# Clustered 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 Elfouly
Qatar University
Doha, Qatar
tarekfouly@qu.edu.qa

***Abstract*—In this work, we extend and adapt the binary consensus algorithm to operate using clusters in a wireless sensor network. Binary consensus is a decision making algorithm used to cause distributed entities to agree on the majority opinion held by the group when posed with a true or false question. Clustering is a common technique used in WSNs to increase the overall lifetime of the network by reducing energy consumed due to communication. Our clustered binary consensus implementation is tested using a large WSN testbed, and the experimental results show that incorporating clustering into the binary consensus algorithm reduces the time required for motes to reach consensus and lowers the number of packets sent in the network by approximately 94%.**

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

## I. Introduction

Cooperative decision making algorithms are used to solve the problem of consensus on a distributed computing network without making use of a central sink. In trying to reach consensus, a network of agents are attempting to corporately and individually determine a value that is contributed to by all motes in the network. For example, a network of temperature sensing motes could use a cooperative decision making algorithm to determine the average temperature seen by all motes in the network. One specific algorithm in this field is binary consensus [1], [2]. In binary consensus, all motes are attempting to learn the majority opinion of a yes or no question. Continuing the temperature example, all motes might want to know the answer to the question "Do the majority of motes in the network see a temperature higher than 27C?"

Work in the area of distributed, cooperative decision making algorithms tends to focus on formal mathematical models and studies involving simulation [3]. This is because of the many complexities involved in testing the algorithms on actual distributed computing devices such as wireless sensor motes. Real world factors such as energy usage, lost packets, mote reach-ability, and others make the work more difficult than in simulation.

In our previous work [4] we adapted the binary consensus algorithm to operate in a WSN. Our implementation involved all motes communicating randomly with their neighbors and using a set of predefined rules [1] to exchanges their current states and eventually reach consensus. The implementation was tested using a variety of network topologies such as ring, max-3 neighbors and random. The experiments focused on exploring the convergence time of the network with varying topologies, initial state distributions, and number of motes.

In this work, we further develop binary consensus in WSNs by reducing energy consumption and convergence time through the use of clustering. This involves adapting the algorithm to allow a cluster head to efficiently compute the correct states of its cluster members and exchange those states properly with other cluster heads. We test our implementation using a 139-mote TinyOS-based testbed and verify that our new implementation is indeed faster and more energy efficient than our previous work.

## II. Related Work

In this section we will present related work in the areas of binary consensus, implementations of consensus algorithms, energy Usage in WSNs and clustering in WSNs.

**Binary Consensus** There is a significant amount of related work in the area of binary consensus [1], [5], [6], [7], [8], [9]. Mostefaoui et al. [9] studied the algorithm with crash failures to solve multivalued consensus. This is achieved by running a series of binary consensus subroutines in order to determine the average. They used binary consensus as distributed averaging on a network. The use of this algorithm might include social networks, distributed data fusion, and coordination of autonomous agents.

In [7], they proved that the algorithm converges correctly with probability 1.

An upper-bound on the convergence time of binary consensus is derived in [1] for a variety of topologies such as complete graph, star, and Erdos-Renyi random graphs.

**Consensus Algorithm Implementations** In [10], the authors have done a hardware implementation of the average consensus algorithm proposed in [11]. In average consensus nodes converge to the average of all the values held by the network. In their solution, the algorithm reaches consensus by defining an accuracy parameter and declaring a counter that will increment once the mote's value is changed. In their assumption, if the value changes in small intervals less than the accuracy parameter, or if the value did not change 2 times, then the motes have converged. They made a simplifying assumption of a fully connected topology which limited the size of the network. Moreover, the usage of their algorithm in WSNs is limited because their proposed algorithm is synchronous and forces the updates to be synchronous.

Another work done in the area of hardware implementation of consensus algorithms is proposed in [12]. The authors performed a hardware implementation of the decentralized consensus algorithm. The decentralized consensus algorithm is also known as self-organization since the nodes have the ability to self organize and collect data over the network. The authors have continued the theoretical work done in [13] to function well in a hardware implementation with discrete time. They have also adjusted the number of iterations appearing in [14] in order to achieve consensus in less time. They implemented their approach in 12 sensor motes for fully and partially connected topologies.

**Energy Usage in WSNs** Significant research effort has focused on finding ways to optimize sensor energy consumption in the network by making it a priority to minimize the number of packet sent within the sensor network. For example, [15] produces a model that controls the number of packets sent in the network resulting in the reduction of the energy consumed. This is achieved by providing a way for the transmitter to dynamically regulate its transmission power in a way that the requested signal to noise ratio can be accepted at the receiver. In the end, total energy consumption in the sensor network was reduced by 15% to 38%.

In [16] the authors propose an adaptive energy saving and reliable routing protocol which aims to adjust the node routing to other nodes since each node may contain several routes to the destination. In this solution, only one route for a destination is kept in the routing table and the selection of best route is based on link weight.

Watteyne et al. [17] propose a reactive approach to neighbor detection that reduces energy usage by removing periodic `Hello` messages between neighbors using a handshaking scheme combined with a energy efficient MAC protocol.

**Clustering** There has been a lot of work in the area of clustering. The most famous WSN clustering algorithm is LEACH [18]. In LEACH the nodes decide to be a CH based on a probability and the normal nodes join their clusters based on the minimum communication energy. CH selection changes periodically between the nodes by choosing a random number between 0 and 1. The chance for a CH to be selected again if the number is less than the calculated threshold. Because selection of the CH is based on probability, the chance of selecting a node with low energy as a CH is increased. As when the CH dies, the whole cluster becomes useless.

In [19] the authors proposed HEED, which improves the life time of the network over LEACH. The CH is selected by adding extra network information such as residual energy. Then the CHs send messages to their neighbors including a secondary constraint that measures the node degree. This is used to let the normal nodes chose the best cluster to join. After that, the selection of the CHs in later rounds is based on the probability correlated with residual energy. In HEED, the selection of low energy nodes as CHs in heterogeneous environments may give larger probability than selecting the high energy nodes.

## III. Background

In this section we will give a brief overview of binary consensus and our previous work on binary consensus in wireless sensor networks.

### A. Binary Consensus

While there are many distributed, cooperative decision making algorithms currently being researched, in this work we focus on the problem of *binary consensus* [5], [6], [7], [8]. In binary consensus, each node holds one of the two states, either 0 or 1, and the algorithm allows each individual node to know which value is held by the majority of nodes in the network. Previous theoretical research has demonstrated important properties of the algorithm. In [1] it was shown that there is an upper-bound to convergence time and in [7] it was proven that binary consensus always reaches the correct conclusion.

In binary consensus, nodes inside the network start with their initial assumption, 0 or 1, and then they communicate with each other and update their states based on an updating protocol. Convergence occurs when all the nodes inside the network agree on the majority opinion. In general, when two nodes in the network communicate together they update their current state based on the state of their partner. Then, they communicate with a different partner and updates their state again. After this has occurred enough times, all nodes end up agreeing on the majority opinion.

There are four valid states that a node might hold at any given moment [4]:

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.

When two motes communicate together in order to exchange and update their states, they follow the following updating protocol, quoted from [1]:

> 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

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

If all nodes inside a network have states belonging to 0 or $e_0$, then the network has converged to opinion 0, which means 0 was initially held by the majority of nodes. Similarly, if all the nodes hold states belonging to 1 or $e_1$, then the network has converged to opinion 1, which means 1 was initially held by the majority of nodes.

### B. The Usage of Binary Consensus in Real World Applications

When applied to real scenarios, binary consensus can be used in situations where motes individually determine a binary answer to a question, and yet must determine the decision held by most motes in the network. For example, consider a scenario where there is a network able to detect the presence of an object by using camera. In this case, the binary states would be "I see the object" or "I don't see it". Binary consensus could be used to determine when a majority of sensors see the item. Another scenario could involved a network of sensors capable of detecting the spread of gas leaks in a refinery situation. In this, the binary states would be whether or not a mote senses gas. Binary consensus could help determine when the leaking gas has spread to the majority of sensors.

### C. Our Previous Work

In [4], we took a mathematical model for binary consensus [1] and adapted it for use in WSNs. This involved modifying the algorithm to include a stop condition (the original algorithm runs forever, which is inefficient in WSNs) and a method for motes to determine who to exchange states with (the original algorithm assumed a fully connected network, which is unreasonable in WSNs).

A stop condition was added by including a tunable heuristic variable, $N$, which is used to allow a mote to estimate when convergence has occurred. If a mote has not substantially changed its state in the last $N$ state updates, then it assumes convergence has occurred and stops initiating communication with other motes. It will still respond to other motes, however. In this way, all motes in the network eventually stop communicating and convergence is finalized.

Motes determine who to communicate with by randomly selecting a free neighbor. This is done by having motes periodically wake up, broadcast their state, and wait for a response. When motes receive a broadcast from a neighbor, they reply after a random timeout value. This ensures that, over time, a mote communicates with all of its neighbors.

We tested our algorithm successfully on 11 hardware sensor motes and further supported our results with simulation by testing our algorithm in a max-3 neighbors topology, a ring topology and a simulated hardware topology with 30 motes. In follow-up testing, we extended our results to test our algorithm on a large hardware sensor motes of 139 motes and we were able to further lower the convergence time.

## IV. Communication

In order to adapt binary consensus to properly make use of a cluster configuration, a number of modifications are required.

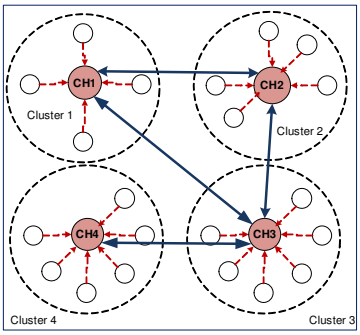


Fig. 1: Cluster formation

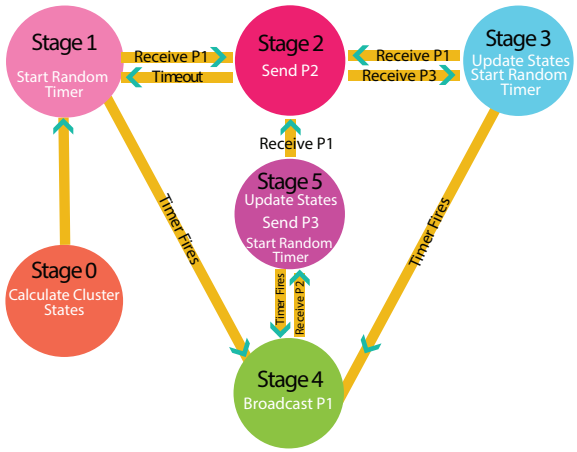


Fig. 2: Intra-cluster communication stage transition diagram

First, communication must be split into inter-cluster and intra-cluster. Second, the state update rules and calculations must be modified to be run by cluster heads on behalf of their cluster members.

We would like to note that the technique used to produce clusters is not the focus of this work. We assume that the motes are already grouped into clusters using an existing clustering scheme, and we will focus on how clustering impacts binary consensus.

### A. Inter-Cluster Communication

When the algorithm begins, all the motes within a cluster send their current state directly to their cluster head (CH). Motes communicate only with their CH, not directly with each other.

Fig 1 shows the clustering model we are assuming. The network is split into clusters where each cluster has a CH that is able to directly communicate with each of its members. We do not consider how clusters are formed in this work. We assume that they already exist, and our algorithm simply makes use of them. Clusters could be formed using existing clustering techniques such as LEACH [18] or HEED [19].

### B. Intra-Cluster Communication

After all the motes have sent their states to the CH, the CHs communicate together using a modified version of binary

TABLE I: Packets used during intra-cluster communication. CH1 and CH2 are cluster heads in the communication.

| Packet | Payload | Description |
|---|---|---|
| P1 | CH1 members' states | CH1 sends this packet to all CHs in range. |
| P2 | CH2 members' states | CH2 replies to CH1 by sending this packet. |
| P3 | - | CH1 sends this packet to CH2 in order to confirm that its states update was successful. |

consensus in order to reach convergence for the network as a whole.

Because the CHs don't have an intrinsic knowledge of the identity of each other, in our system they perform a random broadcast to their neighbors in order to find a CH partner to update states with.

Fig. 2 shows a stage transition diagram for an individual CH running the algorithm. Table I describes the types of packets sent and received during the algorithm. The stages are described as follows:

- *Stage 0*: This stage starts after all the normal motes have sent their states to their respective CH. Each CH runs a consensus algorithm (described later in Section V) that calculates the converged states of all motes within the cluster. Next, the CH always transitions to Stage 1.
- *Stage 1*: In this stage, the CH starts a random timer that will decide when it should wake up and perform a $P1$ broadcast in an attempt to find a neighboring CH partner to exchange states with. If a CH stays in this stage when the timer fires, it will transition to Stage 4. However, if it receives a broadcast ($P1$) from another CH, then it will transition to Stage 2.
- *Stage 2*: When a CH in Stage 1 receives $P1$, it transitions to Stage 2. The CH will reply to the sender of the broadcast with a $P2$ containing the current binary consensus states of all of its members. It then waits for a reply. If it receives $P3$ in reply, then it transitions to Stage 3. If not, then after a timeout period it assumes that its partner no longer wishes to communicate with it, and transitions back to Stage 1.
- *Stage 3*: Once the CH receives $P3$, it knows that its partner does want to update states. In addition, it also knows the current states held by its partner (they were sent previously with $P1$). At this point, the CH computes the new states of itself and its partner using the algorithm in Section V. The mote then starts a random wake-up timer and resumes listening for another partner to exchanges states with.
- *Stage 4*: This stage is entered if a mote's random timer goes off and it needs to broadcast to its neighbors. In this stage the mote broadcasts $P1$, which contains all of its current states. Once it receives a $P2$ reply from a neighbor, it shifts to Stage 5 in order to proceed with exchanging states with that partner.
- *Stage 5*: This stage strongly resembles Stage 3, just from the perspective of the other partner. In this stage the CH uses the algorithm of Section V to compute new states for itself and its partner. It then updates its own states, sends $P3$ to its partner in order to confirm the update has occurred, and starts a random wake-up timer while listening for another partner to exchange states with.

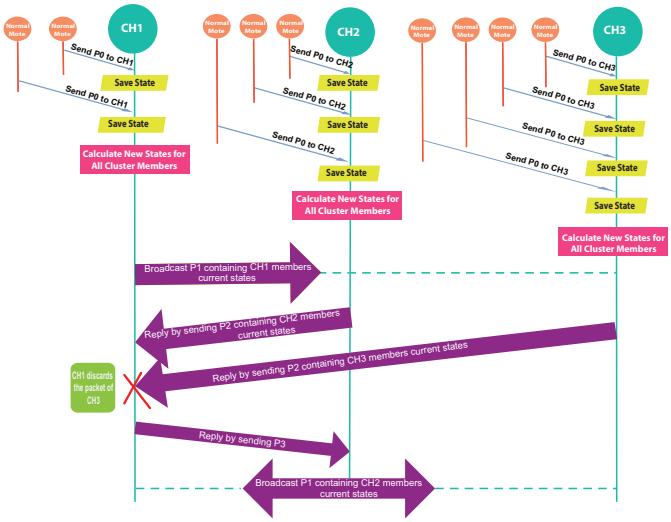


Fig. 3: Example of communication

## C. Example

Fig. 3 illustrates a simple example covering both inter- and intra-cluster communication.

Assume that we have 3 CHs: 1, 2, and 3. Each CH belongs to one cluster and performs the task of updating states for its members.

After initialization, each normal mote sends a packet containing its initial state to its CH. For example, normal motes belonging to CH1 send their states to CH1 and it saves the state of each member. Then, each CH computes the converged states for all members belonging to its cluster and starts a random timer to start the intra-cluster communication.

In this example, CH1's timer goes off before any of the other CHs. So, it broadcasts $P1$ to all of its CH neighbors. CH2 and CH3 both receive the broadcast from CH1 and both reply. However, the reply from CH2 is received first, so the reply from CH3 is ignored.

CH1 receives the $P2$ reply from CH2 and computes its own new states based on the states of CH2. Next, CH1 sends $P3$ to CH2, who receives it and computes its own new states.

In this example we have not discussed packet loss for the sake of clarity. We use the acknowledgement feature that is already built in the radio unit of the motes in order to automatically acknowledge and resend lost packets.

# V. Binary Consensus Clustering Rules

Now that we have discussed how motes communicate, we will present the computations performed by CHs during that communication.

In standard binary consensus, individual motes exchange and update states based on fixed rules. In clustered binary consensus, those rules are modified to allow a CH to efficiently execute them *on behalf of* all the motes in its cluster.

There will be two different types of computations. First, a CH may compute the binary consensus states for all of its members (inter-cluster computation). Second, a CH may compute the binary consensus states for all of its members and the members of a partner cluster (intra-cluster computation).

### A. Inter-Cluster Updating Rules

Within a cluster, the initial state of each member will be either 0 or 1. The role of the CH is to calculate the new states of all the members based on the majority opinion within the cluster. The CH is effectively performing standard binary consensus on behalf of its members. This prevents members within a cluster from needing to communicate directly and perform the computation themselves.

Given a set of 0 and 1 states $S = (s_1, s_2, s_3, ..., s_n)$, a new set of states $X = (x_1, x_2, x_3, ..., x_n)$ is computed that reflects the converged set of states that would occur within the cluster if all motes within the cluster were allowed to run binary consensus to completion.

The states in $X$ are defined by the following equations:

$$N = M - m \tag{1}$$

Where $M$ is the number of nodes that initially hold the majority opinion in $S$, $m$ is the number of nodes initially hold the minority opinion in $S$, and $N$ is the number of states in $X$ that hold the majority opinion (0 or 1) from $S$.

$$U = T - N \tag{2}$$

Where $U$ is the number of nodes that should hold undecided state $e_0$ or $e_1$ based on the majority opinion of $S$, and $T$ is the number of members inside a cluster.

For example, assume there exists a cluster $C$ consists of 6 members $m1$, $m2$, $m3$, $m4$, $m5$, $m6$, initially holding states $(0, 0, 0, 0, 1, 1)$. In this case, the majority of nodes in the cluster hold the opinion 0. The number of nodes initially holding 0 is four, while the number of nodes holding the minority opinion is two. Using equations 1 and 2 above we find that $N = 2$ and $U = 4$. This means that in the final set of states, two motes should hold value 0 (the majority opinion) and four motes should hold value $e_0$ (the undecided state closest to the majority opinion).

The new states will be $(0, 0, e_0, e_0, e_0, e_0)$. This set of states reflects that the cluster, if it were to run standard binary consensus, would converge to 0 as the majority state.

### B. Intra-Cluster Updating Rules

Now that we see how a CH computes converged states, we will look at how the states of two different clusters can be updated together. A CH performing this computation would have two sets of states to compute and merge: Its own states, and the states of its partner.

Performing this calculation is straightforward. The CH simply merges the two sets of states into one list, performs that same computation as used for inter-cluster updating, and then splits the final list between itself and its partner.

For example, assume there exists two cluster $C1$ and $C2$, each consisting of 6 members, holding states $C1 = (0, 0, e_0, e_0, e_0, e_0)$ and $C2 = (1, 1, 1, 1, e_1, e_1)$. The states of both clusters are concatenated together to form one list $C = (1, 1, 1, 1, e_1, e_1, 0, 0, e_0, e_0, e_0, e_0)$.

The number of nodes initially hold the majority opinion 1 is four, while the number of nodes holding the minority opinion 0 is two. Therefore, once again using equations 1 and 2 we find that $N = 2$ and $U = 10$. This implies that the new states in cluster $C$ are:
$C = (1, 1, e_1, e_1, e_1, e_1, e_1, e_1, e_1, e_1, e_1, e_1)$

Next, $C$ is simply split back into $C1$ and $C2$ with states:
$C1 = (1, 1, e_1, e_1, e_1, e_1)$
$C2 = (e_1, e_1, e_1, e_1, e_1, e_1)$

## VI. Experiments

We implemented clustered binary consensus in TinyOS2.x. We tested our algorithm using the Indriya mote testbed at the National University of Singapore [20]. Indriya contains 139 TelosB motes deployed across three floors of the School and Computing.

We tested our algorithm using all 139 motes, but with each experiment varied the number of clusters. Tests were performed with 10, 14, 19, 24, 28, 33 and 40 clusters. Each test was repeated 5 times to ensure consistent results. The initial states of 0 and 1 were distributed randomly in such a way that 65% of the total motes held the majority opinion. The reason for choosing 65% is to have a network with a clear majority opinion, although the impact of initial state distribution and how it affects the convergence has been researched in [4]. The tunable convergence heuristic, $N$, was set at 10 for all tests. The value of $N$ was chosen manually by experimentation.

### A. Cluster Size and Convergence Time

Fig. 5 depicts the convergence time results for 139 motes using 10, 14, 19, 24, 28, 33 and 40 clusters. As can be seen, the convergence time slowly increases as the number of clusters increases until 24 clusters, and then it begins to decrease. At first glance this may seen counter-intuitive, as one would expect convergence time to continue to increase as the number of CHs running the clustered binary consensus algorithm increases. However, the decrease after 24 clusters is caused by the fact that the network starts to become more dense, which causes binary consensus to operate more efficiently because the number of neighboring CHs seen by each CH increases.

### B. Clustered vs. Non-Clustered Binary Consensus

In order to compare clustered binary consensus with standard binary consensus, we ran experiments using our original implementation on the testbed. Fig. 4 shows these results. The

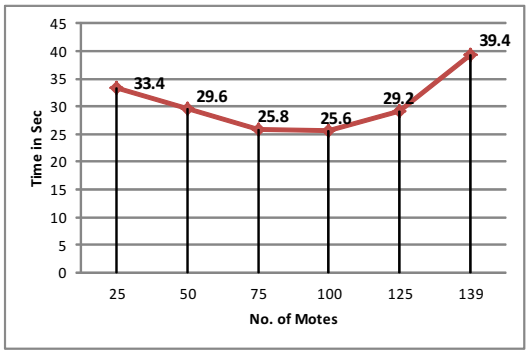


Fig. 4: Convergence time without clustering

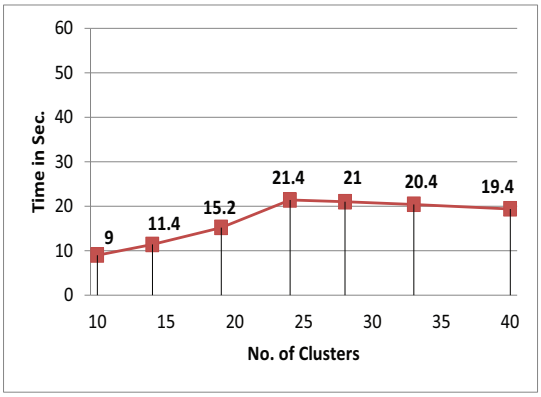


Fig. 5: Convergence time for 139 motes using clustering

effects of network density can be seen in these results as well (notice the distinct U-shape of the graph).

From the graph, we can see that the network converges faster as the number of motes increases from 25 to 100. This is due to the fact that when we have 25 motes distributed randomly on three floors, we get a sparse topology with small number of connections between the motes. Thus the convergence time increases. When the number of motes increases, the topology gets denser and converges more quickly. This is because in dense networks, the number of links between the motes increases so each mote will have more neighbors and the distance between the motes sitting on opposite sides of the network becomes less. This means that fewer state exchanges are required for the states of motes on opposite side of the network to be mixed, which results in faster convergence. After 100 motes, the density continues to increase but moves beyond the optimal density, so the convergence time is primarily influenced but the number of motes [4].

Comparing the results of convergence time when using clustering in Fig. 5 and our previous results in Fig. 4, we can see that the network converges faster when using clustering. For example, when we tested binary consensus with 139 motes without using clustering the convergence time reached 39.4 seconds. However, when we tested the algorithm using clustering the convergence time, in the worst case, was 21.4 seconds. This is due to the fact that when using clustering, the number of packets sent in the network is much less than the number of packets sent without clustering. With clustering, the members inside the cluster will only send their states to the CH, and the CH will take the role of computing the new states and communicating with other CHs. This reduces total packets sent as well as the convergence time.

### C. Packets Sent

To test the potential energy savings of clustered binary consensus, the number of packets required for convergence was measured for both the regular and clustered versions. In [21] it is shown that the energy used to transmit 1 bit is the same as used to execute 3000 lines of instructions. As such, we assume that the number of packets sent in the network can be effectively used to approximate the energy usage.

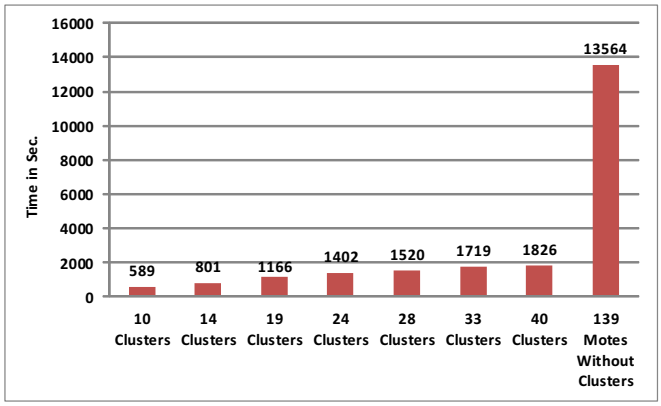


Fig. 6: Total number of packets sent to reach convergence

Fig. 6 shows the numbers of packets transmitted within the network when implementing our algorithm on 139 motes using 10, 14, 19, 24, 18, 33, and 40 clusters and when implementing it on 139 motes without clusters. Our implementation using clusters shows a noticeable enhancement in terms of energy savings. For example, the total packets sent when using 14 clusters for 139 motes is 801 packets. In contrast, the total packets sent in the case of 139 motes without clustering is 13564 packets. This is a reduction of 94%. This results in a huge power savings that increases the lifetime of the network. This savings is due to the fact that most of the motes only send their states at the beginning to the CH and then go to sleep. After that, only the CHs transmit or receive any packets.

The advantage of using a clustering schema with distributed algorithms will be most seen when working with large networks of motes. Without clustering, distributed consensus on large networks will require a significant number of packets to be sent and will drastically increase the convergence time. With clustering, however, normal motes send their states to CHs only, who are responsible for communicating with the other CHs, significantly reducing the number of packets sent. In this way, clustered consensus allows for more efficient use of large networks than non-clustered variants.

## VII. Conclusion

In this work, we have designed and implemented a clustered binary consensus algorithm for wireless sensor networks. We have tested our algorithm using 10, 14, 19, 24, 18, 33 and 40 clusters on 139 TelosB motes. By using a clustering approach we were able to reduce both the power consumed and the time to convergence when compared to a non-clustering binary consensus algorithm.

Future work in this area should include analyzing the algorithm assuming the presence of an attacker and applying the concept of clustering to other consensus algorithms such as the multi-valued and average consensus algorithms,

## 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. We would like to thank the School of Computing at the National University of Singapore for providing the INDRIYA testbed.

## References

[1] 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.

[2] 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.

[3] A. Boulis, "Castalia: revealing pitfalls in designing distributed algorithms in wsn," in *Proceedings of the 5th international conference on Embedded networked sensor systems*. ACM, 2007, pp. 407–408.

[4] N. Al-Nakhala, R. Riley, and T. Elfouly, "Binary consensus in sensor motes," in *9th IEEE International Wireless Communications and Mobile Computing Conference (IWCMC 2013),Cagliari, Italy*. IEEE, 2013.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] L. Chen, G. Carpenter, S. Greenberg, J. Frolik, and X. Wang, "An implementation of decentralized consensus building in sensor networks," *Sensors Journal, IEEE*, vol. 11, no. 3, pp. 667–675, 2011.

[13] S. Barbarossa and G. Scutari, "Bio-inspired sensor network design," *Signal Processing Magazine, IEEE*, vol. 24, no. 3, pp. 26–35, 2007.

[14] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *Systems & Control Letters*, vol. 53, no. 1, pp. 65–78, 2004.

[15] F. Shebli, I. Dayoub, A. M'foubat, A. Rivenq, and J. Rouvaen, "Minimizing energy consumption within wireless sensors networks using optimal transmission range between nodes," in *Signal Processing and Communications, 2007. ICSPC 2007. IEEE International Conference on*. IEEE, 2007, pp. 105–108.

[16] R. Singh, H. Singh, and R. Kaler, "An adaptive energy saving and reliable routing protocol for limited power sensor networks," in *Advances in Computer Engineering (ACE), 2010 International Conference on*. IEEE, 2010, pp. 79–85.

[17] T. Watteyne, A. Bachir, M. Dohler, D. Barthel, and I. Auge-Blum, "1-hopmac: An energy-efficient mac protocol for avoiding 1-hop neighborhood knowledge," in *Sensor and Ad Hoc Communications and Networks, 2006. SECON'06. 2006 3rd Annual IEEE Communications Society on*, vol. 2. IEEE, 2006, pp. 639–644.

[18] M. Handy, M. Haase, and D. Timmermann, "Low energy adaptive clustering hierarchy with deterministic cluster-head selection," in *Mobile and Wireless Communications Network, 2002. 4th International Workshop on*. IEEE, 2002, pp. 368–372.

[19] O. Younis and S. Fahmy, "Heed: a hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks," *Mobile Computing, IEEE Transactions on*, vol. 3, no. 4, pp. 366–379, 2004.

[20] M. Doddavenkatappa, M. C. Chan, and 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.

[21] L. Yong-Min, W. Shu-Ci, and N. Xiao-Hong, "The architecture and characteristics of wireless sensor network," in *Computer Technology and Development, 2009. ICCTD'09. International Conference on*, vol. 1. IEEE, 2009, pp. 561–565.