Subneting Software Defined Wireless Sensor Networks to Support Mobility

Ahmed Nader al-Dulaimy
Institute for Computer Science
University of Koblenz - Landau
Koblenz, Germany
Email: aldulaimy@uni-koblenz.de

Prof. Dr. Hannes Frey
Institute for Computer Science
University of Koblenz - Landau
Koblenz, Germany
Email: frey@uni-koblenz.de

***Abstract*—Mobility in wireless sensor networks plays a significant role in many applications, especially for the internet of things (IoT) and industrial production workflow. However, managing the mobility of nodes in such networks is often problematic mainly because of the dynamics of industrial environments combined with the limitation of the node's resources. Since software-defined networking is a powerful technique to support today's networks' manageability and scalability, both wireless sensor networks and software-defined networks combined are gaining significant interest in research and industry. In this paper, we introduce an approach to handle the mobility and disconnectivity of the sensor nodes in a software-defined network. This approach takes advantage of splitting the network into sub-networks by its address based on subnetting wireless sensor networks (Sub-WSN) and keeping the network topology up to date in the controller by exchanging periodical hello messages. We treat all the nodes in the network as mini controllers and implement an application into the controllers to hand over nodes between the subnets. We evaluated the proposed approach by using Cooja simulator in Contiki. The simulation result yields substantial low control messages, handover time, and latency in handling dynamic networks against a state of the art approach.**

***Index Terms*—Wireless Sensor Network, Software Defined Networking, Hierarchical Addressing, Simulation study, Dynamic Network**

I. Introduction

In contrast to traditional network, a software-defined network separates the functionality of the network into two parts, data plane and control plane. The data plane is presented by the forwarding devices of the data flow without any control logic. The logic and intelligence of the network are moved to a centralized controller to form the control plane. The control plane consist of a controller (or multi controllers) and applications that are resided in the controller to shape the network by instructing the behaviour of the forwarding devices.

SDN is gaining momentum for scalability and manageability characteristics as the world trend toward connecting everything through all IP-based to form the internet of things (IoT). Intelligent applications are emerging widely in smart water networks, agriculture, environment monitoring, smart metering, health care, intelligent transportation. On the other hand, Wireless Sensor Networks (WSN) play an essential role in such innovative applications by connecting intelligent objects.

However, whenever SDN is used in wireless networks, all achievable gains must be traded against the overhead to keep the controller's view of the whole network up to date. Applicability of the SDN concept stands or falls on resource-efficient means for maintaining the necessary network organization. One of the fundamental building blocks are message and memory-efficient means to organize multi-hop communication paths between nodes and controller. Nevertheless, it is easier to manage a scalable and dynamic wireless sensor network when separating its control from the data plane.

We introduced Sub-WSN (subnetting wireless sensor network) in [1], an approach to easily manage and overcome the control messages expenses of the network by dividing it into sub-networks with the use of the address of the node. The contribution of this paper is to reconnect a mobile, newly joined node or a broken link connection to the sink in a dynamic network by using Sub-WSN and utilizing the information stored in the controller with less controlling messages and faster reconnectivity to the network.

The paper is organized as follows: section II provides a literature review on SDNs in the context of WSNs. This is followed by section III giving an overview on Sub-WSN [1], and SDN-WiSE [2], SDN environments for WSNs which we used to implement and to compare our approach. Our approach is then presented in section IV, followed by section V where we compare the approach against SDN-WiSE by example and in terms of a series of simulation runs. Finally, section VI concludes this paper and points towards possible future extensions of our approach.

II. Related Work

A. Introduction of the SDN paradigm in WSNs

Software-defined networking (SDN) is an evolving paradigm that increasingly draws the attention of the researcher. It can reduce the complexity of networks in terms of scalability and significantly improve management capabilities. The preceding researches that adopt SDN in wireless sensor networks have brought many pioneering solutions. Nevertheless, more investigations need to be

ISBN 978-3-903176-42-3 ©2021 IFIP

considered regarding node mobility and highly dynamic link changes. There is a need to cope with these obstacles by identifying how to handle mobility by breaking the network into smaller groups to be easily managed and updated. As discussed in the following, the software-defined wireless sensor network is divided into groups by clustering it, but specific fixed nodes or terminals needed to handle mobility in some research works.

There is a large number of papers that have dealt with SDN in WSN (see the recent survey [3] for example). However, a small amount of work is about clustering the network to tackle the problem that it is often arduous or even impossible to learn and maintain broken connections between nodes or handle the nodes' mobility. Reducing the amount of information to be maintained in the existing clustering-based SDN-WSN approaches suffer from: (1) more extra fixed controllers or specific fixed devices have to serve as cluster heads and (2) still control message sizes are enormous since the cluster formation is not reflected in the addressing scheme. As discussed in the following, none of the existing approaches has considered using the subnetting the nodes by the address (the approach we used from [1]) to manage mobility and links restoration, the approach we consider in this paper.

B. Clustering for SDNs in WSNs

The authors in [4] clustered the network by using the whale optimization algorithm. The SDN controller divides the sensing area into virtual zones (VZ) regarding the node density in the zones to balance the number of cluster heads. In [5] they proposed a centralized load balancing clustering algorithm (C-LBCA) where they moved the clustering calculation to the SDN controller (or controllers) by using cloud resources. The SDN controller utilises particle swarm optimization (PSO) for clustering and load-balancing. Both proposals showed more efficiency regarding the amount of data sent to the sink, though they did not reduce control message overhead.

The authors of [6] discussed that the use of software-defined wireless sensor networking (called SDWSN in the work) might help to deal with some WSN problems (e.g. energy-saving and the management of the network). However, in their proposed SDN architecture, a specified base station is needed to calculate routes.

There are some new approaches to divide the network into clusters of nodes. In [7], the authors introduce an approach, based on the topology characteristics and the quality of the communication channel between the nodes, to form coalitions of the collaborative machine to machine devices. In contrast, the author in [8] introduced another approach by organizing the nodes in non-overlapping clusters depending on code division multiple access schemes.

The authors in [9] propose the integration in the SDN controller of a traffic manager, which is a routing process to assigning different routes depending on the different flows. Moreover, they incorporated a modified TSCH (Time Slotted Channel Hopping) protocol in the SDN-WISE framework in order to send the TSCH schedule.

In our previous work [1], we introduced an approach to divide the software-defined wireless sensor network into sub-networks utilizing a hierarchical addressing scheme to manage the network efficiently. It is based on a tree of address masks that are used to split the network into subnets. Each subnet has a range of host addresses and a subnet head, which acts as a gateway to its children. For showing flexibility and efficiency in reducing the controlling messages and feasibly managing the routing in a massive network and small ones, we are using this approach to implement our approach to handle node mobility.

C. State of Art Implementations

Two well-known states of the art implementations of SDN for WSN for real sensor network hardware are Tiny-SDN [10], and SDN-WISE [2].

Tiny-SDN introduces multiple controllers to cluster the network to reduce control messages overhead and handle mobility. Tiny-SDN shows flexibility in communication; however, it introduces memory overhead. Furthermore, it needs specific network components, which are SDN controller nodes that serve as SDN controller hosts (cluster heads), and it needs SDN enabled nodes as well, which are the SDN end devices.

In SDN-WiSE, the controller is connected to the sink, and all the other nodes are operated as OpenFlow switches. The controllers in SDN-WISE drive applications that can shape the network behaviour by injecting rules and actions in each node. However, SDN-WISE does not employ clustering and neither uses subnetting to overcome control message sizes.

The paper in [11] present an SDN approach, which is built upon SDN-WiSE, to handle node mobility and scheduling in Industrial Wireless Sensor Networks (IWSN). The approach is based on the Time Slotted Channel Hopping (TSCH) protocol of the IEEE 802.15.4 standard, which they call Forwarding and TSCH Scheduling over SDN (FTSSDN). However, this approach has three different sets of nodes: Fixed nodes, Mobile nodes and Sink nodes. Therefore, it can control only the mobility of some network nodes, while the rest should be stationary in their positions.

We contribute to SDN in WSN research by implementing and building an application to handle mobility in a hierarchical addressed network based on trees of subnet masks. Each node in the network is acting as SDN mini controller. The SDN controller is used to calculate the best path and update addresses to hand over mobile nodes from one parent to another. By example and simulation, we show that the control messages, path packets, handover time and latency are significantly reduced compared to if we would handle mobility in SDN-WiSE. SDN-WiSE represents the so far existing WSN-SDN approaches that are all not challenging mobility using this type of hierarchical subnet organization. Though, we are not comparing against TinySDN or the approach in [11] because they require specific network hardware that acts as a controlling fixed cluster head. It is not as flexible as our approach with SDN-WiSE, where any node can be a cluster head and move freely.

III. Subnetting Software Defined Wireless Sensor Network in a Nutshell

Subnetting the wireless sensor network by its address was introduced in [1]. It is developed as an application in the controller of a modified SDN-WiSE.

A. SDN-WiSE

An approach by [2] is meant as a solution for software-defined networks in wireless sensor networks to lower control messages between controllers and nodes and to make the nodes programmable in order to process and execute openFlow operations. SDN-WiSE is the first OpenFlow-like solution implementation for WSN [1].

OpenFlow is a standardized protocol for software-defined networking which operates as an interface to join network switches with a controller. Within the OpenFlow protocol specification, there are flow rules that impact the forwarding switches' performance. The switch forwards packets by using one or more flow tables as shown in figure 1. These flow tables consist of (1) sets of rules matching the flow packets passing the switch, (2) the action to be exerted by the switch for these packets and (3) counters for statistics. The controller installs these flow rules on the switches. More specifications are introduced as the OpenFlow revision is developing (e.g. matching against multiple flow tables in protocol specification v1.2 and combining meters and queuing traffic to perform quality of service in v1.3), yet the essential idea of matching flows and taking action is nevertheless the same [12] [1].

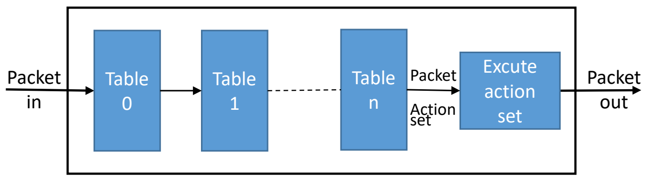


Fig. 1. Packets matching against multiple tables

Each node in SDN-WISE holds an array of accepted IDs and each packet's header includes an accepted ID in order to deal with the broadcasting nature of wireless communication. On receiving a packet, the node will iterate through its WISE flow table entries, if the ID is listed, searching for a match in the matching rule field. Once the matching rule is found, the linked action in the action field will be performed. The matching rule can consider any bit or part of the current packet and compare it with a particular rule. The action on that packet can be forwarding, dropping or and modifying. A request packet would be sent from the node to the controller if there were no matches in the flow table, requesting an instruction to handle such packets [1].

B. Operation of Sub-WSN

The network address is divided into subnets as illustrated in figure 2. Each subnet has a range of addresses covering its tree of nodes. These addresses can be filtered by applying a mask to each subnet [1].

Sub-WSN is using the topology discovery protocol (TD) of SDN-WISE for i) managing the network layout and identifying the next sink hop for each node; ii) determining the current distance to the sink; iii) including the battery level and iv) creating a list of neighbours for each node. After obtaining all network information in the controller (by the TD protocol), Dijkstra's algorithm is applied to determine the shortest path from each node to the sink. Nodes along each one of these paths receive new addresses. Each node in each path is the subnet head to the following nodes in that path from the sink downwards, as shown in figure 2 [1].

To each node that sends an openFlow request packet, the Sub-WSN application responds with a path packet. The path packet contains the parent's address, the node's new address, and the mask. As the path packet reaches the parent node, the parent appends a matching rule entry in its flow table containing:

- The address of the new child
- The mask that belongs to that child sub-network
- Along with the action to be taken to any future coming packets matching the rule

The rule is to match the masked destination address of the incoming packet with the child's address. If there is a match, it will forward the path packet to that child. The child node also appends a matching rule entry to its flow table by receiving the path packet. This entry is a rule for packets destined to the sink and the action to forward this kind of flow to the node's parent address [1].

IV. Mobility on Software Defined Wireless Sensor Network

Keeping track of the nodes in dynamic wireless sensor networks is not an easy task, especially when each node should be reachable individually with its specific address or ID to be updated and do a particular task. We introduce an approach to reconnect disconnected mobile nodes or moved away from their parent nodes for a dynamic network; by applying periodical hello messages and handover application in the controller, the node will get a new address (from an address inventory) to connect to a new parent, as illustrated in figure 6. In this section, we are describing in detail this approach.

A. Mobility and Hello Messages

A hello-packet is periodically sent through the network. The sink is initiating this process by sending a hello packet. This hello packet will trigger every node receiving it to respond with a hello packet in turn. That will trigger the nodes which receive these packets to send one and so on. During this process, a hello packet from the sink will propagate through the whole network; hence every node still connected to this network can observe the change within its surrounding.

The hello packet consists only of a packet length, a packet ID, packet type and the source address as shown in figure 3. Since the packet is destined to all reachable nodes, no destination address is needed in the packet, which is saving

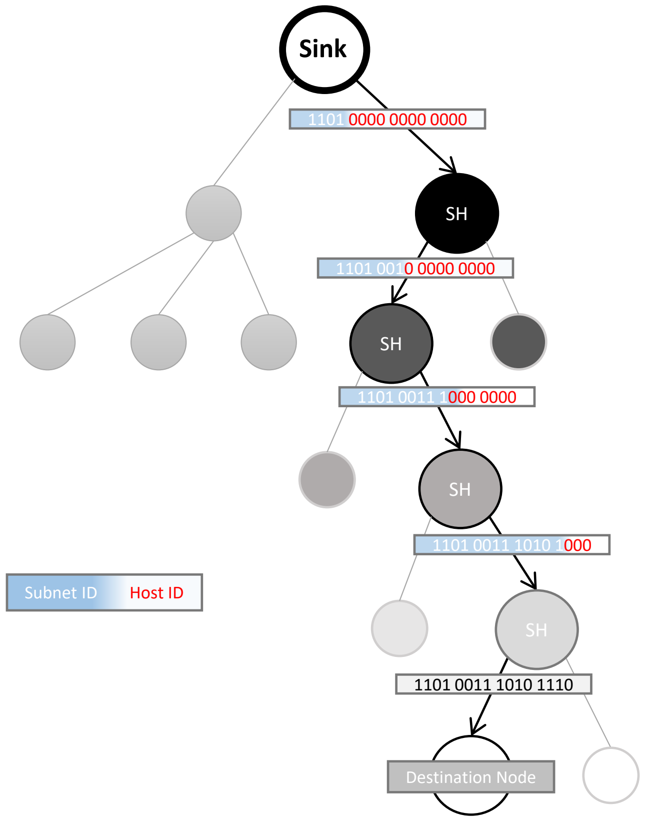
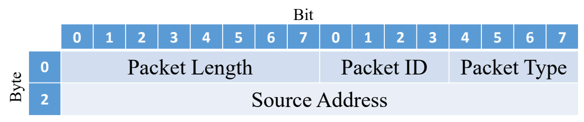


Fig. 2. Subnetting SDN WSN by the network address

some resources. In our implementation, hello packets received with RSSI lower than threshold of -60 dBm will be neglected as of [2]. The packet ID in the hello packet is to identify each packet, so the node can recognize if it has already responded to this packet with a hello packet or not.



Byte \ Bit	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	Packet Length								Packet ID				Packet Type			
2	Source Address															

Fig. 3. Hello packet header

The sink sends a hello packet every precalculated period. This period can be calculated in the network controller depending on the average number of neighbours per node and the number of the handover in a certain time (average speed of the nodes). Figure 4 is the diagram of handling the hello packet in the node. The node will check first the packet ID. If it has not responded to this packet ID, it will send a hello packet with the same ID and updating its hello counter.

As in the diagram, any node that receives a hello packet will check the packet source address against its neighbours' list. If the source is not in the list, the list will be updated with a new neighbour entry. Since the subnet ID part of the address of any node can show at which level the node is (how far it is from the sink), no need to send the number of hops from that node to the sink compared to that of TD protocol.

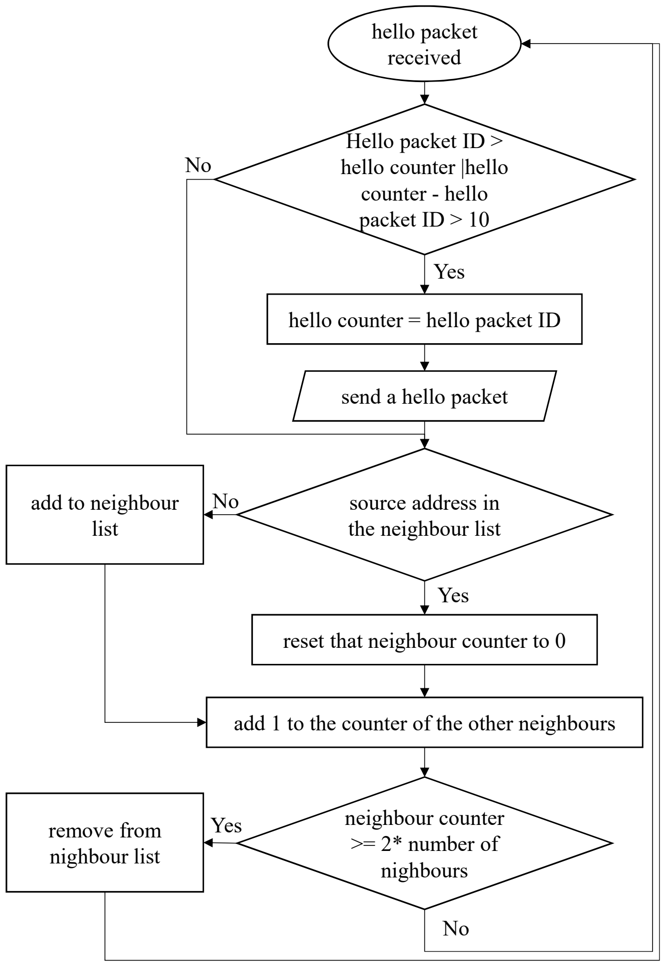

hello packet received
Hello packet ID > hello counter |hello counter - hello packet ID > 10
No
Yes
hello counter = hello packet ID
send a hello packet
source address in the neighbour list
No
add to neighbour list
Yes
reset that neighbour counter to 0
add 1 to the counter of the other neighbours
neighbour counter >= 2* number of nighbours
Yes
remove from nighbour list
No

Fig. 4. Hello packet received in a node daigram

The neighbour's list contains a counter for each neighbour to check if the neighbour is still connected. It is set to 0 each time a hello packet is received from that neighbour and adding one to the other neighbours' counters in the list. Then the node will check the neighbours' counter; if the counter value is more than double the number of neighbours, two hello messages were missed; hence That neighbour is no longer in the surroundings of the node (either disconnected or moved away), and it will be removed from the node neighbours' list.

A node will send a request packet to join another tree in case the node is not receiving a hello-packet from its parent node, or the parent node RSSI difference to another neighbouring node of the same level is more than 15 dBm and the parent RSSI is between -60 to -50 dBm, or the new neighbour is in a higher level than the parent node level. In short, the node will send the request packet to the neighbour with fewer hops to the sink and with higher RSSI.

B. *Addressing Inventory*

To accommodate the need for addresses of the moving nodes, we use an address inventory to calculate the number of the levels needed and distribute the addressing to create redundancy of addresses for each level, i.e. when a newly joined node is asking for an address, it will find one.

In the controller, after the network's initiation, we are using an algorithm to calculate the maximum needed addresses for each level and split the address's bits accordingly. In our simulated network, the 16 bits of the address is split into groups of bits for each level. Each level gets several bits to cover the most significant number of child nodes in any cluster of that level and more extra for mobile nodes. For example, if the largest cluster in a second level has nine child nodes, the number of dedicated bits for that level will be 4 (16 addresses) as shown in figure 5. That will cover the addresses for the nine nodes and seven more nodes for future joining or moving nodes.

Moreover, if there will be extra bits after splitting the address for the entire network, the extra bits will be added one by one for each level, starting from the levels with the least number of spared addresses and the highest to the lowest levels. The example in the figure shows that level three has no spare addresses, so the first extra bit is added to it. Although both the first and the fourth level have just one spared address, the first level is higher in the hierarchy. That is why the second extra bit is added to the first level.

Max. Child Nodes per Level	Bits Needed for Each Level / Number of Addresses left for The Inventory			
Level 1 = 3 Childs	2 bits / 1	**2 bits / 1**	**3 bits / 5**	3 bits / 5
Level 2 = 9 Childs	4 bits / 7	4 bits / 7	4 bits / 7	4 bits / 7
Level 3 = 8 Childs	**3 bits / 0**	**4 bits / 8**	4 bits / 8	4 bits / 8
Level 4 = 15 Childs	4 bits / 1	4 bits / 1	**4 bits / 1**	**5 bits / 17**
Number of The Used Address Bits out of 16	13 bits	14 bits	15 bits	16 bits

Fig. 5. Addressing inventory table example

C. Hand Over to New Cluster

We introduced to the nodes in the network an application HOSub (Hand Over Subnet) to handle node mobility so that any node in the network can act as a mini controller of the tree below it. This application will be triggered after receiving the request packet mentioned above sent to the new potential node parent. HOSub will react first by finding a new address in the address inventory then send back an update packet to the source node. The update packet contains only the source node address as the destination address, the new parent address as the source and the new address. This packet will trigger the HOSub application in the source node to change the address of the node, update its flowtable and broadcast an update packet to its child nodes where the application will be triggered again to change their addresses, update their flowtables and send update packets to their child nodes and so on along the whole tree to get updated as shown in figure 6. As the figure show, the node ID part of the addresses of the nodes will not change. Hence, the source node will send an update packet to its tree nodes of the changed part of its address only.

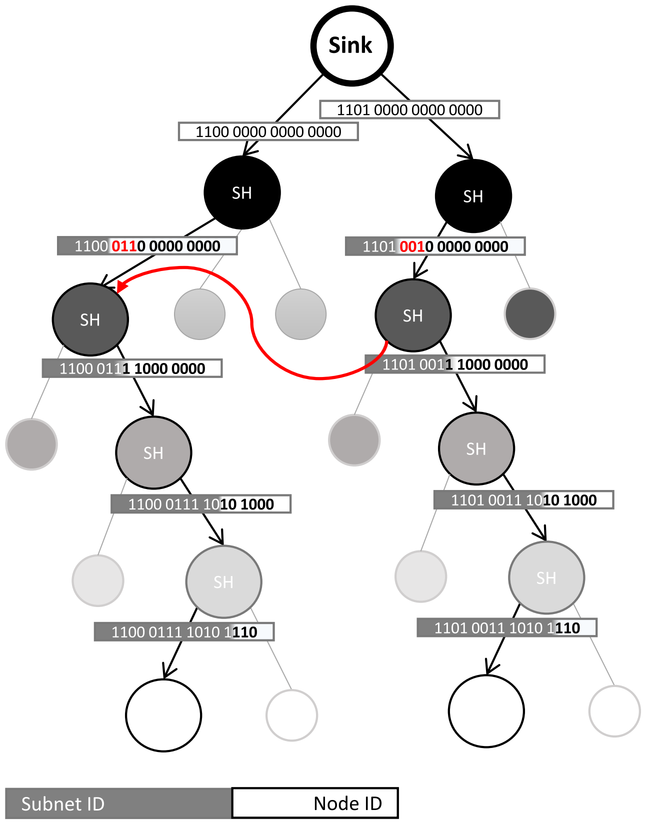


Fig. 6. Handover from a parent to a new parent during node mobility

If the new parent node can not offer a new address (its address inventory is full), HOSub will send a rejection to that request. Then the node will look for a new potential parent.

D. Handover principle Example

Figure 6 shows an example of a node reconnecting to the sink through a new parent (it could be due to loss of connection to the original parent node or because of mobility).

The sink periodically sends a hello packet with a hello packet ID. This hello packet propagates through the whole network, as the signalling diagram 7 illustrate. Every node that receive this packet will update its neighbour list and forward the packet with its address as the source address. When a link is broken between a parent and its child, the child node will miss the hello packet from its parent and start the handover process.

The handover process starts by initiating the HOSub application in the node with the missing parent. The HOSub will search for a neighbour with the least number of hops to the sink and then the highest RSSI value. Then it will send a request packet to that neighbouring node.

The HOSub application in the neighbouring node will be triggered by the request message. HOSub starts with searching for a node ID (address) to accommodate the newly joined node from its address inventory. If it finds one, it will send an update packet containing the address of the newly joined node as destination and the source address is its address as a new parent and the new address. At the same time, it will add this address to its flowtable as a child node.

The node will receive the update packet, and it will change its node ID to the new one. As shown in 6, the new Node-ID for this child is changing from 001 to 011. Then the HOSub will change the parent address in the flow table to the source address in the update packet. Its child nodes addresses in the flowtable will be changed accordingly. Then the HOSub application will send an update packet broadcast to its direct child nodes with the node's old address (1101 0010 0000 0000) as the destination. The source address will be its new address (1100 0110 0000 0000). at the same time it will send these updates to the controller to be informed of the new address. The controller will change its database and the graph network for the whole tree of that node.

The child nodes will receive this broadcasted update packet. They will keep their node IDs but change the part of their address with the source address (the new parent address of 1100 0110 0000 0000). As in the example in the figure, the child with the address 1101 0011 1000 0000 will change it to 1100 0111 1000 0000. Its node ID is staying the same (11 in this example). Then the HOSub application in the child nodes will broadcast an update packet with the same procedure as above to update their child nodes addresses until the end of the tree.

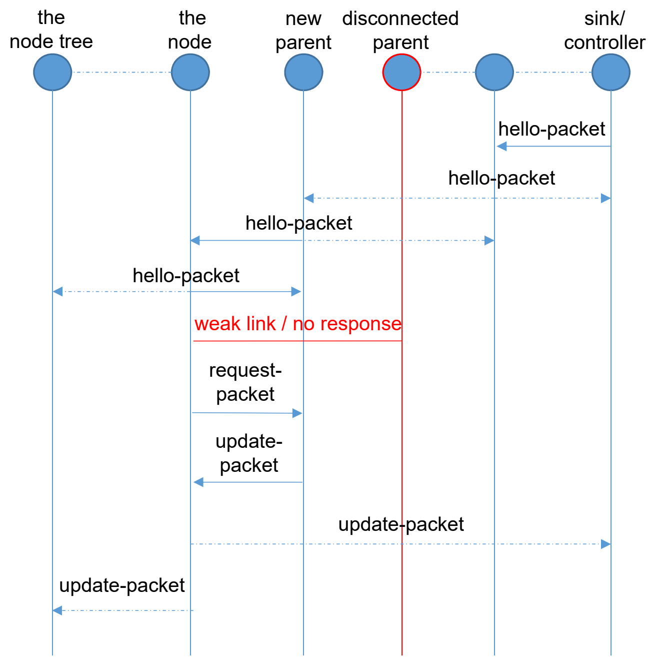


Fig. 7. Signalling during handover

V. Simulation Results and Evaluation

We used the Cooja simulator of Contiki OS to test and run our dynamic SDN and implement the controller's applications along with Sub-WSN application. We are then comparing it against SDN-WISE. Contiki is an operating system for the network with memory-constrained nodes that designed for low-power wireless system devices [13]. Cooja is an extensible Java-based network simulator that can emulate motes.

In order to create a dynamic network, the mobility plugin of Cooja was used to re-position the nodes. We used a random number generator to calculate the new coordination and fed them to Cooja through positions.dat file. Initially, the nodes are randomly positioned in the Cooja simulation area. The transmission range r for the nodes was set to 50m. The width and height l of the simulation area was set such that each node had approximately an average d of 4 neighbors nodes. We computed l according to the following equation:

$$l = \sqrt{\frac{\pi r^2}{d}.n} \tag{1}$$

where n is the number of nodes.

We are using a random waypoint model [14] (namely random walk model) as a model for the movement of the nodes in this evaluation. A randomly generated number (either 1 or -1) is added to the nodes orientations once for every step and for each of the x axis and y axis of each node in order to move each node one step up or down and left or right. In case that the resulted coordination is out of the simulation area, the new addition is being ignored.

According to SDN-WiSE approach's settings, in the first 60 seconds, no node is transmitting in order to initiate and build the network. The running time for all the simulations is set to 10 minutes (600 seconds).

A. Average over Fully and Partially Mobile Network

To evaluate the performance of this approach, we ran fifty simulated networks for 30 minutes each of five different scenarios with 10 nodes with varying numbers of total mobile nodes 20%, 40%, 60%, 80% and 100% mobile nodes plus one Sink fixed node.

This section uses a speed of 2 meters per seconds (7.2 km/h) for our evaluation test, by setting the random number generator to either 2 or -2. We chose this speed to evaluate the network in an intense environment while this speed reflects the average human jogging speed, and it is double the preferred speed for robots or machines in working places [15]. We will analyse the performance of this approach for different speeds in the next section.

Figure 8 shows the performance of the scenarios with 20%, 40%,60%,80% and 100% moving nodes through the network with our approach. As the figure shows, the more the number of moving nodes, the more is the handover. However, the figure also shows that the higher the number of moving nodes, the less the handover process increases. For example, the 100% moving nodes are closer to the 80% in terms of performance than the 80% from the 60% ones. After analysing the data in hand, it shows that it is because of the random movement of the nodes; therefore, they stay longer around their original parents. Furthermore, the fact that some parents are already moving in the same direction as their child nodes hence no need for handover updates for the child nodes as explained in section IV. That is reducing control messages, update-packets, handover time and latency. For the original approach to handling mobility, the whole network must be reinitiated. That needs, for a similar network of 10 nodes and one sink, over 5.7 Kilobytes and 70 seconds to reconnect one moving node (or disconnected link). While in our approach for the same network, the average of bytes needed for one handover is 64 bytes.

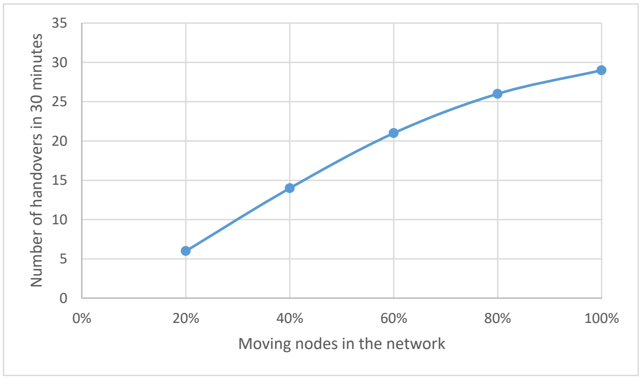


Fig. 8. Average evaluation over low and high Network mobility

B. Average over Different Network Scenarios

To evaluate the performance of this approach for different network sizes, we ran fifty simulated networks of five different scenarios with different numbers of nodes 5, 10, 15, 20 and 25 nodes plus one sink node with 100% of the nodes are moving. This section is also using the node speed of 7.2 km/h.

The number of total average handover numbers in our approach for different scenarios with different network sizes is illustrated in figure 9. And it shows that the more the nodes, the more the handovers over through the same period. Although, the figure shows that 3% of all send packets are the mobility handling packets needed for the five node network. However, in the 25 nodes network, though it is five times the area and five times the node numbers compared to the five node network, the mobility handling packet is less than 8%. That is also because of the nodes' random movement and that some parents are moving in the same direction as their child nodes; consequently, there is no need for handover updates for the child nodes. That means again reducing control messages, update-packets, handover time and latency. Furthermore, the distance between the nodes in a wide area and more extensive network allows more than one node to transmit simultaneously.

C. Average over Different Node Speed Scenarios

In this section, we are evaluating the performance of our approach by running fifty simulated networks of 10 nodes with 100% moving nodes for four different node speeds scenarios with:

- 3.6 km/h, the robots preferred speed [15].
- 5.58 km/h, the walking speed.
- 7.2 km/h, the jogging speed.
- 10.2 km/h, the average running speed.

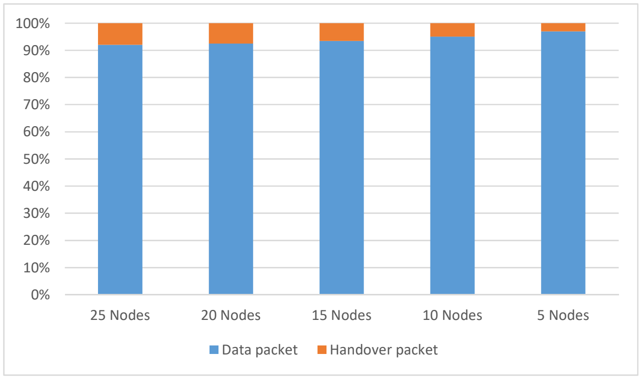


Fig. 9. Percentage of handover packet to data packet for different network sizes

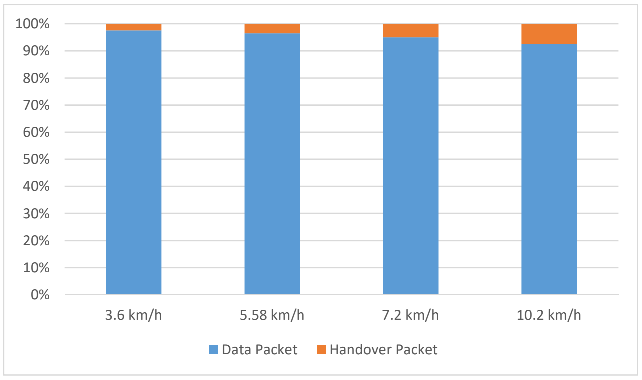


Fig. 10. Percentage of handover packet to data packet for different node speeds

Figure 10 illustrates the percentage of the number of packets needed for handover in contrast to the data packets. The speed is affecting the performance equally, as the figure reveals. The more the speed, the more are the handovers from one parent node to another. Though, it does not add to the network's performance with the higher speed of the nodes compared to the previous scenarios. However, this approach can handle mobility with a tiny percentage of packet loss 0.0083% average.

VI. Conclusions and Future Works

This paper proposes an approach over SDN-WiSE controller in a hierarchical addressed network by Sub-WSN application to handle mobility of the nodes. A handover application was implemented and deployed into mini controllers for that purpose. The result yields a message-efficient update between the nodes and the controller with a very low data loss of about 0.0083% in the studied simulation.

Mobility handling in the network is enhanced by adding applications to the controllers within the SDN-WiSE framework. This demonstrates the influence of SDN in WSN in general, though some adjustments were needed at the motes' software to deal with hello and update packets. We conclude that further research is required at the system level so that, in the future, just injecting applications into the controller of SDN-WSN platforms is feasible without the need to adjust components of the framework implementation. The nodes information in the controller can also be utilized and analyzed even further in many ways, for example, to predict the movement of the nodes by using machine learning.

References

[1] A. N. al Dulaimy and H. Frey, “Subnet addressing in software defined wireless sensor networks,” in *2019 12th IFIP Wireless and Mobile Networking Conference (WMNC)*. IEEE, 2019, pp. 32–38.

[2] L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, “SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for WIreless SEnsor networks,” *Proceedings - IEEE INFOCOM*, vol. 26, pp. 513–521, 2015.

[3] 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, no. 1, pp. 1872–1899, 2017.

[4] T. A. Al-Janabi and H. S. Al-Raweshidy, “Efficient whale optimisation algorithm-based SDN clustering for IoT focused on node density,” *2017 16th Annual Mediterranean Ad Hoc Networking Workshop, Med-Hoc-Net 2017*, 2017.

[5] ——, “Optimised clustering algorithm-based centralised architecture for load balancing in IoT network,” in *2017 International Symposium on Wireless Communication Systems (ISWCS)*. IEEE, aug 2017, pp. 269–274. [Online]. Available: http://ieeexplore.ieee.org/document/8108123/

[6] A. De Gante, M. Aslan, and A. Matrawy, “Smart wireless sensor network management based on software-defined networking,” in *Communications (QBSC), 2014 27th Biennial Symposium on*. IEEE, 2014, pp. 71–75.

[7] E. E. Tsiropoulou, S. T. Paruchuri, and J. S. Baras, “Interest, energy and physical-aware coalition formation and resource allocation in smart IoT applications,” in *2017 51st Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2017, pp. 1–6.

[8] C. R. Lin and M. Gerla, “Adaptive clustering for mobile wireless networks,” *IEEE Journal on Selected areas in Communications*, vol. 15, no. 7, pp. 1265–1275, 1997.

[9] F. Orozco-Santos, V. Sempere-Payá, T. Albero-Albero, and J. Silvestre-Blanes, “Enhancing sdn wise with slicing over tsch,” *Sensors*, vol. 21, no. 4, p. 1075, 2021.

[10] B. Trevizan de Oliveira, L. Batista Gabriel, and C. Borges Margi, “TinySDN: Enabling Multiple Controllers for Software-Defined Wireless Sensor Networks,” *IEEE Latin America Transactions*, vol. 13, no. 11, pp. 3690–3696, nov 2015. [Online]. Available: http://ieeexplore.ieee.org/document/7387950/

[11] L. L. Bello, A. Lombardo, S. Milardo, G. Patti, and M. Reno, “Software-defined networking for dynamic control of mobile industrial wireless sensor networks,” in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE, 2018, pp. 290–296.

[12] R. Kloti, V. Kotronis, and P. Smith, “Openflow: A security analysis,” in *Network Protocols (ICNP), 2013 21st IEEE International Conference on*. IEEE, 2013, pp. 1–6.

[13] A. Dunkels, B. Gronvall, and T. Voigt, “Contiki-a lightweight and flexible operating system for tiny networked sensors,” in *29th annual IEEE international conference on local computer networks*. IEEE, 2004, pp. 455–462.

[14] D. B. Johnson and D. A. Maltz, “Dynamic source routing in ad hoc wireless networks,” in *Mobile computing*. Springer, 1996, pp. 153–181.

[15] J. T. Butler and A. Agah, “Psychological effects of behavior patterns of a mobile personal robot,” *Autonomous Robots*, vol. 10, no. 2, pp. 185–202, 2001.