Polite Broadcast Gossip for IOT Configuration Management

Abstract

In this paper we present a protocol which can be used to form the basis of a consensus protocol for an Internet of Things (IOT) configuration management system. We motivate this discussion by focusing on a large and definitive class of IOT systems, Wireless Sensor Networks (WSN) and some important applications. We show that the current approach to consensus that underpin commonly used configuration management systems is inappropriate for IOT/WSN systems due to high communication overheads. We present a polite broadcast gossip dissemination algorithm which focuses on using a minimal amount of communication to update the configuration of a network of sensor nodes. We present analysis that the politeness of the algorithm does not inhibit its ability to function. The message savings of the algorithm is evaluated in simulation. We present test-bed results which show that our algorithm can disseminate metadata with roughly half of the communication overhead of a dissemination mechanism based on the one used by the IETF proposed standard Routing Protocol for Low Power and Lossy Networks (RPL).

I. Introduction

The Internet of Things (IOT) represents a class of distributed computer system that can be embedded in an environment of interest. They can be fit with sensors, and actuators to take the power of computing off of the desktop and into the world around us.

If these systems are battery powered, then energy is constrained, and energy efficiency is a major design consideration. Sensors can be a major consumer of energy, if they are not low-powered MEMS devices. The devices communicate sensed data using radio networking. Radio communication is expensive from the point of view of energy consumption, so communication must be kept to a minimum to prolong the lifetime of the system.

Configuration management tools are a core part of the operation of any large scale IT system [1]. Configuration management tools are those that enable us to specify a new configuration of some parameter on all of the hosts/nodes in a large scale distributed system. This task is performed remotely and automatically, without having to send a technician to visit every node with a laptop and a programming board. It prevents hours of tedious, repetitive, error prone human labour. This form of configuration management is now considered mandatory in large enterprise IT deployments, such as the desktop computers or servers in banks or universities. It is arguably more important in IOT/WSN class systems where there may be large numbers of sensors and they may be in difficult to access locations.

Ensuring that all IOT/WSN sensor nodes send their data to the correct base station is an example of important configuration information. Network routing protocols like the IETF proposed standard Routing Protocol for Low Power and Lossy Networks (RPL) [2] use a dissemination mechanism based on Trickle [3] to disseminate route and base station configuration. As important as the configuration of this routing information is, there is no mention in the literature as to how consensus is assured on this value.

At the heart of distributed system configuration management is the consensus problem [4]. The consensus problem is the problem of maintaining the same state on a group of loosely coupled, distinct entities. Addressing this problem provides a way to ensure that the configuration of each node is the same. This could be the same version of a configuration file, or the same number of seconds until a sample is taken from a specifically chosen sensor.

Two motivating IOT/WSN applications that will require configuration management are precision agriculture and smart water networks.

Precision agriculture [5] [6] is the use of wireless sensor networks to monitor the production of food. The sensors used in precision agriculture need to be configured to be able to deliver time correlated samples from the correct sensors. Configuration such as moisture sampling rate may need to be increased in a drought or if disease like root rot is discovered in one part of a field to try and predict or prevent its spread.

Adaptive sensing for smart water networks [7] allows the operators of large, urban water networks to monitor water pipes and pumps and manage the quality of the functioning of the service. The sensors for smart water networks will often be in difficult to reach locations, such as under cast iron manhole covers on busy roads, or on large pumps in large, concrete pumping stations. The sensor nodes will have to rely on local neighbour nodes to disseminate new information to them. Configuration updates to these urban water systems are critical, such as a different sampling rate, or a different processing or control algorithm, or a security update.

This paper looks at current solutions in production for distributed system configuration management, and we note that none is suitable for IOT/WSN class systems. We propose a protocol called FiGo that can be used to disseminate a new metadata to IOT/WSN systems which taking into account the requirement of communication overheads. Finally, we provide a reference implementation and some evaluation of the implementation in simulation and on hardware.

II. Consensus and Configuration Management Current Approaches.

In this paper we address the unsuitability of the current configuration management systems and their underlying consensus algorithms on IOT class devices, and suggest a suitable approach to solve the problem.

The consensus problem is the problem of assuring that all of the nodes have the same local state at a certain point in time, over a certain period of time. Assume a model of n independent identical processes, or nodes. All nodes are composed of the same state variables whose value can be either zero or one. Each node can communicate with all or a subset of the system, called its neighbours. A node can exchange information about its state with its neighbours, and change its local state based on its current state and the states of its neighbours. The consensus problem is to design a protocol so that all processes eventually have the same local state, either all ones, or all zeros.

WSN management systems like DISON [8], BOSS [9], and Moteview [10] solve the consensus problem by using a single central controller. Centralised architectures risk failure if the central controller fails. These systems can suffer from the split-brain problem if employed with redundant central controllers.

Split-brain is an example of the type of problem that can occur if multiple, redundant controllers are used for configuration management, without the use of a consensus algorithm. Assume two controllers, a master and a second which takes over if the master fails. If the master fails, or the network becomes partitioned, the second will become a master. If the original master then returns, the system will have two masters, and the system state will become inconsistent. A consensus protocol serves to prevent this problem.

Systems that do not assume the existence of a single controller address the consensus problem by using one of the Paxos family of algorithms such as Fast-Paxos [11], Egalitarian-Paxos [12]), Zab [13], or Raft [14]. These algorithm are grouped into a family because they all use an approach based on quorums.

There are several problems with the use of Paxos/quorum based consensus algorithms in battery powered IOT class systems. The first is that the message overhead is high. It requires multiple phases, a leader election, a proposal phase and an acceptance phase. Each phase requires its own set of messages and responses. Another related problem with the use of Paxos/quorum based algorithms is that they assume a stable network population. If the quorum nodes can fail, then the algorithm will need an extension to add or remove quorum nodes. This results in more communication overhead.

The RPL [2] routing protocol also has to disseminate configuration information about the id of the base station, and could disseminate information about new objective functions. This information is disseminated in DIO messages using the Trickle algorithm [3]. Distributed WSN management systems, like the Sensor Network Management System (SNMS) [15] and Impala [16], use a purely distributed system, and so avoid the above mentioned problems with centralised WSN control. SNMS is built on top of a broadcast gossip protocol called Drip [15] which uses the same Trickle algorithm as RPL.

A seminal work by Fischer, Lynch and Paterson [17] shows that consensus cannot be guaranteed in an asynchronous network. FiGo focuses on providing both a synchronisation mechanism and a metadata dissemination mechanism with low communication overheads, which can be used as the basis of a configuration service suitable for IOT systems.

FiGo is a polite broadcast gossip protocol. Convergence results for broadcast gossip based consensus systems have been demonstrated in several works [18],[19]. On a test-bed, we compare FiGo to a protocol that uses the same dissemination algorithm as RPL, Drip, to show that we can disseminate data with a lower communication overhead. Dissemination in RPL and Drip are both based in Trickle which is not a synchronisation algorithm, and so is not suitable as a consensus protocol.

III. FiGo, the Firefly-Gossip Algorithm

FiGo (Firefly-Gossip) is a proof of concept for the evaluation of the use of a polite broadcast gossip-based protocol to provide consensus in a resource constrained environment. We chose this approach because radio communication is inherently broadcast, and the polite message suppression is a easy way to reduce the communication overheads of broadcast and prevent medium congestion. The gossip based approach is a well established dissemination mechanism.

FiGo is made of a synchronisation algorithm coupled to an information dissemination algorithm. We present the algorithm in 1, then we discuss some of its properties.

Algorithm 1 is made of two procedures. The first is periodic. Every node sets a random timer to fire between during its synchronisation period. When the timer fires, the node sends a synchronisation message. The second procedure is called when a node receives a neighbour synchronisation message. The receiver parses the message as two pieces of information. The node uses the synchronisation information to synchronise to, and the dissemination information to determine its new configuration.

FiGo uses polite message suppression. Once a node receives a message from a neighbour, it cancels its own next broadcast, unless the neighbour's configuration information is out of date. The use of polite message suppression in FiGo means that a node will only receive the synchronisation message of one neighbour. Synchronisation data is received during the whole period but only used if it arrives in the second half of the receiver's period, line 19 of procedure RECEIVE. The first half of the receivers period is referred to as a refractory period, and synchronisation information received during the refractory period is discarded. If a node receives a synchronisation message after the refractory period, then it replaces its next firing time with that of the time of arrival of the synchronisation message. This can be seen in line 12 of procedure TIMER_FIRE.

For synchronisation, the two procedures share two variables, the normal period, called default_period, and the synch_offset. The synchronisation offset is normally set to zero, and every period the timer is given the default period. If a synchronisation message arrives after the refractory period, the node will set its next firing time to be the message arrival time. We assume that the synchronisation messages are sent at the firing time of the sending node, and that the delay of sending and receiving can be ignored.

The disseminated configuration data is parsed and used during the entire length of the synchronisation period. If the data received is the same as the local data, then no further action is taken with regards to dissemination information this period. If the information received is out-of-date, then the local node sends the newer data. The newer data message is marked so as to not be used as a synchronisation message. In this way the node which previously had out-of-date data would now be up to date. If the received data is newer than the local data, the local data is replaced with the newer data, and the receiving node is now up to date.

For dissemination, the two procedures share three variables. One, same_count, is only ever incremented in procedure RECEIVE, and set to zero every period when the timer fires in procedure TIMER_FIRE. The variable local_state is only updated in procedure RECEIVE. The final variable same_threshold is never changed by either procedure.

In the case of simple configuration data like synchronisation period length, the data, along with some associated metadata (like the age of the data), can be parsed and used directly. In the case of the dissemination of larger data, the extra information included in the synchronisation message can be just metadata which references other data that then needs to be communicated separately. The data used by FiGo (the event frequency) is small enough to be included in the sync message along with the metadata.

FiGo is robust to node failures, assuming that node failure is either the complete failure of a node (it ceases to function), or partial failure (it ceases to function, and then returns, functioning correctly). When the node fails, some other node will broadcast in its place. If the node returns, it synchronises to the other nodes, and discovers if its configuration data is the latest. Network partitioning is the worse situation that can happen with the loss of a node. If the network becomes partitioned, then a part of the network will not receive metadata updates. If the network becomes rejoined, then the newest metadata will replace the older, bringing the previously partitioned nodes back up to date.

IV. FiGo Properties

In this section we look at how FiGo handles the basic safety properties required by a consensus system. These properties are from [20], and are: termination, every node eventually decides a value; integrity, every

Algorithm 1 The TIMER loop, occurs periodically. The RECEIVE logic occurs for a gossip message.

```
local_state = 0
same_count = 0
same_threshold = $n$
synch_offset = 0
default_period = $t$
procedure TIMER_FIRE
    if same_count $<$ same_threshold then
        TRANSMIT(local_state)
    end if
    same_count = 0
    if synch_offset $>$ 0 then
        SET_TIMER(synch_offset)
    else if then
        SET_TIMER(default_period)
    end if
    synch_offset = 0
end procedure
procedure RECEIVE(received_state)
    if NOW $>$ default_period/2 then
        synch_offset = NOW
    end if
    if received_state $>$ local_state then
        local_state = received_state
    else if received_state $<$ local_state then
        TRANSMIT(local_state)
    else if received_state == local_state then
        same_count = same_count + 1
    end if
end procedure
```

node only decides one value; validity, a node only decides a value that has been accepted, and is therefore valid; and finally, agreement, a stronger form of termination, states that every node eventually decides on the same value. We show that these safety properties hold given our assumptions.

To show that FiGo will terminate and can provide agreement, integrity, and validity, we first establish that with our assumptions each node will eventually hear a broadcast of newer data.

Assume a network G, made up of nodes and communication links between them (N, L). There are $|N|$ nodes, referred to as n_i where a sub script i is the unique id of the node. Different subscripts of n denote different nodes. All nodes n_i are in N, $n_i \in N$. A link $l_{i,j} \in L$ exists between two nodes n_i, n_j if both nodes can communicate with each other. The set of all the neighbour nodes that an individual node can communicate with in one hop, that is, shares a link with, is referred to its neighbourhood. We assume bi-directional communication, and no message loss. We also assume that a monotonic ordering, like a time-stamp, can be maintained across all of the nodes in the system. This assumption is safe given that FiGo is at heart a synchronisation protocol, and is the way that FiGo is able to provide consensus.

Time is divided into timeslots, each node fires at a random time in the timeslot. The random time comes from a uniform distribution. Because of polite suppression, all nodes will either hear a broadcast message or broadcast themselves. The question now becomes, is it possible that new information could enter the network, and the nodes fire and suppress in such a way that new information entering into the network would not spread?

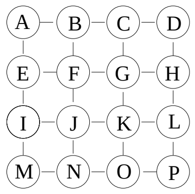

Fig. 1. Four node by four node grid topology.

Let us consider the network of nodes arranged in figure 1. Each node is identified by a letter from A to P. Nodes can only communicate with neighbours one hop away, in the four cardinal directions. New information enters the network at node P. The first round after the new information enters, it will propagate to nodes O and L with a probability of one.

In the next timeslot, there is a probability that the message will not propagate. In order for this to occur, node P must broadcast, silencing O and L, as they all have the same, newer information. In the same timeslot, All of the neighbours of O and L, that is N, K, and H, must be silenced by a set of their neighbours, M, J, G, or D.

We simulated the above model to find the probability that P would fire, and N, K, and L would be silenced by some combination of their neighbours M, J, G, or D. This condition is given by condition 1.

$$P \wedge \neg\{N, K, L\} \quad (1)$$

Each timeslot produced a configuration of all of the nodes that fired, and all of the nodes that were silenced. The simulation was run for ten million timeslots, and the number of occurrences of each configuration was recorded. The number of occurrences of condition 1 were summed together and divided by the total number of timeslots, to give the probability of a configuration that would stop information from flowing from nodes L, O, and P.

In the four by four topology with connectivity in the cardinal directions there were 90 possible combinations of node firing orders per timeslot. Only two met condition 1. The probabilities of the occurrence of condition 1 was %6. This is for one time slot. The probability of a disconnected combination two timeslots in a row is %0.36. As the number of timeslots increases, the probability that configurations occur that prevent information from flowing diminishes.

We modelled the same topology with a five by five topology and connections in the cardinal directions, and observed a %5.8 probability of a disconnected configuration.

We also changed the connectivity of the nodes, by specifying that each node could communicate with all adjacent one hop neighbours. The probability for an unconnected configuration in a four by four network was %4.2 for a timeslot, and for a five node network was %2.

The probability that new information does not spread exists, but it is very low, and vanishes as the number of timeslots increases or the connectivity of each node increases. This suggests that sparse topologies may cause problems for polite gossip. We now continue with the assumption that new information will eventually spread to all of the nodes in a dense network.

A. Termination, Agreement, Integrity and Validity

Termination simply states that all nodes will eventually decide a value. For FiGo this is simply an extension of our assumption that all nodes will eventually receive new state. We can see in line 23 of algorithm 1 that as long as newer data is received, the node will update its value. If no data is received, the current value remains. Either way, the node has decided on only one value.

For agreement, we show that the following condition will hold:

$$\forall n(n_i, n_j \in N | n_i.state = n_j.state) \tag{2}$$

We will address this property by looking for conditions under which this property would not hold, and show that they contradict our assumptions.

In order for an up to date state not to be accepted by a node, it must either, have a newer state, or never hear a broadcast. If a correct node had a newer value, then by the algorithm 1 line 8, it would broadcast its newer version immediately, and inform its neighbours of a newer value. If a correct node never heard the newer value then we can assume that it is either disconnected from the network, or is in a part of the network disconnected from the part of the network into which the update was received. We assume that the network is fully connected, and so this is a contradiction of our assumptions.

Therefore, if a newer state value is in the network, and the network is connected, then it will propagate to all members of the system, and all of the members will replace their state with the new one.

This also covers integrity which states that each node will only have one state, expressed by the following condition:

$$\neg \exists n : |n.value| \geq 2 \tag{3}$$

This property follows from the actions in algorithm 1 line 23. The only time that the value is updated is when it receives a new value from a neighbour. The only action that it performs is to replace its value with the new one. So, each node will only have one state, and that will be the newest one.

Validity states that only a value that has been proposed can be accepted. FiGo is different from Paxos and its variants. There is no separate group of acceptors, no use of the quorum concept, so any value that is proposed, and is newer than the current value of the node will be accepted. This property can be shown by line 23 of the algorithm 1. This is the only place where the local state can be updated, and it is updated only if a newer state value has been received, and it is updated to the received value.

We have shown that, given our assumptions, the polite message suppression aspect of FiGo does not prevent message dissemination. By showing that dissemination will occur, we show that the safety properties of termination, integrity, validity, and agreement all hold for FiGo. Therefore FiGo can function as a consensus protocol.

V. FiGo Evaluation

The evaluation of FiGo is concerned with two things. The first is the ability to disseminate data and synchronise the network. This relates to the fundamental assumption we made while examining the properties of FiGo. This set of experiments was performed in simulation. The second thing is the ability to perform this function cheaply in terms of communication costs measured by the number of messages used. This was evaluated on a WSN test-bed against a form of the same protocol used by RPL to disseminate network routing information, Drip.

A. FiGo Simulation Evaluation.

In this section we present simulation results evaluating the performance of FiGo. The simulation was written in NesC [21] and run with the TOSSIM simulator [22]. We use the standard TinyOS Active Message interface and MAC layer. The simulations output two metrics: The time to disseminate a new event period and synchronise to it for all of the nodes, and the percentage of messages used to disseminate and synchronise. The new configuration data which is disseminated is new event periods.

In the simulation, all nodes started with the same firing period(frequency), but were all out of phase with each other. First we measured their time to synchronise. Once the nodes were synchronised, we waited for three minutes, and then disseminated a new firing period. We then measured the time from the beginning of the dissemination to the point at which all of the nodes were synchronised with the same period and phase. The results shown were the times to synchronise from random start, and from each of the period changes.

We used a grid topology with a fixed neighbour population (four in this case) for the inner nodes, three neighbours for the nodes on the edges, and two neighbours for the nodes on the corners. The Grid topology allowed us to test multi-hop communication. We used populations of 9, 16, 25, 36, 49, 64, 81, and 100 nodes each one having an $(\sqrt{n} - 2) + \sqrt{n})$ maximum hop count (where n is the node population). The parameters for the simulation are shown in table I.

We varied the message suppression in three ways to see its impact on the time to disseminate and synchronise after the frequency changes, and percentage of messages used per firing event. The first suppression policy was random with a probability of 20%. This was to compare against a purely random policy. The next two policies suppressed a node from sending if it received messages with the same time and data payload. The number of messages needed to suppress communication were two messages and one message.

B. Simulation Results

Figures 2, 3, and 4 show the time to disseminate and synchronise a multi-hop network with a grid topology for each of the three sync periods. They compare the sync times of FiGo using random message suppression against fixed message suppression of thresholds of 2 and 1 messages. In all cases the times to sync are almost identical. Both fixed thresholds take marginally longer to sync in all cases, yet there is a marginal reduction in messages broadcast during the same periods. Message suppression after one message uses 90% less messages than the use of no suppression policy and half of the messages of the random suppression policy. Observing that the time to sync is on average 10 seconds more for each period presented, we can optimise on either 'time to sync' or 'message overhead' depending on application requirements.

Another point worth discussing is that the number of cycles needed to synchronise from a random start in Figures 2, 3 and 4 are nearly identical. By dividing the time to sync by the period (in seconds), Figures 2 and 3 are the same, and Figure 4 uses slightly less cycles. This is not unusual, because before the new period is introduced into the network, the nodes are synchronised. The rate of dissemination is therefore very quick, and the nodes are able to re-sync themselves rapidly. Therefore, we attribute much of the time to dissemination time. This result also shows the stability of the synchronisation part of the protocol with respect to disturbances such as, part of the network changing its synchronisation period.

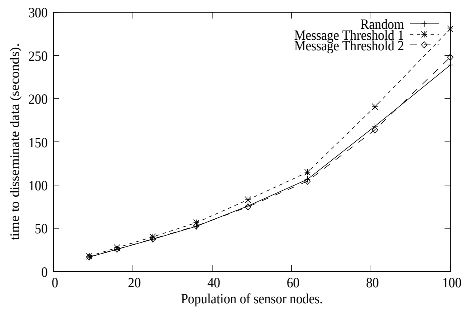

Fig. 2. Comparison of the time to disseminate new data with an event period of one second.

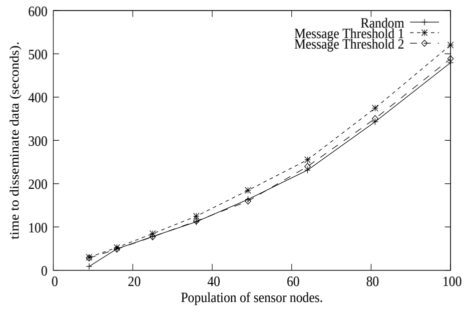

Fig. 3. Comparison of the time to disseminate new data with an event period of two seconds.

The simulation showed that the savings of messages used for each time period were invariant with the

Parameter	Simulation	Testbed
packet size	128 Bytes	128 Bytes
clock period	1sec	1sec
TX power	-	0dB
clock drift range	0 - .01	-
node populations	[9, 16, 25, 36, 49, 64, 81, 100]	117
trials	1000	10
run time/trial	60min	4hrs
synch window	100ms	100ms
periods used	[500ms,1000ms, 2000ms]	1024ms
radio noise model	Meyer Heavy	-
most neighbours	4	20
max hops	18	8hops

TABLE I. Parameters for the simulation experiments performed.

Protocol	Avg. Time	Std. Deviation	Avg. Messages Used
FiGo	25.79sec	8.37sec	416,668
Drip	21.07sec	9.24sec	805,968

TABLE II. A comparison of dissemination times in seconds on the Indriya Testbed.

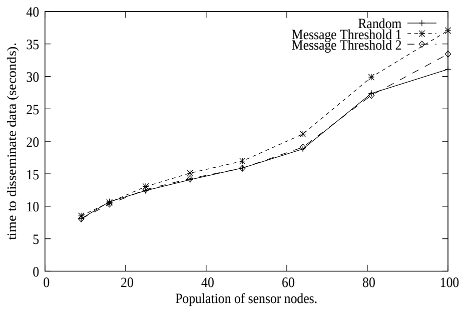

Fig. 4. Comparison of the time to disseminate new data with an event period of half a second.

population of nodes. If all of the nodes sent synchronisation messages the messages used would be 100%. Fixed message suppression with a threshold of two and random suppression with a message 20% of the time both used 80% less messages than no suppression. A message threshold of one message used 90% less communication compared to no suppression. The reduction in communication is offset by a slightly longer time to disseminate and synchronise as seen in Figures 2, 3 and 4.

All of the simulation results show that FiGo lends itself well to the combination of configuration data with synchronisation data without impeding the functioning of any individual functionality. Synchronisation can be maintained and new configuration data can be disseminated, creating a basis of a consensus protocol.

C. FiGo Experiments on a Remote Testbed.

The next set of experiments were run on the Indriya [23] wireless sensor network testbed. The experimental parameters are given in table I. All 117 Telosb nodes were programmed with the same code which consisted of a temperature sensing application which took synchronised samples every second and sent them to a single base-station every second. Every thirty seconds, a dissemination root would disseminate a command to change the colour of the LED. The LED was flashed every second to indicate that a temperature sample had been taken. We had two versions of the application. One used the Drip [15] dissemination protocol and represented the approach used by the RPL DIO message dissemination process. The other version used FiGo. We compared the number of control messages used by each application, and the time to disseminate new information.

The temperature sensing application was run for four hours, five times. Both applications used CTP for data collection. FiGo sent a synchronisation pulse every second, and Drip used a Trickle Timer with a maximum period of one second. FTSP sent a sync pulse once every three seconds. These sending rates show the performance of FiGo under high load conditions.

D. FiGo Testbed Results.

Our first set of test-bed experiments were a comparison of the time in seconds for each protocol to disseminate a new LED colour. We compared Drip against FiGo. The results are presented in table II. These results show that on average Drip disseminates code 18% faster than FiGo. This result is not surprising because both Drip and RPL use a Trickle timer. A Trickle timer sends data advertisements at a variable rate. When a node detects (or receives) new data, then the Trickle timer reduces the rate at which advertisements are sent, thereby increasing the speed at which nodes discover that they need new data. FiGo always maintains the same advertisement rate because it is coupled to the synchronisation mechanism.

The increased advertisement rate of Drip makes it disseminate new data 18% faster. This increase in speed comes at a cost. In the final column of Table II We show that the increased speed of dissemination required 48% more messages. Added to this is the fact that messages FiGo is using are also providing synchronisation information at the same time. This result shows that FiGo uses roughly half of the messages to perform the same task.

VI. Conclusion

This paper has taken a look at current approaches to IOT/WSN management. We have noticed that the distributed approaches have not give proper consideration to the consensus problem. In order to address this issue we presented a polite broadcast gossip protocol called FiGo as a way to provide a consensus protocol for IOT systems. We presented analysis to show that the polite aspect of the protocol will not prevent it from disseminating new data. Testbed evaluation showed that FiGo can synchronise the network and disseminate configuration information whilst reducing by approximately 50% the messages required by established approaches.

References

[1] T. Delaet, W. Joosen, and B. Van Brabant, "A survey of system configuration tools." in *LISA*, vol. 10, 2010, pp. 1–8.

[2] T. Winter, "Rpl: Ipv6 routing protocol for low-power and lossy networks," 2012.

[3] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko, "Rfc6206: The trickle algorithm," *Internet Engineering Task Force (IETF) Request For Comments, http://ietf. org/rfc/rfc6206.txt*, 2011.

[4] M. J. Fischer, "The consensus problem in unreliable distributed systems (a brief survey)," in *Foundations of Computation Theory*. Springer, 1983, pp. 127–140.

[5] A. Z. Abbasi, N. Islam, Z. A. Shaikh *et al.*, "A review of wireless sensors and networks' applications in agriculture," *Computer Standards & Interfaces*, vol. 36, no. 2, pp. 263–270, 2014.

[6] T. Ojha, S. Misra, and N. S. Raghuwanshi, "Wireless sensor networks for agriculture: The state-of-the-art in practice and future challenges," *Computers and Electronics in Agriculture*, vol. 118, pp. 66–84, 2015.

[7] S. Kartakis, E. Abraham, and J. A. McCann, "Waterbox: A testbed for monitoring and controlling smart water networks," in *Proceedings of the 1st ACM International Workshop on Cyber-Physical Systems for Smart Water Networks*. ACM, 2015, p. 8.

[8] T. M. Cao, B. Bellata, and M. Oliver, "Design of a generic management system for wireless sensor networks," *Ad Hoc Networks*, vol. 20, pp. 16–35, 2014.

[9] H. Song, D. Kim, K. Lee, and J. Sung, "Upnp-based sensor network management architecture," in *Proc. International Conference on Mobile Computing and Ubiquitous Networking*, 2005.

[10] M. Turon, "Mote-view: a sensor network monitoring and management tool," in *Proceedings of the 2nd IEEE workshop on Embedded Networked Sensors*. IEEE Computer Society, 2005, pp. 11–17.

[11] L. Lamport, "Fast paxos," *Distributed Computing*, vol. 19, no. 2, pp. 79–103, 2006.

[12] I. Moraru, D. G. Andersen, and M. Kaminsky, "There is more consensus in egalitarian parliaments," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 2013, pp. 358–372.

[13] F. P. Junqueira, B. C. Reed, and M. Serafini, "Zab: High-performance broadcast for primary-backup systems," in *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*. IEEE, 2011, pp. 245–256.

[14] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, 2014, pp. 305–319.

[15] J. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pp. 81–94, 2004.

[16] T. Liu, C. Sadler, P. Zhang, and M. Martonosi, "Implementing software on resource-constrainted mobile sensors: Experiences with zebranet and impala," *ACM MobiSYS*, 2004.

[17] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM (JACM)*, vol. 32, no. 2, pp. 374–382, 1985.

[18] A. Khosravi and Y. S. Kavian, "Broadcast gossip ratio consensus: Asynchronous distributed averaging in strongly connected networks," *IEEE Transactions on Signal Processing*, vol. 65, no. 1, pp. 119–129, 2017.

[19] H. Wang, X. Liao, Z. Wang, T. Huang, and G. Chen, "Distributed parameter estimation in unreliable sensor networks via broadcast gossip algorithms," *Neural Networks*, vol. 73, pp. 1–9, 2016.

[20] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *Journal of the ACM (JACM)*, vol. 43, no. 2, pp. 225–267, 1996.

[21] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesC language: A holistic approach to networked embedded systems," *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, pp. 1–11, 2003.

[22] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: accurate and scalable simulation of entire tinyOS applications," *Proceedings of the 1st international conference on Embedded networked sensor systems*, pp. 126–137, 2003.

[23] M. Doddavenkatappa, M. Chan, and A. Ananda, "Indriya: A low-cost, 3d wireless sensor network testbed," *Testbeds and Research Infrastructure. Development of Networks and Communities*, pp. 302–316, 2012.