Embedded vs. External Controllers in Software-Defined IoT Networks

Miheer Kulkarni*, Michael Baddeley†, and Israat Haque*

*Department of Computer Science, Dalhousie University, Halifax, Canada – {miheer;israat}@dal.ca
†Secure Systems Research Center (SSRC), Technology Innovation Institute (TII), Abu Dhabi, UAE – michael@ssrc.tii.ae

2021 IEEE 7th International Conference on Network Softwarization (NetSoft) | 978-1-6654-0522-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/NetSoft51509.2021.9492688

***Abstract*—The flexible and programmable architectural model offered by Software-Defined Networking (SDN) has re-imagined modern networks. Supported by powerful hardware and high-speed communications between devices and the controller, SDN provides a means to virtualize control functionality and enable rapid network reconfiguration in response to dynamic application requirements. However, recent efforts to apply SDN's centralized control model to the Internet of Things (IoT) have identified significant challenges due to the constraints faced by embedded low-power devices and networks that reside at the IoT edge. In particular, reliance on external SDN controllers on the backbone network introduces a performance bottleneck (e.g., latency). To this end, we advocate a case for supporting Software-Defined IoT networks through the introduction of lightweight SDN controllers *directly* on the embedded hardware. We firstly explore the performance of two popular SDN implementations for IoT mesh networks, μSDN and SDN-WISE, showing the former demonstrates considerable gains over the latter. We consequently employ μSDN to conduct a study of embedded vs. external SDN controller performance. We highlight how the advantage of an embedded controller is reduced as the network scales, and quantify a point at which an external controller should be used for larger networks.**

I. Introduction

There has been a significant effort from the networking community to harness the flexibility of Software-Defined Networking (SDN) [1], which lifts the control plane from individual devices, centralizes decision making, and encourages dynamic network configuration through the provision of open APIs on commodity hardware. SDN has typically been applied to the wired and optical networks found in cloud computing use-cases, but has also found a home in wireless communications [2]. More recent efforts have tried to extend the SDN concept to the Internet of Things (IoT) networks, where SDN's programmable networking model could help support research challenges such as adapting to dynamic application requirements and providing end-to-end Quality-of-Service (QoS) guarantees for IoT communications [3]. However, Software-Defined IoT networks face a significant hurdle. Although supported by cloud and edge services, IoT networks typically consist of low-power embedded devices which face lossy wireless channels and limited device and computing resources. Furthermore, industrial applications often consist of large multi-hop wireless sensor and actuator networks with hundreds of nodes. In such scenarios, SDN's fully-centralized networking model is no longer viable: concessions and trade-offs must be made [4]. Crucially, reliance on an externally situated controller on a backbone network introduces an additional and constant overhead to SDN control messaging.

978-1-6654-0522-5/21/$31.00 ©2021 IEEE

To address this challenge, we advocate embedding an SDN controller directly within a low-power wireless network and virtualizing basic control functions on constrained devices. By removing reliance on an external controller implementation, this significantly reduces latencies between the controller and embedded SDN devices. Furthermore, modern low-power wireless chipsets are considerably more powerful than prior generations, and are more than capable of hosting simple control functions. Yet while there are a number of publicly available SDN architectures for IoT [5], [6], [7], only the Coniki-NG based μSDN [4] has (to-date) implemented an embedded controller. Furthermore, although μSDN specifies it has taken this approach, there exists no subsequent study specifically evaluating its performance against an external controller.

Our contributions. We show how an embedded SDN controller can achieve significant gains in lower bit-rate networks, and quantify the performance of an embedded vs. external controller as the network scales. We observe a 'crossover' point and highlight the scenarios where one controller outperforms the other. Specifically, as the network scales, traffic load has a significant effect on the performance of an embedded controller – negating earlier latency gains. Thus, application developers can choose embedded vs. external controller based on their demand, e.g., an embedded controller for small networks to offer low latency and resource utilization. While externally situated controllers can take care of large networks at the price of additional latency. To the best of our knowledge, we are the first to conduct an extensive performance evaluation of controllers for low-power IoT networks. We also share our code [8] for reproducibility and extension of our findings.

II. Background and Related Work

In this section, we provide necessary background on the RPL routing protocol and an overview of SDN implementations for low-power IoT networks.

RPL routing protocol. The IPv6-based Routing Protocol for Low-Power-Lossy Networks (RPL) [9] organizes the routing topology as a tree-like *Destination Oriented Directed Acyclic Graph (DODAG)* that is rooted at the sink or the data collector. Once the root node is established, RPL sends DAG Information Object (DIO) messages towards the other nodes, which carry

different routing metrics (e.g., link quality) and an objective function. Each node uses that objective function to select an appropriate parent towards the root and initiate a DAG Advertisement Object (DAO) message to advertise their location to the root. Nodes can also join the network after the topology is formed; in that case, a newly joined node can use DAG Information Solicitation (DIS) messages to solicit a DIO from neighbors. RPL can operate in a *non-storing* mode, where all messages are routed through the root node – which is the case in our work.

μSDN. It is a lightweight SDN architecture for embedded IEEE 802.15.4 devices. Specifically, the controller sends μSDN Configuration (CONF) messages a RPL DAO from joining nodes – provisioning that node's flowtable with default rules. Once configured, nodes will periodically send μSDN Network State Update (NSU) messages to the controller to infer their state information (e.g., energy level). On the data plane, μSDN exploits RPL Source Routing Headers (SRH). Specifically, it injects a desired path *directly* into matching packets (as opposed to routing through the root node) and allowing the message utilize regular IPv6 forwarding at intermediate nodes – reduce the resource consumption and control traffic. Furthermore, μSDN periodically refreshes regularly used rules to avoid repeated queries to the controller. The optimized architecture and protocol stack of μSDN makes it the state-of-the-art software-defined solution for low-power IoT networks.

SDN-WISE. In contrast, the SDN-WISE [5] extends the OpenFlow protocol for sensor networks. Unlike traditional stateless data plane devices, SDN-WISE sensor nodes maintain three data structures: WISE state arrays, accepted ID arrays, and WISE flow table. The former maintains some state information; the latter two support a chosen set of packet processing and match-actions rules. The protocol stack at each sensor consists of Topology Discovery and Maintenance (TDM), Packet Forwarding (PF), In-Network Packet Processing (INPP), and Application layers. The first two layers construct and maintain the routing topology and the flow-based packet forwarding, whereas INPP allows a sensor to perform In-Network Packet Processing. The Adaption layer provides an API between a sensor and the controller. However, the stateful SDN-WISE design does not offer support for IPv6 and RPL in low-power IoT networks.

A. Related work

Aside from μSDN and SDN-WISE, there is now a considerable body of literature examining SDN in IoT.

SD-WISE [10] facilitates the heterogeneity characteristic of IoT by leveraging the Network Operating System (NOS), i.e., integrating ONOS controller. SDSense [11] delegates some network functions (e.g., neighbor maintenance) to sensors, while the controller takes care of topology discovery and maintenance, etc., that require a global network view. It also provides reliability by adopting the routing topology from [12], [13]. Another energy-aware software-defined WSN design is presented in [14]. However, none of these solutions is compatible with RPL and IPv6. CORAL-SDN [6] proposes an RPL-based topology discovery and maintenance. Recent work in [15] is also built on RPL and Contiki OS to support IPV6 and Network Function Virtualization (NFV), where virtualized data aggregation is optimally deployed in some IoT devices to reduce energy consumption. *Crucially, none of these works examine the effect of controller placement (i.e., external vs. embedded) on SDN performance in low-power IoT networks.*

III. Design and Architecture

We compare μSDN and SDN-WISE, two open-source SDN implementations for low-power IoT mesh networks. These implementations each take a different approach to the SDN controller, with μSDN embedding the controller directly in the mesh while SDN-WISE takes a more traditional route of placing the controller externally on the backend network. In theory, an embedded controller pushed directly onto the mesh should exhibit lower latencies for SDN control messages, while a more powerful controller situated on the backend network should be capable of scaling as the mesh grows. In this regard, to ensure fair comparison between the two architectures, we extend SDN-WISE to support an embedded controller, while we likewise extend μSDN to support an external ONOS controller.

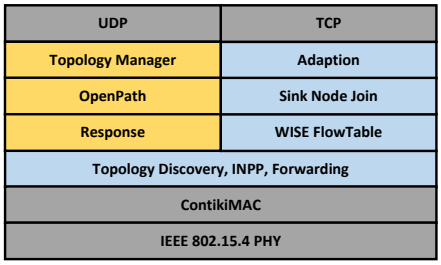


Fig. 1: Modified SDN-WISE architecture. Yellow components show embedded controller modules, and blue ones are the original SDN-WISE node architecture.

A. Integrating Embedded Controller in SDN-WISE Architecture

μSDN introduces its own lightweight embedded controller, μSDN-Atom [3], which extends the capabilities of a regular low-power node with some limited SDN controller capabilities. μSDN-Atom control nodes are capable of receiving flow requests and responding with flowtable instructions to/from all nodes within its Layer-3 network topology (using RPL), as well as collecting periodic status updates (link quality, neighbors, etc.). This control signaling between the μSDN-Atom controller and μSDN nodes within the IoT mesh network are summarized in Table I. Since the controller is within the network this allows μSDN-Atom to respond to requests quicker than external controller. In contrast, SDN-WISE situates its controller on a backend network outside of the IoT mesh [5] and has recently been extended to support the ONOS SDN Controller [10].

We firstly port SDN-WISE's external controller logic to an embedded approach similar to μSDN. Specifically, we support the controller functions *Topology Manager*, *OpenPath*, and *Response* within a standard SDN-WISE low-power node. As in μSDN, these are implemented in the sink node. The *Topology*

Signaling	Description
Flowtable Query (FTQ)	Request controller instruction
Flowtable Set (FTS)	Install rule on node
Network State Update (NSU)	Report state to controller
Configuration (CONF)	Initialize joining node

TABLE I: μSDN controller $\leftrightarrow$ node control signaling

Signaling	Description
Config	Initialize joining node
OpenPath	Collect state information from node
Request	Request controller instruction
Response	Install rule on node
Report	Report state to controller

TABLE II: SDN-WISE controller $\leftrightarrow$ node control signaling

Manager collects state information (e.g, energy, node position, neighbors) from SDN-WISE nodes to construct and maintain the routing topology. The *OpenPath* module helps the controller initiate control packets to those nodes to collect network state information and update the network state view. Finally, the controller uses the *Response* module to send the packets containing flow rules for requesting nodes. Fig. 1 shows the modified architecture for SDN-WISE with embedded control logic, while Table II lists the communication messages between the SDN-WISE controller and nodes.

B. Integrating ONOS Controller to μSDN Architecture

Likewise, we extend the μSDN stack to support an external SDN controller ONOS – creating ONOS components to support μSDN control functions externally rather than on the embedded μSDN-Atom controller.

Adaption of ONOS Subsystems. A number of different ONOS components were developed. Fig. 2 shows rectangular components designed as per the μSDN-Atom controller functionality and hatched ones extended to support the μSDN message format. Each subsystem is marked in the same color, while the arrows demarcate the directions in which messages travel within the ONOS controller.

SensorNode Subsystem – when ONOS receives a message at the protocol layer, an event is triggered that notifies the SensorNode provider that a node has joined. This information is then propagated to the SensorNode manager that maintains node information (e.g., NodeID). The SensorNode API offers access to this information. *FlowRule Subsystem* – FTQ messages from μSDN nodes are sent to the FlowRule Provider. This wraps the FTQ in the ONOS format. The corresponding FTS message is reformatted back to the μSDN protocol and sent back to the mesh. The FlowRule API is designed to support this format conversion. *Packet Subsystem* – ONOS needs to maintain topology and relies on μSDN NSU messages to achieve this. The Packet subsystem is designed for the Provider layer that receives NSUs and generates the corresponding CONF messages. The Packet provider also offers the format conversion between ONOS and μSDN before passing them to an appropriate subsystem. *DeviceControl Subsystem* – depending on the required device configuration for the underlying low-power sensors, this subsystem generates appropriate messages like the device on/off. These messages are then converted to the right format while they travel between the ONOS and μSDN layers.

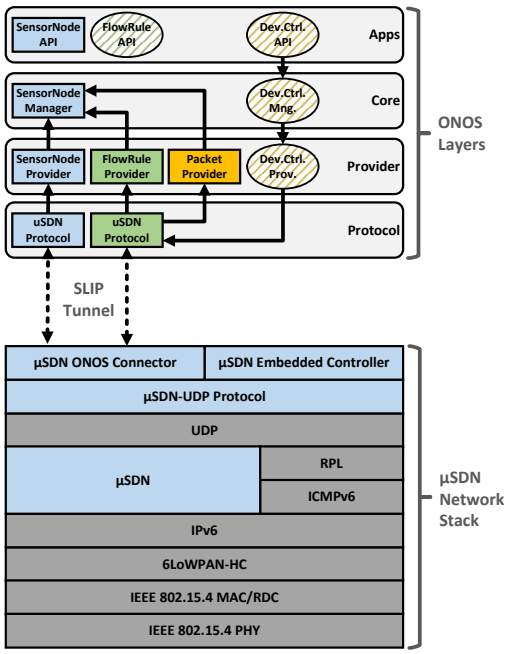


Fig. 2: Extension of μSDN to the ONOS controller architecture.

Workflow of the μSDN-ONOS Architecture. μSDN traffic is now handled by ONOS in three stages. *Controller Discovery and Join* – firstly, the μSDN sink receives DAOs from nodes that have joined the RPL DAG and forwards these to ONOS. These flow to the SensorNode subsystem that stores and maintains the topology. The Packet subsystem then generates a CONF message to set the nodes with flowtable rules, timers, etc. After converting to μSDN format, these are sent back to the sink to forward over RPL. *Node Updates* – once configuration is complete, nodes are ready to communicate, i.e., convey data plane traffic to the sink. However, the ONOS controller needs to know the current network state. Accordingly, μSDN nodes periodically send NSUs with configured metrics (energy, neighbors, etc.) to the controller. The Packet subsystem receives these NSUs and updates the topology in the SensorNode subsystem. The DeviceControl subsystem actively keeps track of these metrics and sets priorities on the information that the controller must aware of. *Controller Response* – in addition to the periodic control message sharing, nodes also need to generate on-demand control traffic. For example, μSDN nodes send an FTQ message to the controller if it misses a flow-rule entry. The FTQ message is handled by the FlowRule subsystem, where the FlowRule provider generates an FTS message that contains appropriate flow rules for the requesting node.

IV. Evaluation Setup

In this section, we outline the evaluation setup. Our experiments run on a machine with a 3.8 GHz 6 core CPU and 16 GB RAM equipped with Cooja 2.7 simulator for Contiki OS. μSDN nodes consist of a MSP430F5438 platform with a CC2420 radio. SDN-WISE nodes consist of EMB-Z2530PA

based sensor nodes. Table III describes protocol stacks of μSDN and SDN-WISE used in the evaluation. In both cases, the nodes form 50 node grid or random topologies, where the sink node has the embedded customized SDN controller for low-power IoT networks.

Layer	μSDN	SDN-WISE
Control	Embedded	Embedded
Network	IPv6 + RPL	TD INPP Forwarding
MAC	CSMA/CA	CSMA/CA
RDC	Contiki MAC	Contiki MAC
PHY	IEEE 802.15.4	IEEE 802.15.4

TABLE III: μSDN and SDN-WISE protocol stacks

We also connect an ONOS controller to the μSDN architecture to compare the performance of the external and embedded controllers. In this setup, we use a SLIP connection between the sink and ONOS. The sink node acts as a border router to facilitate the connection between μSDN nodes and the ONOS controller. We repeat each evaluation 50 times to take the average with a 95% confidence interval. Table IV presents various parameters and their settings in the Cooja simulator that we use in the evaluation.

Parameter	Value
Duration	5 min
Transmission Range	50m
Interference Range	50m
Max Bit rate	9 bit/sec
Max Flow request rate	102 flows/sec
Link Quality	90%
Radio Medium	UDGM (distance loss)
RPL mode	Non-storing mode
RPL Route Lifetime	5min
Flowtable Lifetime	5min

TABLE IV: List of Cooja parameters used in the evaluation.

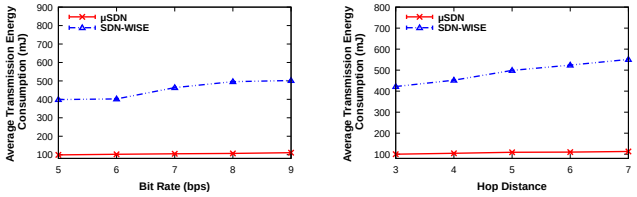
V. Results Discussion

This section first presents a detailed discussion on the μSDN and SDN-WISE architecture comparison results. Then, we discuss embedded and external controller comparison.

A. Node Architecture Comparison

As part of this comparison, we study the communication energy consumption, delay, and packet delivery ratio (PDR) over varying load and route length to infer the superiority of one architecture over the other.

Communication energy consumption. The average energy consumption from Fig. 3 depicts that μSDN has a significantly lower consumption compared to SDN-WISE. μSDN is built on the standard IEEE 802.15.4 low-power IoT protocol stack and (as previously describes) contains optimizations for SDN control traffic. Also, μSDN is a stateless architecture, where relay nodes do not maintain any routing tables. Finally, it deploys interference-aware routing that reduces the number of re-transmissions. All these features contribute to the significant energy savings of μSDN, which is not the case for the stateful SDN-WISE. We observe a similar performance trend



(a) Bit Rate (b) Hop Distance

Fig. 3: Mean energy consumption in a grid topology.

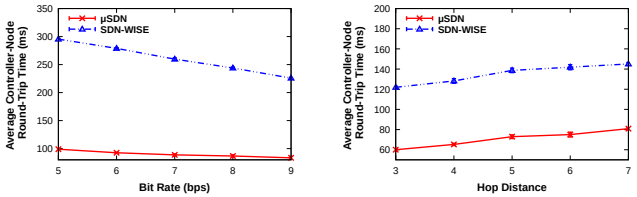


Fig. 4: Mean controller-node delay in a grid topology.

with varying load and path length, where route length has more impact on SDN-WISE. The behavior that again can be explained as the lack of interference-aware packet forwarding.

Flow-rule installation time. As seen in Fig. 4, μSDN shows better performance when we measure the node-to-controller round-trip time of flow rule installation requests. μSDN has better performance due to using source routing at the relays. Their number is usually much higher compared to sources. However, relays in SDN-WISE also maintain flow tables that impact the rule installation time. As the bit rate increases, we see a decrease in that time as packets reach destinations quickly. Nonetheless, the time increases with the increasing hop-distance due to the chance of interference and additional requests to the controller.

Packet reliability. The average Packet Delivery Ratio (PDR) for both architectures is presented in Fig.5. μSDN has a higher PDR than SDN-WISE, as expected. The optimization across the protocol stack, interference-aware source routing all contribute to this performance of μSDN. For example, it forwards packets over alternative routes in the presence of interference instead of packet drop or retransmission in SDN-WISE. Thus, building the μSDN architecture on top of a standard protocol stack with the required optimization for software-defined networks brings considerable advantages. In the following, we measure its performance with an embedded (part of the IoT networks) and standard external controller (ONOS) to find the appropriate controller choice for the low-power IoT networks.

B. Controller Comparison

This section compares the performance of embedded (μSDN) and external (ONOS) controllers considering standard evaluation metrics like throughput, latency, topology discovery, and update time. In the rest of this section, we present the results for the grid topology as the performance trend is similar on the random one.

Throughput and latency measurement. Our goal in this evaluation is to check the performance of the chosen controllers in terms of handling a varying number of flow-rule installation

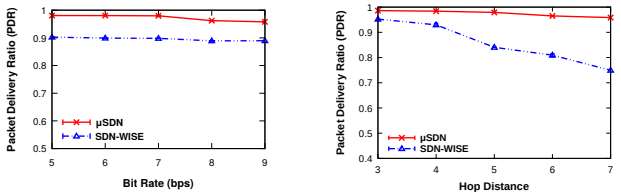


Fig. 5: Mean PDR in a grid topology.

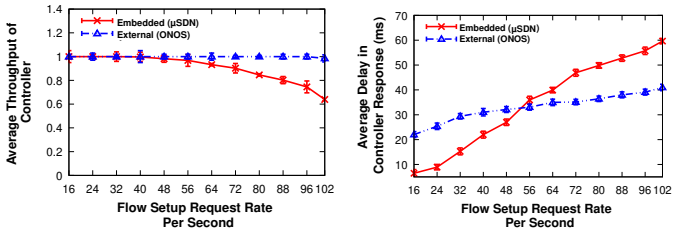


Fig. 6: Mean throughput and latency.

requests per unit time from all sensor nodes, i.e., we consider a highly loaded network. The throughput is presented in Fig. 6(a). We observe an interesting performance trend. Both controllers have the same throughput for up to 50 requests per unit time. The performance of the embedded μSDN Atom controller starts degrading after that point with the increasing number of requests. The standard distributed ONOS controller's processing capability takes the lead under high load, which is not the case for the domain-specific embedded one. We observe a similar performance trend in the case of latency measurement. μSDN 's embedded controller latency in Fig. 6(b) is significantly better than the external ONOS controller for a request rate up to 50. After that point, it shows the opposite trend compared to ONOS. The embedded controller can process all requests within its resource budgets and get back to the nodes quickly at a low rate. However, ONOS suffers from the communication delay as it is external to the network. As the rate increases, communication latency is compensated by the processing capability of ONOS, where the embedded controller falls back.

Topology discovery and update time. We measure the topology discovery time for small to large topologies, where ONOS takes a constant amount of additional time compared to the embedded controller. This result is expected as the embedded one is part of the IoT network and works on the same protocol stack that resides in the IoT nodes. However, the gateway or sink node connects to the external ONOS controller that furthermore requires protocol compatibility. Note that due to the space limitation we are not presenting the discovery time comparison. Topology updates are required because of a change in the network due to the failure or unavailability of nodes or links. In this experiment, we randomly fail a fixed number of links from the network for varying sources and measure the topology update time in two settings: with and without data traffic flowing in the network. In both cases, we observe a performance trend (Fig. 7(a)) similar to that of the latency, i.e., the embedded controller takes less time while the number of sources is small. As this number increases, ONOS

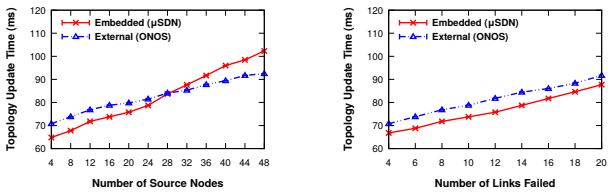


Fig. 7: Mean topology update time for source and link failures.

takes the lead over the embedded one. One the other hand, when we vary the number of failed links for a given number of sources in Fig. 7(b), we observe that the update time increases with the increasing number of failed links. Thus, the number of sources or the network size has the most significant impact on the performance of the controllers.

VI. Conclusions

This paper has performed an extensive evaluation of two SOTA low-power IoT network architectures: SDN-WISE and μSDN. We have found that the control-layer optimizations present in the IEEE 802.15.4 and IPv6 compliant μSDN allow it to outperform SDN-WISE across energy, latency, and reliability metrics. However, we have shown that while the embedded controller approach in μSDN initially demonstrates significant performance advantages, as the network scales this advantage is reduced. Crucially, we can quantify a 'crossover point' of around 50 nodes / requests-per-second at which an external controller with greater resources (such as ONOS) is better positioned to handle larger or more demanding networks: thus a controller should be chosen based on applications' QoS.

References

[1] E. Haleplidis *et al.*, "Software-Defined Networking (SDN): Layers and Architecture Terminology," RFC 7426, Jan. 2015.

[2] I. Haque *et al.*, "Wireless Software Defined Networking: A Survey and Taxonomy," *IEEE Comm. Surveys and Tutorials*, May 2016.

[3] M. Baddeley, "Software Defined Networking for the Industrial Internet of Things," Ph.D. dissertation, Univ. of Bristol, UK, 2020.

[4] M. Baddeley *et al.*, "Evolving SDN for Low-Power IoT Networks," in *IEEE NetSoft*, Jun. 2018.

[5] L. Galluccio *et al.*, "SDN-WISE: Design, Prototyping and Experimentation of a Stateful SDN Solution for WIreless SEnsor Networks," in *IEEE INFOCOM*, May 2015.

[6] T. Theodorou *et al.*, "CORAL-SDN: A Software-Defined Networking Solution for the Internet of Things," in *IEEE NFV-SDN*, May 2017.

[7] R. Alves *et al.*, "IT-SDN: Improved Architecture for SDWSN," in *Symp. on Comp. Networks and Dist. Sys.*, 2017.

[8] M. Kulkarni *et al.*, "Performance Evaluation of Controller for Low-power IoT Networks," 2020, [Online] https://github.com/MiheerKulkarni/Controller-Compare-Thesis.

[9] T. Winter *et al.*, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," RFC 6550, Mar. 2012.

[10] A. Anadiotis *et al.*, "Towards a Software-Defined Network Operating System for the IoT," in *IEEE INFOCOM*, May 2015.

[11] I. Haque *et al.*, "SDSense: An Agile and Flexible SDN-Based Framework for Wireless Sensor Networks," *IEEE Trans. on Vehicular Technology*, February 2019.

[12] I. Haque and C. Assi, "Olear: Optimal localized energy aware routing in mobile ad hoc networks," in *IEEE ICC*, 2006.

[13] I. Haque *et al.*, "On Selecting a Reliable Topology in Wireless Sensor Networks," in *IEEE ICC*, 2015.

[14] Z. Ding *et al.*, "An Interference-Aware Energy-Efficient Routing Algorithm with Quality of Service Requirements for Software-Defined WSNs," *IET Communications*, 2019.

[15] D. Saha *et al.*, "An Energy-Aware SDN/NFV architecture for the internet of things," in *IFIP Networking*, Paris, France, 2020.