Fragmentation-based Distributed Control System for Software Defined Wireless Sensor Networks

Hlabishi I. Kobo, Adnan M. Abu-Mahfouz and Gerhard P. Hancke

***Abstract*—Software Defined Wireless Sensor Networks is a new and emerging network paradigm that seeks to address the impending issues in Wireless sensor networks. It is formed by applying Software Defined Networking to wireless sensor networks whose basic tenet is the centralisation of control intelligence of the network. The centralisation of the controller rouses many challenges such as security, reliability, scalability and performance. A distributed control system is proposed in this paper to address issues arising from and pertaining to the centralised controller. Fragmentation is proposed as a method of distribution, which entails a two level control structure consisting of local controllers closer to the infrastructure elements and a global controller which has a global view of the entire network. Distributed controller system brings several advantages and the experiments carried out shows that it performs better than a central controller. Furthermore the results also show that fragmentation improves the performance and thus have a potential to have major impact in the IoT.**

***Index Terms*—Industrial Wireless Networks, Industrial Internet of Things (IIoT), Software Defined Wireless Sensor Networks (SDWSN)**

I. Introduction

THE Internet of Things (IoT) is at the centre of the future internet. The IoT framework is an interconnection of many devices, systems, and applications to the internet. Accordingly to Cisco, an estimated 50 billion devices will be connected to the internet by the year 2020. The IoT paradigm is envisaged to permeate into the industrial manufacturing and production, leading to Industrial Internet of Things (IIoT) [1]. The IIoT is envisioned to inspire great economic growth and rapid production in various industrial production systems. This digital oriented industrialisation is termed the "the fourth industrial revolution or 4IR".

The 4IR is described as the fourth disruptive and major industrialisation trend; an epoch marked with rapid growth and development as a result of automation and data technologies in various disciplines. Some of the major technologies behind this trend includes but no limited to IoT, IIoT, Artificial Intelligence, Virtual Realities, Cyber-Physical systems, Cloud, and Cognitive computing. Most of the devices and elements which will partake in these technologies especially IIoT, will be equipped with sensors and actuators; some wireless and some wired. The networking of these sensor nodes augment the scope and purpose of Wireless Sensor Networks (WSNs), whose involvement has always been confined to monitoring [2] in the main. These sensor nodes are small, inexpensive and intelligent due to the advancement of Micro Electrical Mechanical Systems (MEMS). The major challenges facing industrial systems include the management of the various systems with different proprietary protocols as well as their sensitivity to time delay, failure and security.

The Software Defined Networking (SDN) is a new emerging networking and computing paradigm earmarked as a potential resolve of most of the above mentioned challenges. SDN advocates for a common standardised protocol to avoid the challenge of vendor locking [3]. The SDN model separates the control and data forwarding on the networking elements; thus removes the control logic from the network devices and centralise it on a controller [4]. The adoption of SDN has gained traction in both the industry and the academia. Most of the 4IR systems are already applying SDN including IIoT and WSN such as in [5]–[7].

Software Defined Wireless Sensor Networks (SDWSN) is an emerging model formed by applying the SDN model in WSNs. The emergence of SDWSN as a pivot, in the stead of WSNs, for the highly anticipated and imminent IoT and IIoT paradigms has ignited much interest and research focus. WSNs are envisaged to play a vital role in IoT as a major building block [8]. However WSNs have always been riddled with challenges emanating from their inherent nature of resource constraints thereby hindering their progress, efficiency and applicability [9].

The application of SDN in WSNs is also receiving much attention especially on the basis of its imminent role in IoT [10]. SDWSN is regarded as a potential to overcome some of the challenges besetting WSNs while meeting the demands of IoT. Most of the energy intensive functions are moved from the sensor node to the controller. With SDWSN, the sensor nodes would be sheer devices with only the forwarding capabilities whereas the control intelligence is centralised.

A distributed control system in SDWSN seeks to address

Hlabishi I. Kobo and Adnan M. Abu-Mahfouz are with the Department of Electrical, Electronic and Computer Engineering, University of Pretoria and the Meraka Institute, Council for Scientific and Industrial Research (CSIR), South Africa (hkobo@csir.co.za and a.abumahfouz@ieee.org).

Gerhard P. Hancke is with the Department of Computer Science, City University of Hong Kong, Hong Kong, China and the Department of Electrical, Electronic and Computer Engineering, University of Pretoria, South Africa (ghancke@ieee.org).

the challenges of a centralised controller in order to achieve reliability, scalability, and efficient performance. There are different ways and forms of distribution mainly determined by the nature of the network or the data concerned. This paper presents an efficient distribution technique suitable for SDWSN control system. It takes into account all considerations pertaining to SDWSN challenges such as the inherent ills of low bandwidth, energy, memory and processing as well as the amount of data exchange or update expected especially from the IoT perspective. The proposed method uses the concept of fragmentation. Thus each cluster segment of the network has its own controller which is lean and very close to the infrastructure elements. There also exists a global controller which has a view of the whole network. This two level control architecture allows a faster response between the sensor nodes and the control.

Central to distributed systems is the consistency models, which determines the data convergence on the distributed participants (nodes). There are two major consistency models in Eventual consistency and Strong consistency. The former uses gossip protocols such as anti-entropy and rumour mongering whilst the later uses consensus algorithms such as RAFT [11] and, or Paxos [12]. The choice of the consistency model depends on the need of the application and the type of data. This paper investigates the applicability of these consistency models and their algorithms in the distributed control systems for Software Defined Wireless Sensor Networks (SDWSN). The contributions of this paper are:

1. To investigate the feasibility of distributed controllers for SDWSN.
2. To propose Fragmentation as a method of distribution to achieve efficient SDWSN control.
3. To propose alternative algorithms to Best effort and Anti-entropy algorithms to achieve a suitable consistency data model for Fragmentation.
4. To show through the evaluation that Fragmentation does bring efficiency to SDWSN control.

The rest of the paper is organised as follows: We highlight a brief overview of SDWSN in section II which includes controller and related work. Section III highlights the distributed control system for SDWSN with the proposed model. Gossip or epidemic protocols are discussed in section IV. Sections V and VI deals with the experimental setup and results respectively while section VII is a discussion. Section VIII concludes the paper.

II. Software Defined Wireless Sensor Networks

A. SDN and WSN

Software Defined Networking is an emergent computing and networking model which has brought convenient disruption in both the academia and the industry at large [4], [13], [14]. It is regarded as innovative, simple, evolutionary etc. SDN decouples the control plane from the data forwarding plane. The decoupling leaves network elements or devices as dump devices which only forwards packets. The control intelligence is centralised into a controller. SDN comprises of three layers or planes in data/infrastructure plane, control plane, and an application plane. The data plane is made of the sheer infrastructure devices which only understand instructions from the controller. The control plane hosts the controller. The application plane host various application that offer functionality and services to the network. The application and adoption of SDN to various computing and networking platforms is rapidly growing [15]. The fusion of SDN and WSN begets SDWSN which has a potential to resolve some of the WSN's inherent challenges. The SDWSN is envisioned to play a significant role in the IoT revolution.

B. Controller

A controller plays a very vital role in SDWSN. It is the central point of control where decisions are made. This SDN abstraction brings lots of benefits especially in management and configuration [16]. It also allows new policies and other changes to be implemented with ease. However this model is not immune to drawbacks. The critical shortcomings emanate from its centralisation of the control intelligence, thereby invoking issues such as reliability, security, performance, and scalability [17]. A central controller implies a central point of failure and also a potential target for adversaries [4], [18]–[20]. Also, as the whole network relies on it for functionality i.e. flow rule setup, potential performance degradation can be encountered as the network grows. Furthermore, SDWSN is envisioned to handle lots of data especially with the advent of IoT. Unlike enterprise networks, SDWSN will deal with lot of sensory data which are delay sensitive. Therefore a central controller for SDWSN will not be practical [15], thus the need for a distributed control system. Distributed controllers do exists for mainstream SDN enterprise networks but not so in the SDWSN space. Some of the popular distributed controllers in the SDN community are OpenDaylight [21], ONOS [22], Hyperflow [23], Kandoo [24], Elasticon [25].

C. Related work

SDN-WISE [26] is a comprehensive SDWSN framework based on state automata. The SDN-WISE software stack gives detailed description of the SDN sensor nodes, sink nodes and the emulated nodes. The stateful structure includes the flow table composition in the follow of Openflow. It also details the protocol architecture dealing with the packet handling and processing, and topology discovery techniques. The stack is adaptable to various SDN controllers and as such an adaptation with ONOS was implemented by the authors and tested for interoperability with the enterprise network [27].

The combination of SDN-WISE and ONOS is progressively in the right direction towards the realisation of an effective and efficient SDWSN for IoT. In [28] this solution was tested using multiple controllers in a distributed fashion. The authors of SDN-WISE also implemented a custom java controller to test this framework which used Dijkstra's algorithm. Dijkstra's algorithm finds the shortest paths between nodes in a connected graph. In [29], Collection Tree Protocol (CTP) is used to find a controller in a multiple controller solution called TinySDN. The controller is attached to the sink node through a serial connection; a hierarchical extension of this solution was implemented in [30].

III. Distributed SDWSN Control System

A. Background

Distributed systems found more prevalence in database theory. E Brewer profoundly stated at the 2000 Symposium of distributed computing that "*in any highly distributed data system there are three common desirable properties: consistency, availability, and partition tolerance. However, it is impossible for a system to provide all three properties at the same time.*" [31], [32]. This became known as the CAP theorem. Coronel and Morris [32] defines these three properties as follows: Consistency – all nodes should see the same data at the same time, which means that the replicas should be immediately updated. Availability – a request is always fulfilled by the system. Partition tolerance – the system continues to operate even in the event of a node failure.

In the context of this paper, Consistency would mean all fragments (local controllers) have the same network state all the times i.e. symmetric. Availability – all nodes available; Partition tolerance – network continues to function after a node failure. Another phenomena common in database systems is ACID (Atomicity, Consistency, Isolation, and Durability); regarded as the four properties of transactional database. ACID ensures that all transactions results in a consistent database state. Coronel and Morris notes that this is well suited for a centralised and small distributed database systems [32]. Otherwise latency becomes an issue as the system scales. They further state that it is for this reason that many systems sacrifice consistency and isolation for availability, leading to another phenomena BASE (Basically available, soft state, eventually consistent). In BASE data exchanges are not immediate but propagate slowly until all nodes are eventually consistent [31].

A distributed SDN control systems draw reference from the above, particularly the database systems. However the fundamental roles of a SDN controller and a database system are distinguishable. The primacy of a database is to store data and enable the CRUD (create, read, update, and delete) operation. The SDN controller on the other hand is an engine of the network; more pointedly to control the infrastructure devices by defining data propagation rules. Furthermore, SDWSN brings another dimension, different from the traditional SDN by putting the sensor nodes at the periphery of the network. This also tilts the paradigm of the controller's role. Our system follows the BASE as a consistency model thereby preferring availability over consistency.

This paper undertakes the eventually consistency model and customise it for SDWSN control. This process takes cognisance of the perpetual SDWSN challenges such as limited energy, memory, bandwidth etc. This method applies concurrency and parallelism on the local controller nodes. This effectively ensures that each local controller node independently controls its segment (cluster) of the network and work concurrently with the other local controller nodes. These nodes would operate simultaneously, thus guaranteeing parallelism. This would allow a faster response between the sensor nodes and the controller. Most of the data would be upstream and only infrequently would the controller send control instruction to the devices.

B. Fragmentation

We propose fragmentation for our distributed control solution. Fragmentation entails dedicating part of the control system to different segments (fragments) of the sensor network. In addition to fragmenting the control of the network, this could be extended to abstracting different sensor traits together e.g. temperature nodes. Fig. 1 depicts the distributed control system with fragmentation. This architecture is akin to Kandoo [24], which also has local controllers as well as a global controller overseeing the whole network. However, Kandoo does not implement fragmentation as defined in this paper. Also like other distributed SDN controllers such as ODL, Hyperflow etc., they are dedicated to traditional SDN networks and are not applicable to SDWSN.

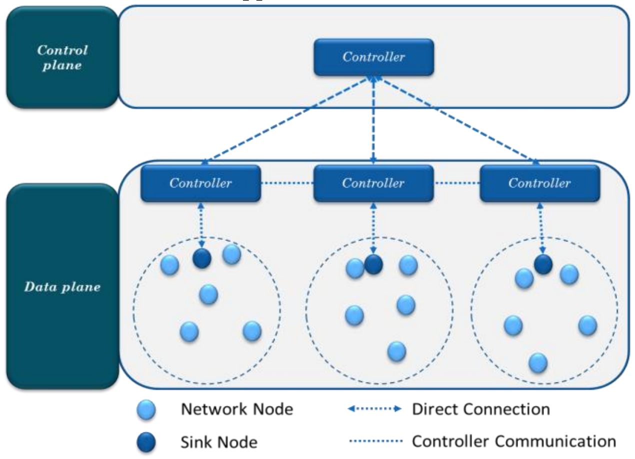

Fig. 1. Distributed control system with fragmentation.

Fragmentation takes distribution further by taking the control logic closer to the infrastructure devices. The reduced distance between the controller and the sink node saves much transmission power and also reduces pressure from its low data rates capacity thereby improving throughput. Fragmentation involves a two level control structure where there is a global or central controller overseeing the whole network and a local controller in charge of only a portion or segment of the network. The local controllers only have the knowledge of the portion of the network they are controlling. The characteristics of each controller as well as the sink node are as follows:

1) *Global controller:*

- Global view/knowledge of the network.
- Failure mechanism.
- Load balancing.
- SDN functionalities of a controller.
- Failure mechanism.
  - Failure of the global controller does not affect the operation of the network, only a temporal disruption of display and other functions that requires global knowledge.

- Replication method used to create redundancy.

2) *Local Controller*

- Take charge/control of a cluster.
- Updates the global controller.
- Local cluster knowledge.
- Lightweight for cost effectiveness.
- Failure mechanism.
  - Another local controller takes over.
  - Learns cluster state from global controller.
  - Sink connects to the closest controller.

3) *Sink Node*

- Connects to the local controller (the closest).
- Communicates with the sensor nodes.
- Relays/convey information to the local controller.
- Uses RF to communicate with the other sensor nodes and internet to connect to the local controller.
- Failure mechanism.
  - Sensor nodes find another closest sink node.

There are three fundamental things that distribution seeks to achieve: Reliability, Scalability and Efficiency. However a special consideration has to be accorded in SDWSN due to the inherent low capacity of WSNs.

Reasons for fragmentation:

- Dedicated controller to a cluster.
- Avoid having global knowledge on all controllers, reduce overhead cost by updates.
- Improve responsiveness.
- Allow proper isolation of sensed data types.
- Reduce redundancy on the local nodes.
- Updates to the global controller cannot affect the operation of the network i.e. delay, congestion.
- Low latency – Controller close to the sensor nodes.

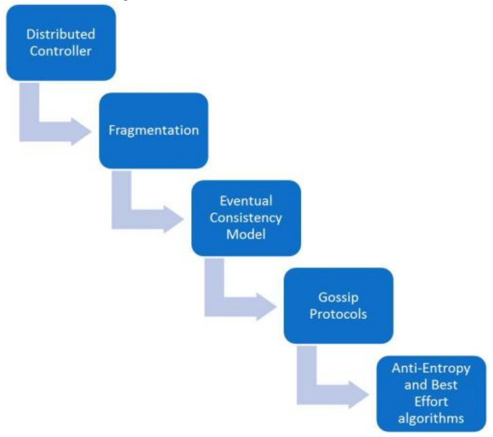
Distributed Controller
Fragmentation
Eventual Consistency Model
Gossip Protocols
Anti-Entropy and Best Effort algorithms

Fig. 2. The complete research structure.

Fig. 2 depicts the high level design of the research mechanism technique; each controller, herein referred as a fragment controlling a cluster of nodes. This diagram shows all concepts considered in this paper by order of application.

IV. Epidemic/Gossip Protocols

An epidemic is the spread of disease (infectious) to a large number of people in a population within a short period of time [33]; whilst Gossip is the spread of rumours in an informal way in the social circles. These two concepts, with the common denominator of "spread", inform the basis of Gossip/Epidemic protocols. Gossip protocols disseminate information across a distributed system using a gossip like method. A participant randomly pairs with a peer and exchange update information between them and after a period of time full consistency is reached.

The evolution gossip protocols was initially introduced by Demer's 1987 paper titled "Epidemic algorithms for replicated database maintenance" [34]. There are different gossip protocols in Best effort also called Direct mail, Anti-entropy and Rumour mongering. This paper focuses on Best effort and Anti-entropy.

Best effort ensures that every new event or update is sent to all other nodes immediately. Anti-entropy compares the replicas/nodes and reconciles the differences; Thus updating each copy to the newest [35]. Rumour-mongering floods the network with updates for a period of time sufficient enough to have all nodes updated. These methods suites networks with a moderate latency tolerance, however would potentially lead to latency challenges under high update load [35]. The SDWSN is envisaged to be a high update network with little to none latency tolerance and given the low capacity bandwidth; these two methods could be problematic.

The Best effort or Direct mail is event based. An update to other nodes is triggered by an occurrence of an event. Upon receiving an event, the node will broadcast that event to all other nodes in the cluster. An event could be any occurrence of an update such as a new node, node failure etc. The receiving nodes in the cluster evaluate the recency of the event. If recent, the matching entry will be updated accordingly. Otherwise, the sender will have to update its state. The following tables show the pseudocodes of the algorithms and their time efficiency. Table I shows the constructor which forms the base of all the algorithms discussed. Table II shows the pseudocode of the Best effort algorithm.

TABLE I CONSTRUCTOR

*Let **S** be a set of participants: **S = {p,q,r…..}***
*The state of the participant p is modelled as state: **K←V.T** where* ***K** is the set of Keys* ***V** is the set of values* ***T** is the set of Timestamps* *Thus the state: **s(k) ← (v,t)***

TABLE II BEST EFFORT ALGORITHM [34].

OPERATION	COMPLEXITY
Upon receiving event p.state (v,t$_i$) do *For each q ∈ S do* *Send state to q*	*O(n)*
Upon receiving state: (v,t$_i$) *If state.time < t then* *State ← (v,t)* *else (v,t) ←* ***(v,t$_i$)***	*O(1)*
Total time complexity O(n) + O(1)	*O(n)*

The Anti-entropy algorithm is almost similar to Best effort in content but different. Anti-entropy is periodic, thus the synchronisation occur after every set period of time. A peer periodically chooses a random partner from the list of peers (nodes) and starts to exchange state information. Thus peer *p* sends its state to *q*, and *q* applies it to its own state, this is called Push method. Otherwise in Pull method, *p* sends its state to *q* which only consists of keys and timestamps, then *q* responds with appropriate matching updates to *p*. Pull-Push method is the combination of both methods, while *q* sends updates to *p* as in the pull method, it also sends its outdated values to *p*. This is the most used and most efficient method. Thus using the Pull-Push method, a node sends its state to a peer node.

The peer node checks the received state for recency and if recent, it updates its state accordingly. Otherwise it sends a message to the sender node with a set of updates. Upon receiving the reply, the original sender (now the receiver) applies the changes, first by checking the recency, then updating if recent. If an entry is missing, it is requested. The pseudocode of the algorithm is described in Table III.

TABLE III ANTI-ENTROPY ALGORITHM [34], [35]

OPERATION	COMPLEXITY
For every T period do	
Randomly select a peer q from set of peers S	*O(nlogn)*
Send p.state: ***(v,t$_i$)*** *to q with state (v,t) # Update*	
Upon receiving p.state *#update*	*O(1)*
Check if p.state.time > q.state.time i.e t	
*q.state: (v,t) ← p.state:****(v,t$_i$)***	
else if (p.state.tim:e ***(v,t$_i$)*** *<q.state.time: (v,t))*	
send a reply to p	
Upon receiving a reply	*O(1)*
If p.state.time <q.state.time	
p.state: ***(v,t$_i$)*** *←q .state.time: (v,t)*	
if p.state: (v,t) ∈ S such that (vi,tj)∉ q	*O(1)*
request (v$_i$,t$_j$)	
Total time complexity O(nlogn)+O(1)+O(1)	*O(nlogn)*

The ONOS architecture is based on the eventual consistency data model; however applications that require stronger data guarantees can use the strong consistency model as an alternative. The strong consistency model is backed by RAFT [11] algorithm. The eventual consistency uses Best effort to update all other peers when an event occurs and Anti-entropy to resolve the differences amongst nodes.

The first intervention towards fragmentation is to change the behaviour of the gossip algorithms. The Best effort algorithm is redesigned to ensure that it only sends updates to the global controller, thus all updates triggered by events are sent to the global controller. Upon receiving an event, the node sends it to the global controller. This step reduces the computation of the algorithm from O(n) to O(1). The global controller then checks if the event is recent before updating its state. If the global controller does not have that entry, it is created; however if an entry exist in the global controller and not in the local controller, the local controller ignores the entry. This is to ensure that each local controller has a view and control of its cluster of the network. The Best effort algorithm is modelled as described by the pseudocode in Table IV for the fragmentation model. The steps described above on are further depicted by the flow chart on Fig. 3.

TABLE IV BEST EFFORT ALGORITHM WITH FRAGMENTATION

OPERATION	COMPLEXITY
Upon receiving event p.state: (v,t$_i$) do	*O(1)*
Send state to global controller gc: (v,t)	
Upon receiving state	*O(1)*
If p.state.time < gc.state.time then	
p.state ← gc.state:(v,t)	
else gc.state: ***(v,t$_i$)*** *← gc.state: (v,t)*	
if p.state: (v,t) ∈ S such that (v$_i$,t$_j$) ∉ q	*O(1)*
if sender is global controller	
ignore	
else request ***(v$_i$,t$_j$)***	
Total time complexity O(1)+O(1)+O(1)	*O(1)*

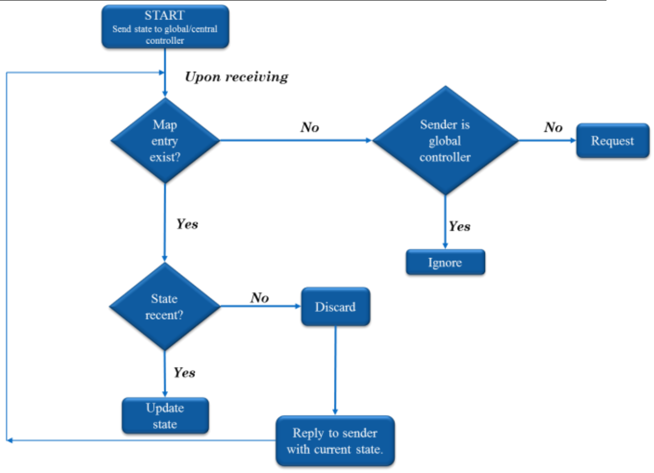
START
Send state to global/central controller
Upon receiving
Map entry exist?
No
Sender is global controller
No
Request
Yes
Yes
Ignore
State recent?
No
Discard
Yes
Update state
Reply to sender with current state.

Fig. 3. The flow chart of the Best effort algorithm with fragmentation.

The Anti-entropy algorithm is also redesigned from updating its peers to only update the main global controller. The Anti-entropy handling method is enhanced to be able to distinctively handle updates from either the global controller or the local controller. The local controller does the main updates to the global controller. If the local controller has new devices, the global controller will be updated during the data exchange. However the local controller cannot accept any new devices from the global controller (might be from other clusters). The local controller is envisaged to only get updates from the main global controller if it’s coming alive due to have been down or when taking over another down node. In case a node goes down, the sink node will connect to the next available controller node. To get the updated state, the new controller will do an Anti-entropy synchronisation with the global controller; however this would be rare given the nature of updates in SDWSN. The sensory data is mostly upstream and therefore most updates will be from the local controller to the global controller.

TABLE V ANTI-ENTROPY ALGORITHM WITH FRAGMENTATION

Operation	Complexity
For every T period:	*O(n)*
Synchronise with the global controller: send state	
if p.state: (v,t) ∈ S such that (v$_i$,t$_j$) ∉ q	*O(1)*
if sender is global controller	
ignore	
else request ***(v$_i$,t$_j$)***	
Total time complexity O(n)+O(1)	*O(n)*

The anti-entropy protocol between the local controller and the global controller is depicted in Table V. The steps of the algorithm are depicted on the flow chart in Fig. 4. As stated above, there are similarities between these two algorithms. The main difference is on the initiation stages, Best effort algorithm is triggered by an event while the Anti-entropy algorithm is periodic. The similarities are on the information exchange between the nodes. As shown in Fig. 3 and Fig. 4, the contents of the algorithms are similar.

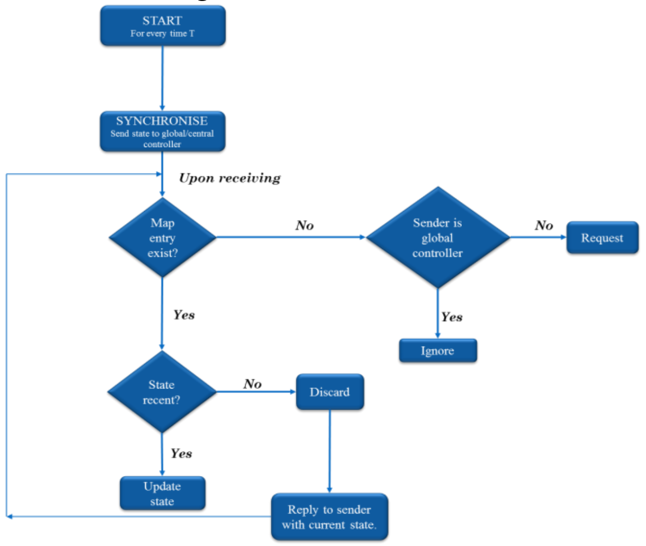

Fig. 4. The flow chart of the Anti-entropy algorithm with fragmentation.

A. *Time complexity*

Time complexity is the amount of time or steps an algorithm takes to run as a function given the length of the input. The more complex an algorithm is, the longer it takes to run. Although many scholars consider it insignificant or negligible because of the advanced processing capabilities of the modern systems, it is still vital to the SDWSN. The time complexities of the above are listed alongside the algorithms. The Best effort algorithm which sends updates to all peers upon a new event runs at O(n); after the change which sends all updates to the global controller; the time complexity reduces to O(1). On the other hand the anti-entropy algorithm changes from O(nlogn) to O(n).

V. Experimental Setup

The experimental evaluation seeks to determine the viability of the fragmentation model for the SDWSN. Hypothetically, the two algorithms used in the formation of the fragmentation model have an efficient time complexity as shown through the big O notation. The big O notation is a model used to measure the performance or complexity of an algorithm. This section describes the experiments used for this evaluation.

The experimental setup used SDN-WISE solution guidelines. Thus a Cooja simulator to establish SDN enabled sensor nodes making the SDWSN as well as the SDN-WISE ONOS controller. SDN-WISE 1, which uses ONOS 1.0.2, is used. ONOS provides a mastership service where there is a master and slave controllers. In a case of failure, one slave is elected to the master. In this context, each local controller becomes a master while the others in the cluster become slaves. The algorithms were implemented in the ONOS controller. Three tests experiments were conducted comparatively; a single central controller, a distributed controller and a distributed controller with fragmentation. The first two experiments were previously explored in [28], where a determination was made that a distributed control system for SDWSN is indeed possible and a necessity.

All the experiments were run in independent virtual machines (VM). All SDWSN simulations were conducted from a VM with 2GHz CPU and 2G RAM; the same VM is also used for the global controller. All other controller variations are conducted from different VMs with 2GHz CPU and 1G RAM specifications. Experiment A, as depicted in Fig. 5 uses a single central controller. Experiment B uses three distributed controller in a form of a cluster ran in three different VMs as shown in Fig. 6. This experiment used the symmetric ONOS, herein referred as ONOS original where all the controller instances eventually converge to an equal state. In experiment C, depicted in Fig. 7, three distributed controllers are used for the fragmentation as local controllers each independently controlling their own clusters as well as a global controller.

The experiments were evaluated through three main evaluation metrics in Round-Trip Time (RTT) or Round-Trip Delay, Standard Deviation, and Packet Error rate. The RTT measures the time it takes for a packet to traverse from the simulation to the controller and back. The standard deviation measures the variation of the time (RTT) or the delay across all packets recorded over the evaluation period of 300 seconds. This assists in determining the relational proportion of consistency or lack thereof. The packet error rate looks at the rate of packet loss.

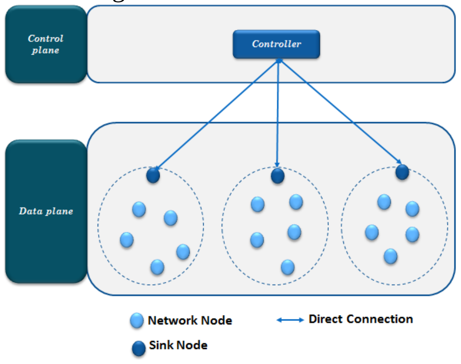

Fig. 5. Experiment A, single central controller.

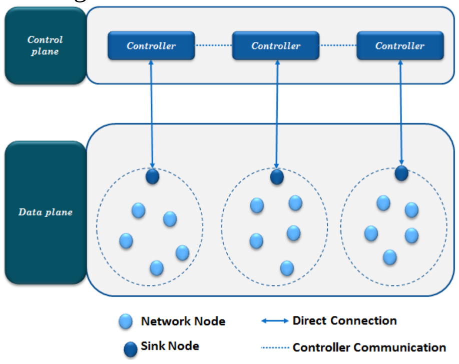

Fig. 6. Experiment B, distributed controllers.

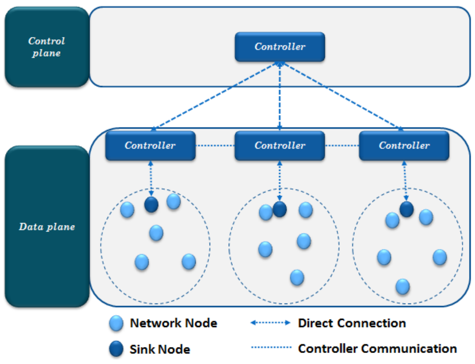

Fig. 7. Experiment C, distribution with fragmentation.

TABLE VI. RESULTS

	CENTRAL			DISTRIBUTED ONOS			DISTRIBUTED FRAGMENTATION		
Nodes	15	30	39	15	30	39	15	30	39
RTT ave	1087	1192	886	453	463	430	416	434	413
RTT min	7	5	5	183	193	202	174	172	208
RTT max	5252	10891	16062	2618	3673	4079	948	1284	1573
RTT med	694	589	519	413	424	378	397	406	384
RTT std dev	1001	1543	1171	216	269	284	125	145	136
Packet error	9	8	2	10	4	4	10	4	2
PER %	1.81	0.98	0.18	2.03	0.50	0.36	2.00	0.51	0.18
SYNC 1	779	355	1008	403	475	374	460	461	516
SYNC 2	1129	808	930	370	433	359	429	395	437
SYNC 3	810	440	849	483	596	403	340	458	516
SYNC ave	906	535	929	418	501	379	410	438	489
Number of Packets	497	815	1075	501	820	1095	510	830	1135

VI. RESULTS

This section provides the detailed results obtained from the described experiments. All the results are listed in Table VI. Each experiment was run using 3 sink nodes plus 15, 30, 39 sensor nodes respectively; herein referred as scenario 15, scenario 30 and scenario 39.

The RTT of the SYNC packet indicates the time it takes by the controller to process the first packet. This also highlights the controller setup time. The first sink nodes to establish a connection to the controller always has the lowest connection time. The results show that the average setup time is high on the central controller, all scenarios. They also show that the distributed controllers exhibit a better setup time. The fragmentation model took longer than the other distributed experiment, however the difference is not huge.

The central controller exhibited a higher RTT than the distributed versions. Fig. 8 depicts the RTT of scenario 39 (3 sink and 39 sensor nodes), other scenarios are not shown but they are relatively similar. Fig. 9 shows the average RTT for all scenarios. The distributed versions differ slightly, with the fragmentation model slightly lower than the original version, as shown in Fig. 8 and Fig. 9. The 30 nodes scenario exhibited a slightly higher RTT on average on all the scenarios. The results also show that the addition of nodes has a slight impact on the average RTT. In the first experiment, scenario 15 has the very least on the minimum RTTs compared to the rest. This is because the first few packets enjoy the free reign before the other sinks joins in and can be largely attributed to the manual method used to connect to the controller. In contrast, the very same scenario (central) exhibits the highest maximum RTTs across all experiments and scenarios. The maximum RTTs are also in proportion of the increase in nodes. The fragmentation model exhibit the lowest minimum and maximum RTTs on the distributed versions.

The contrast between the minimum RTTs and the highest RTTs on the central controller experiment has a major impact on the standard deviation. Thus the variation of time delay is inconsistent with high levels of fluctuations. The distributed experiments show a lower variation with the fragmentation exhibiting the lowest as depicted in Fig. 10.

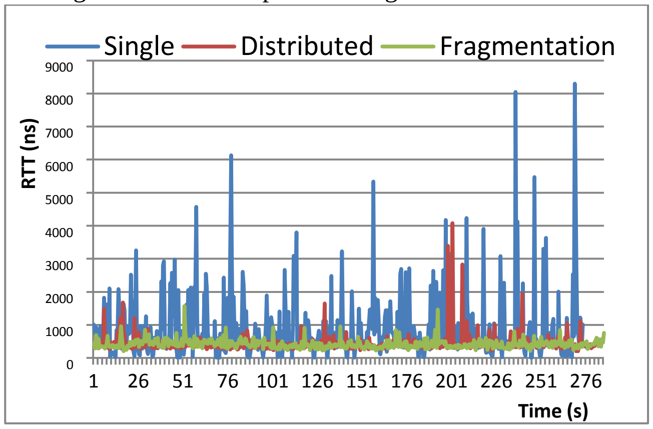

Fig. 8. Average RTT on 39 nodes.

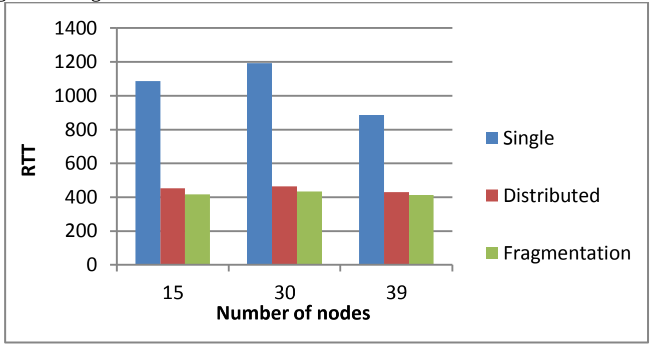

Fig. 9. RTT vs number of nodes.

The packet error rate is very minimal however not negligible. This can be attributed to the SDN-WISE inherent architectural maturity as described in [26]. The packet error rate is relative across all experiments. It is also high when the numbers of nodes are few. As shown in Fig. 11 the total packet loss is around and less than two percent of the total packets across all experiments. The total number of packets increases in direct proportion of the number of nodes. The fragmentation model produced more packets than the other experiments as shown in Fig. 12.

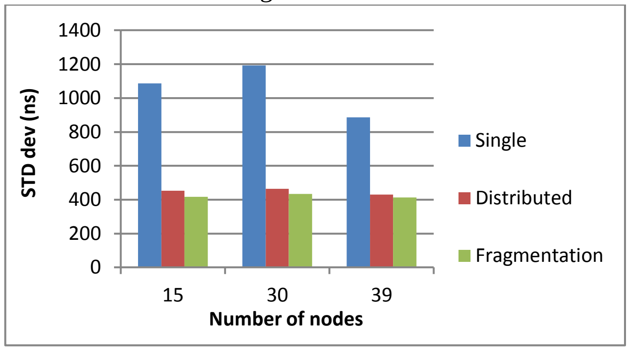

Fig. 10. Standard deviation.

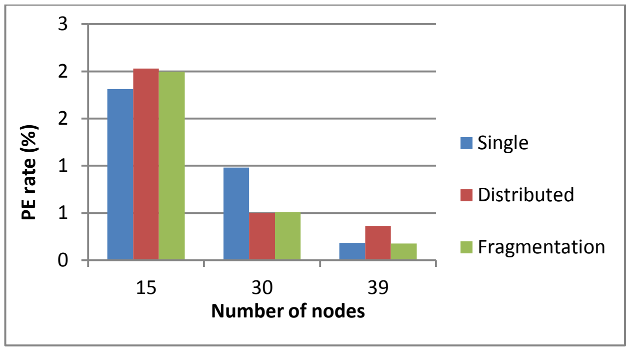

Fig. 11. Packet error rate.

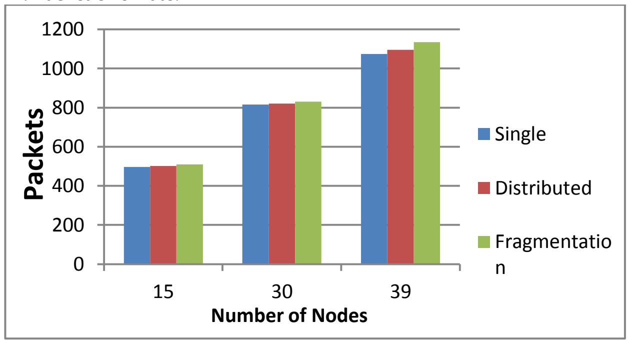

Fig. 12. Number of packets produced during the test.

VII. Discussion

The purpose of the proposed system is to investigate the viability of using fragmentation as a method of distribution for SDWSN controller. The controller setup time shows that the fragmentation model suffers some additional delay compared to the original version because of the two-level architecture. However this only affects the first packet of handshaking. The subsequent packets are handled and processed by the local controllers. The RTT is a very important measure to gauge the interaction between the SDWSN simulation and the controller. Although some research work, such as Erickson [36] and Dixit *et al.* [37] asserts the controller can handle millions of packets per second, it is important to qualify this on the perspective of SDWSN. The average RTT is high on the central controller as compared to the distributed versions. The central controller performs well in the initial stages and stagger as more packets come through. This can also be observed as the number of nodes increases. Although limited in the scaling of the nodes, it can be deducted that scalability has direct effect on RTT. This can also be affirmed by the distributed version. The two distributed versions of the experiments exhibit consistent RTTs across. The fragmentation is slightly lower, which is promising improvement. The slight difference in the average RTT is because the fragmentation model produced more packets than the other distributed experiment and thus the average is out of more samples. The real extend of the difference can be observed from the variation of the RTTs in the standard deviation; the quantity does not affect the average. The improvement can also be qualified by looking at the minimum and maximum RTTs. The minimum average RTTs is relative but the maximum RTT on the fragmentation is lower across all scenarios and also, the variation against the increase in nodes is minimal compared to the other two experiments. However the real extend of the improvement can only be conclusively ascertained with an extended degree of scalability which at this stage could not be reached due to the limited capacity of the simulation tool.

The variation trajectory in proportion of the scenarios is clearly observed in the standard variation. The central controller exhibited the worst with an average of over 1000 ns, while distributed versions in experiment B and C are better at over 200 and 100 ns respectively. This shows amongst others, that the fragmentation model does exhibit a consistent variation. The packet error rate or loss is very low; this is inherent and thus the distribution had no impact. However this needs a further evaluation on a large network for absolute certainty. The three experiments produced different number of packets. The packets are mainly determined by the number of nodes, the time of the test, and the efficiency of the controller(s). The first two are apparent and common logic; the third is intrinsically related to the response time, which is determined by the effectiveness of the protocol. This is because the distributed controllers operate independently and closer to the infrastructure elements and also because of the improved timed complexity of the algorithms. The only caveat is on the first handshaking packet. On the original ONOS, the lateral exchange of data amongst the controllers increases the delay, the fragmentation model averts this.

VIII. Conclusion

The SDWSN model is a new networking paradigm that arises as a result of applying SDN into WSN. The controller, as in other SDN based systems, holds the intelligence of the entire network. This paper proposes an efficient distribution mechanism for the SDWSN controller using fragmentation.

Fragmentation entails distributing the control logic into different segments (fragments), each responsible for a particular cluster. This method consists of small and lean local controllers closer to the infrastructure elements. These local controllers operate independently and only occasionally with a global controller. The proposed system uses Eventual consistency data model, which consists of Best effort and Anti-entropy algorithms. These algorithms are restructured and reused to ensure fragmentation.

The results shows that distributed controllers perform better than a central controller. The results showed a slight improvement in performance for the fragmentation as compared to the original implementation. Therefore, it can be deducted that distribution is indeed essential and that fragmentation does bring efficiency to SDWSN control system. The improvement in time is particularly important because many industrial Internet of Things systems and application which the SDWSN is envisaged to play a critical role are time sensitive. The proportional difference can relatively improve as the network scales. The future work thus needs to investigate the scalability issue in detail to adequately gauge impact the proposed system. The efficiency of the SDWSN framework has an immense contribution to the digital industrial revolution.

REFERENCES

[1] P. Hu, "A System Architecture for Software-Defined Industrial Internet of Things," in *2015 IEEE International Conference on Ubiquitous Wireless Broadband*, 2015.

[2] B. Cheng, L. Cui, W. Jia, W. Zhao, and P. H. Gerhard, "Multiple Region of Interest Coverage in Camera Sensor Networks for Tele-Intensive Care Units," *IEEE Trans. Ind. Informatics*, vol. 12, no. 6, pp. 2331–2341, 2016.

[3] J. W. Guck, M. Reisslein, and W. Kellerer, "Function Split Between Delay-Constrained Routing and Resource Allocation for Centrally Managed QoS in Industrial Networks," *IEEE Trans. Ind. Informatics*, vol. 12, no. 6, pp. 2050–2061, 2016.

[4] D. Kreutz, F. M. V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[5] J. Li, L. Huang, Y. Zhou, S. He, and Z. Ming, "Computation Partitioning for Mobile Cloud Computing in a Big Data Environment," *IEEE Trans. Ind. Informatics*, vol. 13, no. 4, pp. 2009–2018, 2017.

[6] D. A. Chekired, L. Khoukhi, and H. T. Mouftah, "Decentralized Cloud-SDN Architecture in Smart Grid: A Dynamic Pricing Model," *IEEE Trans. Ind. Informatics*, vol. 14, no. 3pp. 1220–1231, 2017.

[7] G. S. Aujla, N. Kumar, A. Y. Zomaya, and R. Rajan, "Optimal Decision Making for Big Data Processing at Edge-Cloud Environment: An SDN Perspective," *IEEE Trans. Ind. Informatics*, vol. 14, no. 2, pp. 778–789, 2017.

[8] M. Jacobsson and C. Orfanidis, "Using Software-defined Networking Principles for Wireless Sensor Networks," in *In: Proc. 11th Swedish National Computer Networking Workshop, 2015*, 2015, pp. 1–5.

[9] G. A. Akpakwu, B. J. Silva, G. P. Hancke, and A. M. Abu-Mahfouz, "A Survey on 5G Networks for the Internet of Things: Communication Technologies and Challenges," *IEEE Access*, vol. 5, no. 12, pp. 3619 - 3647, 2017.

[10] K. M. Modieginyane, B. B. Letswamotse, R. Malekian, and A. M. Abu-Mahfouz, "Software defined wireless sensor networks application opportunities for efficient network management: A survey," *Comput. Electr. Eng.*, vol. 66, no. 2, pp. 274-287, 2018..

[11] D. Ongaro and J. Ousterhout, "In Search of an Understandable Consensus Algorithm," in *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, 2014, pp. 305–320.

[12] L. Lamport, "Paxos Made Simple," *ACM SIGACT News*, vol. 32, no. 4, pp. 51–58, 2001.

[13] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Commun. Surv. Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[14] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and Haiyong Xie, "A Survey on Software-Defined Networking," *IEEE Commun. Surv. Tutorials*, vol. 17, no. 1, pp. 27–51, 2014.

[15] H. I. Kobo, A. M. Abu-Mahfouz, and G. P. Hancke, "A Survey on Software-Defined Wireless Sensor Networks: Challenges and Design Requirements," *IEEE Access*, vol. 5, pp. 1872–1899, 2017.

[16] M. Ndiaye, G. P. Hancke, and A. M. Abu-Mahfouz, "Software Defined Networking for Improved Wireless Sensor Network Management : A Survey," *Sensors*, vol. 17, no. 5:1031, pp. 1–32, 2017.

[17] G. M. Omolemo and A. M. Abu-Mouhfaz, "Utilising Artificial Intelligence in Software Defined Wireless Sensor Network," in *The 43rd IEEE conference of Industrial Electronic Society*, 2017, pp. 6131–6136.

[18] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov, "Security in Software Defined Networks: A Survey," *IEEE Commun. Surv. Tutorials*, vol. 17, no. 4, pp. 2317–2346, 2015.

[19] D. Kreutz, F. M. V. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," *Proc. Second ACM SIGCOMM Work. Hot Top. Softw. Defin. Netw. - HotSDN '13*, p. 55-60, 2013.

[20] S. W. Pritchard, G. P. Hancke, and A. M. Abu-Mahfouz, "Security in Software-Defined Wireless Sensor Networks: Threats, challenges and potential solutions," in *The15th IEEE International Conference of Industrial Informatics*, 2017, pp. 168 – 173.

[21] J. Medved, R. Varga, A. Tkacik, and K. Gray, "OpenDaylight: Towards a Model-Driven SDN Controller architecture," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, 2014, pp. 1–6.

[22] P. Berde, W. Snow, G. Parulkar, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, and P. Radoslavov, "ONOS," in *Proceedings of the third workshop on Hot topics in software defined networking*, 2014, pp. 1–6.

[23] A. Tootoonchian and Y. Ganjali, "HyperFlow: a distributed control plane for OpenFlow," in *Proc. of the 2010 internet network management conference on Research on enterprise networking*. USENIX Association, pp. 3–3, 2010.

[24] S. H. Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," *Proc. first Work. Hot Top. Softw. Defin. networks*, pp. 19–24, 2012.

[25] A. A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. Kompella, "ElastiCon," in *Proc. of the 10th ACM/IEEE symposium on Architectures for networking and communications system*, 2014, pp. 17–28.

[26] L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, "SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for WIreless SEnsor networks," in *2015 IEEE Conference on Computer Communications*, 2015, pp. 513–521.

[27] SDN-WISE, "Controlling heterogeneous networks using SDN-WISE and ONOS." [Online]. Available: http://sdn-wise.dieei.unict.it/docs/guides/GetStartedONOS.html. [Accessed: 26-Apr-2017].

[28] H. I. Kobo, G. P. Hancke, and A. M. Abu-Mahfouz, "Towards A Distributed Control System For Software Defined Wireless Sensor Networks," in *43rd IEEE IECON*, 2017, pp. 6125-6130.

[29] B. T. de Oliveira, C. B. Margi, and L. B. Gabriel, "TinySDN: Enabling multiple controllers for software-defined wireless sensor networks," *2014 IEEE Latin-America Conference on Communications*, 2014, pp. 1–6.

[30] B. T. de Oliveira and C. B. Margi, "Distributed control plane architecture for software-defined Wireless Sensor Networks," in *2016 IEEE International Symposium on Consumer Electronics*, 2016, pp. 85–86.

[31] P. Bailis and A. Ghodsi, "Eventual Consistency Today: Limitations, Extensions, and Beyond," *Queue*, vol. 11, no. 3, p. 20, Mar. 2013.

[32] C. Coronel and S. Morris, *database systems design implementation and management 11e*, 11th ed. 2015.

[33] J. Holliday, R. Steinke, D. Agrawal, and A. El Abbadi, "Epidemic algorithms for replicated databases," *IEEE Trans. Knowl. Data Eng.*, vol. 15, no. 5, pp. 1218–1238, Sep. 2003.

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

[35] R. van Renesse, D. Dumitriu, V. Gough, and C. Thomas, "Efficient reconciliation and flow control for anti-entropy protocols," *Proc. 2nd Work. Large-Scale Distrib. Syst. Middlew*, p. 1-6, 2008.

[36] D. Erickson, "The beacon openflow controller," *Proc. Second ACM SIGCOMM Work. Hot Top. Softw. Defin. Netw.*, pp. 13–18, 2013.

[37] A. Dixit, F. Hao, S. Mukherjee, T. V Lakshman, and R. Kompella, "Towards an Elastic Distributed SDN Controller," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 7–12, 2013.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2018.2821129, IEEE Transactions on Industrial Informatics

Hlabishi Kobo received his Bachelor and Masters of Science in Computer Science from the University of the Western Cape in 2010 and 2012 respectively. He worked for Telkom SA as a technology architect. He is currently a PhD researcher at the Council for Scientific and Industrial Research (CSIR) and a PhD student at the University of Pretoria. His main interests are software defined networking, software defined wireless sensor networks and software architecture.

Dr Abu-Mahfouz received his MEng and PhD degrees in computer engineering from the University of Pretoria. He is currently Principal Research Engineer at the Council for Scientific and Industrial Research (CSIR), Research and Innovation Associate at Tshwane University of Technology and Extraordinary faculty member at University of Pretoria. His research interests are wireless sensor and actuator network, low power wide area networks, software defined wireless sensor network, cognitive radio, network security, network management, sensor/actuator node development, smart grid and smart water systems. Dr Abu-Mahfouz is an associate editor at IEEE Access, Senior Member of the IEEE and Member of many IEEE Technical Communities. He is currently the principal investigator of a large multidisciplinary collaborative project entitled "Smart Water Management System". Dr Abu-Mahfouz is the founder of the Smart Networks collaboration initiative that aims to develop efficient and secure networks for the future smart systems, such as smart cities, smart grid and smart water grid

Gerhard P. Hancke (S'99-M'07-SM'11) is an Assistant Professor with City University of Hong Kong. He received B.Eng and M.Eng. degrees in Computer Engineering from the University of Pretoria (South Africa), in 2002 and 2003 respectively, and a Ph.D. degree in Computer Science from the University of Cambridge (UK) in 2008. His research interests include system security, embedded platforms, and distributed sensing applications. He is also an Extraordinary Senior Lecturer at University of Pretoria, South Africa.