 come the correct conclusion [7], and that there is an upper-bound on the time to convergence [1].

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

The updating protocol, as quoted from [1], 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 $x \leqslant y$, 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.

Consider the following theoretical example of how the binary consensus algorithm works, independent of the implementation methodology. Assume 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

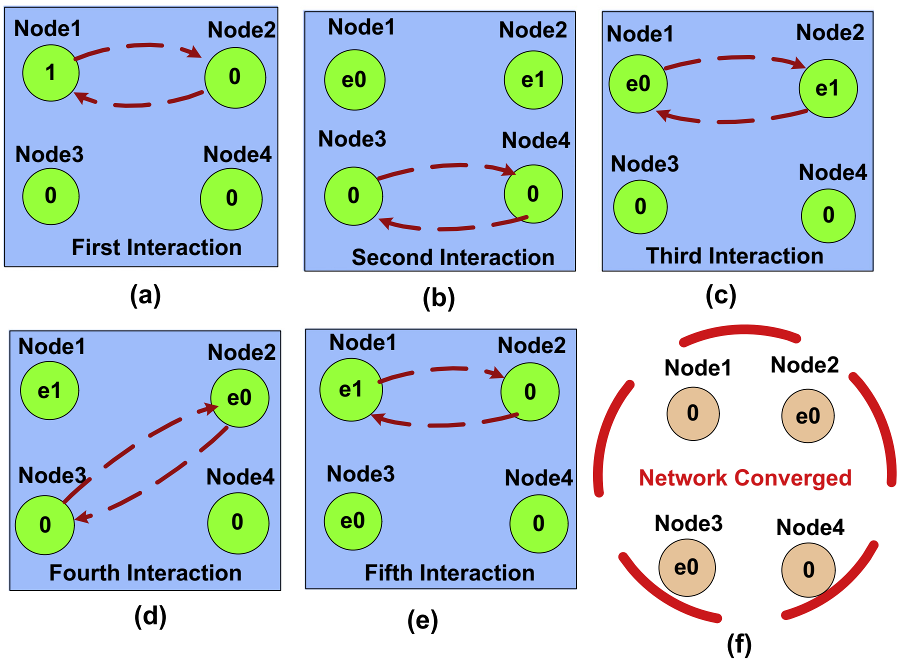


Fig. 1. Example of binary consensus algorithm functionality.

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.

2.2. 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 of having a network of sensors capable of sensing temperature in a large area such as a stadium, and using that network to control the air conditioning of that area. The temperature may vary in different parts of the area, and you would only want to adjust the AC when a majority of nodes believe the temperate is above a certain threshold. Binary consensus could be applied to such a network in order to accomplish this goal.

Another application could be the detection of gas leaks in a gas processing or storage facility. In such facilities small, transient leaks may be acceptable while people are working on the equipment and small amounts of gas is released. If a network of gas sensors was deployed in a small area, one or two of them detecting gas may be a non-issue, while if the majority detect gas then a larger gas leak is certainly occurring. Binary consensus could be used to determine the majority opinion of these sensors.

3. Design and implementation

While the binary consensus algorithm described in [1] and Section 2.1 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 WSNs. 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.

3.1. 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 illustrates a stage transition diagram of our communication algorithm.[1] Table 1 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

[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.

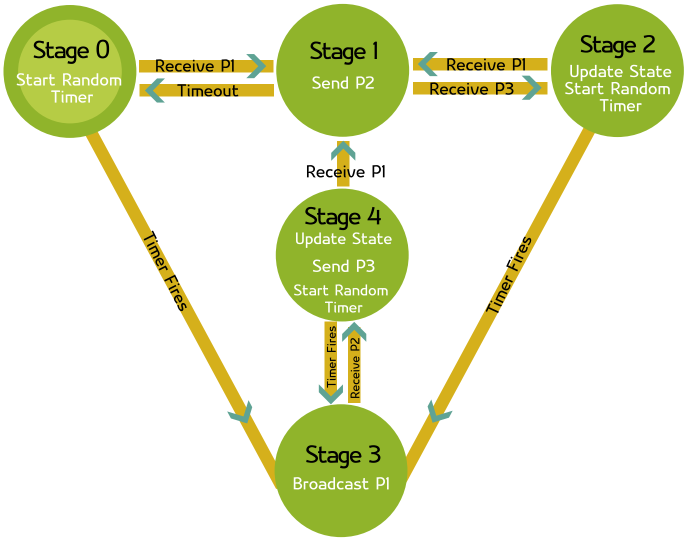


Fig. 2. Stage transition diagram of the communication algorithm.

Table 1
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

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.

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 acknowledgment features built into the radio unit of our sensor 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.

3.2. 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 value called N that plays important role in estimating convergence. Whenever a mote updates its state, it also keeps track of the last N states that it has held. If the last N states $\in \{0, e_0\}$ or $\in \{e_1, 1\}$, then the mote

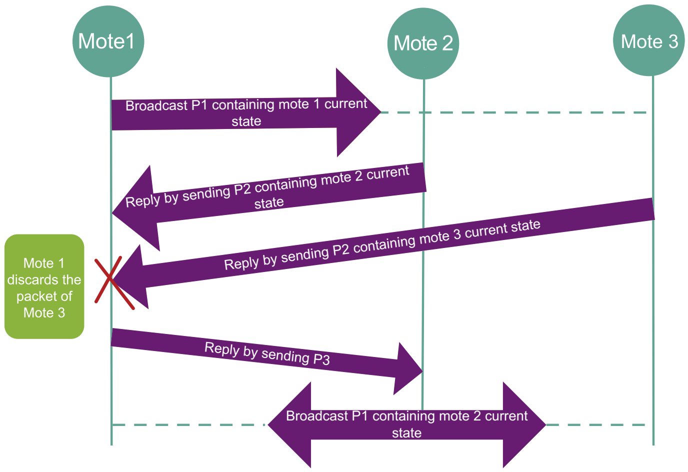


Fig. 3. Simple example of mote communication state updates.

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 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. Similarly, if after convergence a mote changes its state, it will broadcast its new state and other motes who receive the broadcast packet will reactivate their timers and the communication will start again.

During experimentation, for each network type a suitable N was manually chosen which ensured the network converged properly. Choosing the value of N is critical and affects the convergence time. If N is very high, then unnecessary extra packets will be sent in the network and this will increase the convergence time and consume energy. If it is too low, then convergence could occur prematurely and incorrectly.

4. Experiments

We have implemented our algorithm in [5] using IRIS family of sensor motes from the MEMSIC corporation [8,9]. 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. On the mote itself, our application required 15 kbytes of ROM storage and 600-bytes of RAM at runtime.

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

4.1. Testing the algorithm in a larger sensor network

We tested our algorithm in a large network of 139 motes. In order to achieve this, we used the wireless sensor network testbed provided by Indriya [10]. Indriya is a three-dimensional wireless sensor network deployed across three floors. The status of each mote during the tests was recorded through a serial port attached to each mote in the testbed. This allows us to run our wireless protocol while still logging the activity of all the motes.

We were able to minimize the convergence time reaching 139 motes. As in our algorithm the motes communication is based on random timer, we tweaked the random timer by minimizing it to the best value that will guarantee that the convergence will occur. This also affected the tunable convergence heuristic value N (described previously in Section 3.2) that is used to estimate the convergence. We ran several tests in order to choose the best value for the random timer and for choosing the best value for N.

We tested the algorithm for 25, 50, 75, 100, 125 and 139 motes as shown in Table 2. The initial states 0, 1 were distributed randomly so that the majority value has a percentage of 60%. The motes during the tests were chosen carefully, so that the network includes motes from all three floors. The value for the tunable convergence heuristic, N was also experimented with. During the tests we were able to minimize the heuristic value N, which indicates convergence, to $N = 10$ for 139 motes. In these tests we generalized the N value for 25, 50, 75, 100, 125 and 139 to be 10 since this value is best fit for 139 motes. This led to having a convergence time as shown in Table 2 reaching on average of $T = 39.4$ s when the number of motes is 139. We also found that the convergence time and the heuristic value N depend on the random timer that we used in our design. When minimizing the random timer, the value N will be minimized and both (the random timer and N value) will result on minimizing the convergence time.

Table 2
Convergence time and N values for 139 hardware motes.

Motes	T1 (s)	T2 (s)	T3 (s)	T4 (s)	T5 (s)	AVG (s)	N
25	30	30	35	35	37	33.4	10
50	27	28	30	31	32	29.6	10
75	24	24	26	27	27	25.8	10
100	23	23	27	27	28	25.6	10
125	27	29	29	30	31	29.2	10
139	35	37	40	42	43	39.4	10

Fig. 4 depicts the convergence time results. While one might expect that convergence time would uniformly increase as the number of motes increases, instead the graph has an convex shape that shows that convergence time initially decreases as the number of motes increase before then beginning to increase rapidly. The reason for this has to do with the density of the motes in the network. A small number of motes, such as 25, scattered across such a large area is not a very dense network. This means that it will take longer for sufficient mixing of initial states to occur and cause convergence. At around 100 motes, the network reaches sufficient density and adding additional motes causes the expected rise in convergence time.

4.2. Packets sent

Fig. 5 shows the numbers of packets transmitted within the network when implementing our algorithm on 25, 50, 75, 100, 125 and 139 motes with and without considering lost packets. As can be seen, the number of packets grows linearly with respect to the number of motes, independent of the density or convergence time.

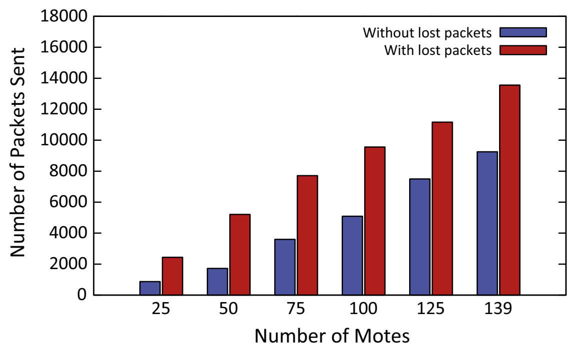


Fig. 5. Total number of packets sent to reach convergence.

4.3. Simulation

In order to test our algorithm with a variety of specific 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 hardware noise floor readings and points of interference [11]. For the signal strength between motes we made a simplifying assumption of −55 dB for all connections.

4.4. Max-3 neighbors topology vs. ring topology

We tested and 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. 6.

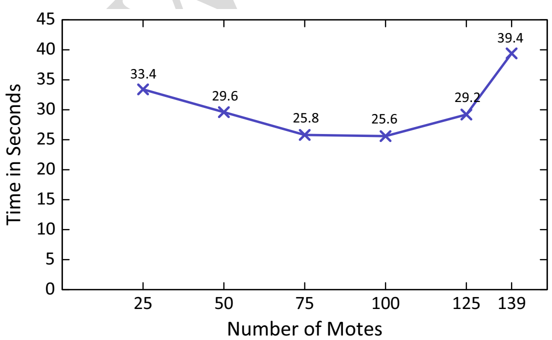


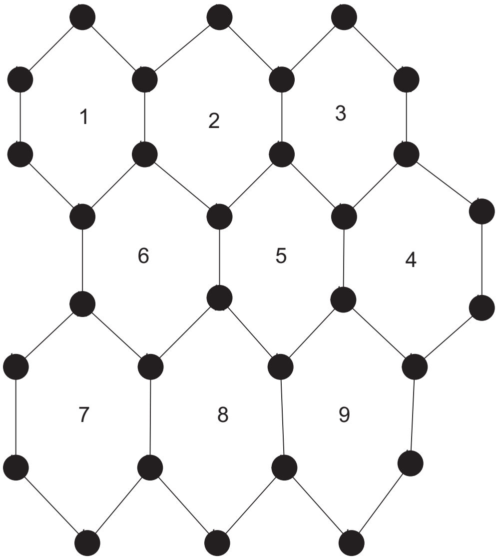
Fig. 4. Average convergence time for large hardware motes.

Three trials (signified T1–T3 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 quickly. In T3, 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 made use of distributions that slowly shift from T1 to T3. (T1 is the "easiest" distribution, T2 is slightly more difficult, etc.).

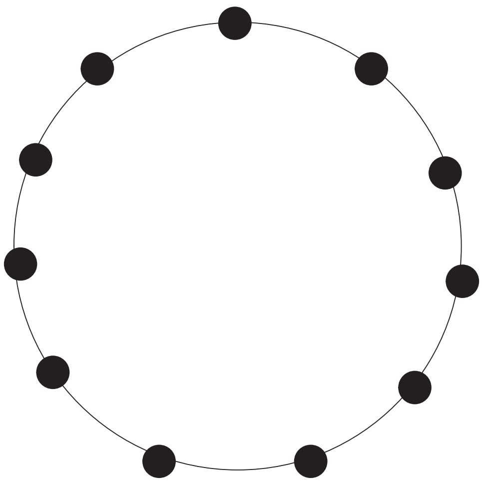
The max-3 section of Table 3 shows the results for the max-3 neighbors topology. Fig. 6a shows a max-3 neighbors topology with various "areas" labeled. As would be expected, as the number of motes increases so does the convergence time as well as the required N. In T1, the distribution of 0's and 1's 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 and 3 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.

The Ring section of Table 3 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 convergence time as well as the value of N increase rapidly as well. In the worst case, reaching $t = 123$ s 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 T3 (where the left and right halves of the rings had the initial states concentrated) was around 250%.

Fig. 7 compares the convergence time between max-3 neighbors and ring topologies. The figure shows 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;



(a) Max 3 neighbors topology



(b) Ring topology

Fig. 6. Sample topologies simulated in TOSSIM.

Table 3
Convergence time and N values for simulation experiments.

Motes	T1 (s)	T2 (s)	T3 (s)	AVG (s)	N
Max-3					
5	4.2	4.7	5.8	4.9	7
7	8.4	9.5	11.6	9.8	10
11	8.2	14.4	16	12.9	10
20	14.4	18.6	23.9	19	15
30	20.9	26.9	32.5	26.9	20
Ring					
5	4.1	4.7	4.9	4.6	5
7	4.6	5.5	6	5.4	5
11	9.1	12.6	14.3	12	7
20	32.1	51.2	76.2	52.5	40
30	32.7	59.6	123	89	50

however, this advantage is lost. In this case, max-3 neighbors converges faster and increases slowly and has a smaller N value than ring. This is likely due to the fact that as the ring becomes large, it takes significantly more time for state changes to propagate completely around the ring. Which results in a rapid increasing in convergence time for ring topology when the number of motes increases to more than 12.

This is the because the shortest path between two motes sitting in opposite side of the ring topology grows with complexity of $O(N)$ (with a constant of roughly $\frac{1}{2}$) while the shortest of max-3 neighbors grows with complexity $O(\sqrt{N})$. As such, there is an intersection point on the graph at roughly 11 motes where the shortest path for max-3 neighbors will be less than that of ring, and hence the convergence time is lower as well.

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.

5. Related work

There is a plethora of related work on binary consensus [7,1,12,6]. Mostefaoui et al. [12] 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 [7], the algorithm is proven to converge to the correct solution with probability 1. In [1] 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. In [2] the authors proposed a method of information processing aimed at improving consensus convergence over noisy channels using Gaussian noise models.

There is a large amount of existing work on routing protocols [13–15] in WSNs. In theory, these protocols could be

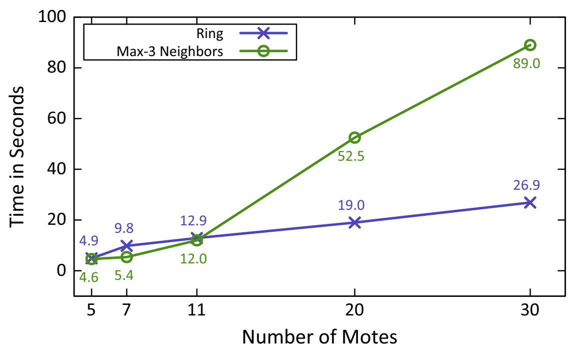


Fig. 7. Comparison of average convergence time between max-3 neighbors and ring topologies.

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. [16] performed a hardware implementation of the average consensus algorithm proposed in [17]. 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 mote's value is changed. They assume that if the value of the mote 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 mote can communicate directly with every other mote). This assumption greatly simplifies their algorithm, but limits the size of the network it can support. Moreover, their algorithm is synchronous and forces the update to be synchronous which limits the usage of their algorithm.

6. Discussion

In this section we will discuss the factors related to convergence time, applicability of our work to other distributed algorithms, and alternative implementation ideas.

In [5] we found that convergence speed depends on the topology type, the number of motes present in the network and the distribution of the initial 0 and 1 states. In this study we found that the convergence time depends on the network density as well. Convergence occurs more quickly in dense networks than in sparse networks. This is due to the fact that when the network gets more dense, the number of links between the motes increases, meaning that each mote will have a larger number of neighbors. In this situation, the distance between motes on opposite sides of the network decreases, meaning less state exchanges are required for states to propagate completely inside the network, hence lowering the time to convergence.

While the algorithm described in this work is specific to binary consensus, a number of the principals used in its design would apply to other cooperative decision making algorithms as well. Distributed average consensus [18], for example, follows a similar model to binary consensus in that it involves individual nodes communicating with each other and updating state. As such, our approach of communicating with random neighbors would be applicable. In addition, it is important to note that binary consensus, as a primitive, can be used to solve other problems such as multivalued consensus [12]. This means that our existing algorithm can be easily expanded to solve those types of problems as well.

This work is not primarily concerned with minimizing the number of packets sent (and hence energy consumption), but there are other approaches that could be used to try and further optimize it. One approach is to organize motes into clusters in order to minimize the number of motes involved in the binary consensus exchange. While our own related work in this area does indeed reduce the number of packets sent [19], it does not consider the overheard of a distributed clustering algorithm, which would be non-trivial. Another idea is to incorporate a formal transmission schedule with time sharing in order to minimize the number of lost packets seen in Fig. 5. In [16] TDMA was used to this end, but because scheduling was done on the level of the entire network, this would greatly increase the time needed for convergence in our approach as the network increases on size. Proper application of time sharing, therefore, would need to break the network up into smaller, independent regions and schedule packet transmission within them.

7. Conclusion

This work represents a new starting point for a real implementation of one of the cooperative algorithms in wireless sensor network which is the binary consensus algorithm. The concept of binary consensus can be used in real life applications in different fields to increase the accuracy of a certain decision. We have adapted the binary consensus algorithm for use in wireless sensor networks. This is achieved by specifying how motes find partners to update state with as well as by adding a heuristic for individual motes to determine convergence. In this work, we were able to minimize the convergence time. We evaluated our algorithm successfully in 139 hardware motes and the network converged in very short time compared to our previous results in [5]. The results also showed that when the network is getting more dense the convergence time will be minimized. It also showed that the convergence speed depends on the number of motes presented in the network. Our results in this work completed our previous results in [5]. During the experiments none of the motes failed and our algorithm converged correctly. In traditional binary consensus, individual nodes do not have a stop condition, meaning nodes continue to transmit even after convergence has occurred. In WSNs however, this is unacceptable since it will consume power. So in order to save power sensor motes should stop the communication when the whole network converges. For that reason we have designed a tunable heuristic value N that will allow motes to estimate when convergence has occurred. The hardware as well as the simulation results show that the convergence speed depends on the topology type, the number of motes 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 12 motes, while the ring topology converges faster when the number of motes is less than 12. Convergence speed depends on the topology type, the number of motes present in the network and the distribution of the initial 0 and 1 states and the network density. Dense networks converge faster than sparse networks.

Acknowledgments

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. We would like to thank the School of Computing, at the National University of Singapore, for providing the Indriya testbed used in our experiments.

References

[1] M. Draief, M. Vojnovic, Convergence speed of binary interval consensus, SIAM J. Control Optim. 50 (3) (2012) 1087–1109.
[2] Y. Ruan, 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.
[3] Y. Choi, Y. Jeon, S. Park, A study on sensor nodes attestation protocol in a wireless sensor network, 2010 The 12th International Conference on Advanced Communication Technology (ICACT), vol. 1, IEEE, 2010, pp. 574–579.
[4] L. Yong-Min, W. Shu-Ci, N. Xiao-Hong, The architecture and characteristics of wireless sensor network, International Conference on Computer Technology and Development, 2009, ICCTD'09, vol. 1, IEEE, 2009, pp. 561–565.
[5] N. Al-Nakhala, R. Riley, T. Elfouly, Binary consensus in sensor motes, in: 9th IEEE International Wireless Communications and Mobile Computing Conference (IWCMC 2013), IEEE, Cagliari, Italy, 2013.
[6] E. Perron, D. Vasudevan, 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.
[7] F. Benezit, P. Thiran, 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.
[8] CrossBow Technology Inc., IRIS Wireless Measurement System Datasheet. <http://www.xbow.com/Products/Productpdffiles/Wirelesspdf/IRISDatasheet.pdf> (last accessed 23.12.12).
[9] Crossbow Technology Inc., Professional Kit for Wireless Sensor Networks. <http://www.xbow.com> (last accessed 23.12.2).
[10] M. Doddavenkatappa, M.C. Chan, A. Ananda, Indriya: A low-cost 3d wireless sensor network testbed, in: Testbeds and Research Infrastructure, Development of Networks and Communities, Springer, 2012, pp. 302–316.
[11] TinyOS Community Wiki, Tossim Simulator. <http://docs.tinyos.net/tinywiki/index.php/TOSSIM> (last accessed 23.12.12).
[12] A. Mostefaoui, M. Raynal, F. Tronel, From binary consensus to multivalued consensus in asynchronous message-passing systems, Inf. Process. Lett. 73 (5–6) (2000) 207–212.
[13] K. Akkaya, M. Younis, A survey on routing protocols for wireless sensor networks, Ad Hoc Networks 3 (3) (2005) 325–349, http://dx.doi.org/10.1016/j.adhoc.2003.09.010. <http://www.sciencedirect.com/science/article/pii/S1570870503000738>.
[14] L.J. Garca Villalba, A.L. Sandoval Orozco, A. Trivio Cabrera, C.J. Barenco Abbas, Routing protocols in wireless sensor networks, Sensors 9 (11) (2009) 8399–8421, http://dx.doi.org/10.3390/s91108399. <http://www.mdpi.com/1424-8220/9/11/8399>.
[15] S. Singh, M. Singh, D. Singh, Routing protocols in wireless sensor networks–a survey, Int. J. Comput. Sci. Eng. Survey (IJCSES) 1 (2) (2010) 63–83.
[16] J. Kenyeres, M. Kenyeres, M. Rupp, 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.
[17] R. Olfati-Saber, J. Fax, R. Murray, Consensus and cooperation in networked multi-agent systems, Proc. IEEE 95 (1) (2007) 215–233.
[18] L. Xiao, S. Boyd, S.-J. Kim, Distributed average consensus with least-mean-square deviation, J. Parallel Distr. Comput. 67 (1) (2007) 33–46.
[19] N. Al-Nakhala, R. Riley, T. Elfouly, Clustered Binary Consensus in Sensor Motes, European Wireless 2014.

Noor Al-Nakhala received the B.S. degree in Computer Engineering and the M.S. degree in Computing in 2012 from Qatar University. She is a research assistant with the Computer Science and Engineering Department at Qatar University. Her current research interests include Wireless Sensor Networks (WSNs), Networking, Embedded Systems, Distributed Systems and Intelligent Control Systems.

Ryan Riley received the B.S. degree in Computer Engineering and the Ph.D. degree in Computer Science in 2009 from Purdue University. He is an Assistant Professor of Computer Science with Qatar University, Doha. His current research interests include virtualization technologies, malware, operating systems, and security.

Tarek Elfouly is an assistant professor at Qatar University. He obtained his Ph.D. from the University of Franche Comte in France. He has several active QNRF and industry grants related to wireless networking and security. He has 12 years of experience in network and security research. He published over 30 papers, more than half of them are related to wireless sensing and network security. He has many projects under development related to assistive technologies for people with disabilities. His projects won many national and regional awards.