Journal of Computer Engineering
ISSN 10003428

Original Research Article
Vol 14, Issue 12, 2025 | ISSN: 1000-3428

Proactive Flow Installation Mechanism for Delay Minimization in Software-Defined Networking for IoT Systems

Dr. Elena Vasquez-Perez, Dr. Javier Martinez-Gonzalez, Dr. Nikolaos Koutsourelis

Department of Computer Science, University of California, Los Angeles (UCLA), Los Angeles, CA, USA; Department of Electrical and Computer Engineering, National Technical University of Athens (NTUA), Athens, Greece

***Abstract*—In Software Defined Networks (SDN) a Controller installs forwarding rules or queries the status of network devices through a separate control channel using the OpenFlow protocol. Such queries may ask for instantaneous metrics, like port status, packet error rate, or transmitted packets. In turn, the Controller uses these metrics, or context, to condition forwarding rules. As network conditions change over time, forwarding rules are set to expire if not used (or matched) during a configurable idle timeout. Consequently, they are erased from the flow table, forcing a query to the Controller every time a packet for which there is no forwarding rule arrives at the network device. This work presents experimental results showing the delay increase produced by timing out forwarding rules of periodic flows, such as the generated by sensors in IoT deployments. Furthermore, it proposes and experimentally evaluates a pro-active forwarding rule installation mechanism able to reduce such delay by 90%, easing the implementation of context-aware forwarding strategies for IoT.**

I. Introduction

The concept of Internet of Things (IoT) refers to the connection of islands of sensors and actuators networks to the Internet, enabling remote monitoring but also remote actuation, event generation, and remote configuration of distributed appliances.

These networks may be conceived using different communications technologies in order to increase the network lifetime, redundancy, or to provide Quality of Service (QoS) for sensitive traffic. Wireless Sensor Networks (WSN) are one of the key technologies enabling IoT. Usually, these networks are composed of small devices equipped with a radio transceiver and sensors that include, but are not limited to, temperature, noise, humidity, CO_2 concentration, among others.

Sensor nodes are specially constrained in terms of processing power and battery, so are often configured to turn on the transceiver antenna and transmit sensor data periodically. The configuration of this period, or *duty cycle* generally varies according to the sensor type. For instance, it might make little sense to report the temperature of a room every 0.5 s; instead, it will be more energy efficient to set the duty cycle to a higher value. On the other hand, sensor nodes may also be configured to trigger additional transmissions once an event is detected, e.g.: a seismic sensor, forest fire detection sensor.

Sensor nodes (or other *things*) can be connected to IP networks by employing standards like 6LoWPAN [1]. This promises to increase the granularity of data acquisition and remote network management. Furthermore, it allows the design of Software Defined Networks (SDN) at the Gateway level, enabling features like gathering network-wide state information, context-aware forwarding strategies, redundancy, selective load-balancing, and anomaly detection.

SDN decouples control and data forwarding tasks from the network device (switch, router, or Gateway), placing the control or *intelligence* of the whole network at a centralized SDN Controller. Network devices rely on the Controller to provide flow rules and instructions for how to treat incoming flows. For instance, when a packet from a sensor node arrives at the SDNcontrolled Gateway for which there is no entry in the flow table, it queries the SDN Controller using Open Networking Foundation's (ONF) OpenFlow protocol [2] through a separate control channel. If a solution is found, the Controller installs forwarding instructions in form of a flow rule entry on the Gateway's flow table. This way, subsequent packets of the same flow[1] are forwarded through the data plane without further aid from the Controller. SDN is thought to be another key enabler of IoT

1

How to cite: Dr. Elena Vasquez-Perez, Dr. Javier Martinez-Gonzalez, Dr. Nikolaos Koutsourelis (2025). Proactive Flow Installation Mechanism for Delay Minimization in Software-Defined Networking for IoT Systems. Journal of Computer Engineering, 14(12), 40–49. DOI: https://doi.org/10.5281/zenodo.19359479

applications, particularly relevant for orchestration of Gateway resources in multi-network environments.

Within the SDN paradigm, network devices will keep getting cheaper and simpler, mainly because they are devoid of complex logic [3], specially in SDN IoT [4]. Moreover, as information from the whole network can be gathered at the Controller, forwarding decisions may also rely on the network's current condition. That is, output ports or paths can be chosen according to instantaneous metrics from the whole network, such as congestion, error rate of a particular port/path, or flow table size. This is referred to as context-aware forwarding.

Context-aware forwarding strategies can be specially beneficial in radio-heterogeneous networks, such as those composed of Gateways in a SDN IoT scenario. These devices could be equipped with several wired and wireless interfaces in the uplink in order to: enforce redundancy, provide means for traffic differentiation, or to offer QoS.

To take advantage of all the aforementioned benefits, though, flow tables need to be modified constantly. This is usually enforced configuring idle timeouts on each flow rule, so if a particular entry is

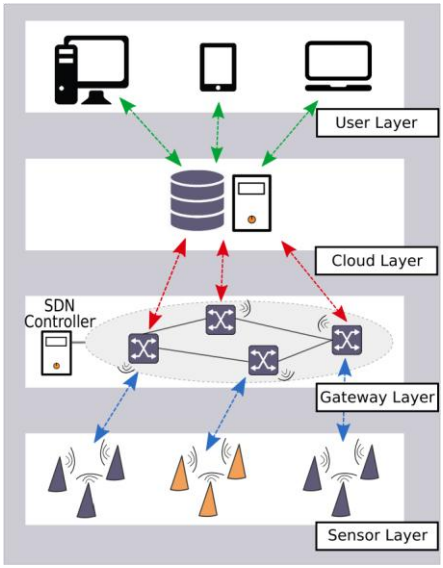

Fig. 1. An example end-to-end IoT platform architecture.

not used (matched) during a certain time period it is automatically erased from the table. Nevertheless, the effects of this measure suggest an increase in the message exchange between Gateways and Controller, additional delays over periodic sensor traffic, or even packet loss due to control channel congestion.

The contributions of this work are the result of experimental evaluations on a testbed of IoT Gateways using Commercial Off-The-Shelf (COTS) hardware, and real traffic from sensor nodes. They can be summarized as follows:

- Implements a SDN pro-active flow installation mechanism using real hardware.
- Proposes extensions to the aforementioned mechanism to:
  - Support sensor nodes with arbitrary duty cycles.
  - Detect and forget periodic flows based on flow rule matches.
- Provides experimental results showing a reduction in flow rule installation messages compared to the reactive approach.
- Achieves a flow rule installation delay reduction of around 90%.

An overview of SDN flow installation procedures is contained in Section II. Section III proposes a proactive flow installation mechanism, while details of the experimental setup and results are contained in Section IV. Finally conclusions are drawn in Section V.

II. Related Work

IoT keeps pushing for IP connected things using protocols such as 6LoWPAN. In turn, Gateway devices may compose a SDN that would carry traffic from an underlying sensor network to upper layers of an IoT architecture (as the one shown in Figure 1).

This section overviews the OpenFlow protocol, a popular Open Networking Foundation (ONF) standard interface SDN Controllers use to interact with the

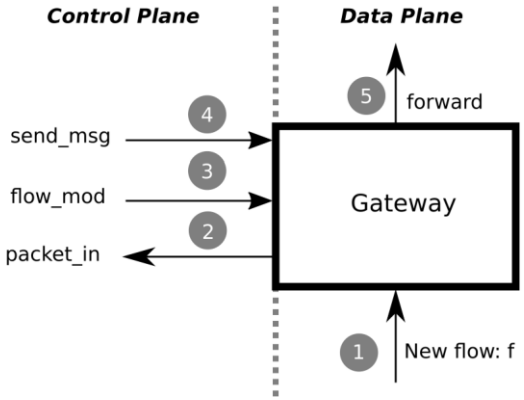

Fig. 2. OpenFlow message exchange between a Gateway and a

SDN Controller upon arrival of a packet for which there is not rule in the flow table

forwarding table of network devices. Next, it briefly describes the role of a SDN Controller during flow rule installations. The section ends giving a description of the reactive flow rule installation delays, and how the

associated problems could be mitigated with a SDN application.

A. OpenFlow

OpenFlow (OF) [2] is one of the first SDN standards to define a SDN Controller communication protocol for interacting with network devices' forwarding plane. Instead of hoping for hardware vendors to provide an open and programmable interface to their routers and switches (which may threaten companies with undesired competition), OpenFlow exploits a common set of functions related to flow tables, simply requiring a minimum set of actions[2].

A SDN device relies on the SDN Controller for updating its flow table and to determine all forwarding decisions. This is done exclusively using protocols such as OpenFlow messages through a separate control channel.

B. OpenFlow Controller

It is a platform upon which SDN applications are developed. Generally, SDN applications[3] control the data plane of network devices, performing actions like adding, modifying, and removing flow entries employing OpenFlow messages. They may also be stacked and designed to work together. For instance, a set of centralized SDN Controller applications may use instant network-wide metrics like port type, queue length, or flow statistics from every forwarding device as inputs to an algorithm that computes energy-efficient paths. Figure 2 shows a simple example of a reactive flow installation process (occurring through a control channel) between a Gateway that receives the first packet from a flow (head of flow) that is not contained in the local flow table, and the Controller[4]. As shown in the figure, after reception of the unmatched head of flow (step 1 in Figure 2), the Gateway queries the Controller using OpenFlow's packet_in message (step 2), which may include a copy of the original head of flow. If a forwarding rule is found for such flow, the SDN application running at the Controller returns a flow rule using an OF flow_mod message (step 3). Next, it instructs the Gateway to forward the buffered head of flow according to actions specified in an OF send_msg message (step 4). After the transmission of the head of flow (step 5) other packets from the same matched flow are forwarded following the newly installed flow rule, without further exchange with the Controller.

C. Problem Statement

Apart from the enabling functionality that SDN brings to IoT Gateways, it also proposes additional challenges. Particularly, the reactive flow rule installation process increases the delay of the head of flow, which may be unacceptable for delay critical applications such as the envisioned factories of the future [5]. Furthermore, [6] shows that under high flow arrival rates (r flows/second), the network device's OpenFlow Agent is sometimes unable to generate packet_in messages for all flows. This results in packet losses due to control channel congestion, wasting precious energy resources from sensor nodes and potentially reducing the lifetime of the WSN in an IoT SDN scenario.

Interestingly, the particular case of periodic traffic can be protected from control channel congestion issues if a Controller application is be able to:

1) Identify periodic flows by learning their respective duty cycles.
2) Install flow rules before the expected arrival of the head of flow (with an idle timeout, t_{idle}, lower than the duty cycle).

This way Gateways will be prepared to forward the expected traffic, effectively changing from a reactive to a pro-active flow installation procedure. Also, as no packet_in messages are expected for already learned periodic flows, the number of OpenFlow messages through the control channel is reduced to only flow_mod messages (a reduction from 3, to 1 OpenFlow message per head of flow).

Finally, because t_{idle} is configured according to the duty cycle of each flow, each pro-active rule installation may also consider the most recent context information at the Controller, opening the possibility for the computation of energy-efficient paths in dynamic network settings, or QoS.

III. Pro-active flow installations to reduce delay and control channel congestion

The Pre-emptive Flow Installation Mechanism (PFIM) [7] is an SDN application running at the Controller. It defines a local PFIM table to store time instants of packet_in message receptions in order to determine the periodicity of a flow. If a determined flow is found to be periodic, PFIM generates the appropriate flow_mod message and sends it to the appropriate Gateways before the next arrival of the periodic flow.

PFIM is able to reduce control channel traffic by learning the duty cycle of periodic flows and installing flow rules in a pre-emptive manner. Further, it also eliminates the delay experienced by the head of flow during the flow installation procedure. Nevertheless, as conceived in [7], it heuristically disregards a flow as non-periodic after 15 pre-emptive flow installations are executed. This measure fails to bring PFIM

[4] The workflow is implemented as a SDN application.

benefits to flows that change between periodic and bursty, like the produced by sensor nodes reporting periodic and event triggered metrics.

A. Extending PFIM

To allow a diversity of duty cycles as well as eventgenerated traffic from sensors, this work proposes two Conditions for PFIM to forget the periodicity of a specific flow (or what is the same as erasing the flow from the PFIM table):

1) Condition 1: if a packet_in arrives requesting a flow rule that is already scheduled to be installed.
2) Condition 2: if a rule that was installed following PFIM is not matched before it is erased (due to t_{idle}).

For Condition 1, a packet_in message at the Controller suggest that the Gateway had no flow rule installed for that specific flow. Follows directly that if the same flow is scheduled for a proactive flow rule installation, then its periodicity must have been broken. Consequently, the flow is removed from the PFIM table.

Condition 2 is enforced by turning on the OFPFF_SEND_FLOW_REM flag for flow rules of periodic traffic. This forces the network device to notify the Controller via a flow_removed OpenFlow message every time it erases a flow rule[3]. flow_removed messages carry information about the flow rule, such as the number of packets matched.

B. Extended PFIM Workflow

The following algorithms show the way PFIM and its extensions (or Extended PFIM) work. Two SDN applications work together for Registering incoming OpenFlow messages, and Learning the periods of transmissions. The Registering phase (shown in Algorithm 1) takes care of incoming OpenFlow messages, as well as filling or erasing entries from the PFIM table. The Learning phase (see Algorithm 2) is always looking for periodic transmissions registered in the PFIM table. PFIM extensions appear highlighted on Algorithm 1.

Starting with the Registering phase in Algorithm 1, upon reception of a packet_in message at the Controller, PFIM creates a unique flow identifier (Line 4).

Algorithm 1: PFIM running at a SDN Controller: Registering phase

```
1 new PFIM_table;
2 new timer(t, f); // installs rule at t for flow f
3     if packet_in then
4         flow = hash(packet_in);
5         if flow not in PFIM_table then
6             PFIM_table.create_list_at(flow);
7         else
8             if flow in timer.pending() then
9                 PFIM_table.remove(flow);
10            timer.set(current_time, flow);
11            PFIM_table[flow].append(current_time
              );
12        else if flow_removed then
13            flow = flow_removed.getFlow();
14            if hash(flow) in PFIM_table then
15                if flow.packet_count() == 0 then
16                    PFIM_table.remove(hash(flow));
```

Then, if the flow identifier is not found in the PFIM table, it is included alongside an empty list. This list will be filled with the current_time with every packet_in message of the same flow. On the other hand, if the flow is already registered in the PFIM table and is pending for a flow rule (Line 8), it is removed. After either of the cases a reactive flow rule installation ensues (as specified in Condition 1, see Line 10 in Algorithm 1).

On the alternative case, when a flow_removed is received, PFIM looks for the number of packets forwarded using the expired flow rule. If it was not used (Line 15) it is removed from the PFIM table (as specified in Condition 2).

Extended PFIM learns the period of transmissions after i packet_in messages are received for a registered flow. As shown in Algorithm 2, PFIM is always navigating the PFIM table and counting the number of packet_in entries per flow (Line 10). If a flow is found to have sufficient entries (specified by i), it estimates a period for scheduling a proactive flow rule installation (see Lines 13-15 in Algorithm 2). IV. Experimental Results

A. SDN Controller selection

The popularization of OpenFlow and SDN has led to the creation of a wide variety of Controllers [8]. Focusing on open source alternatives, the main difference among them relates to the programming language and multi-threading support. These two aspects have a direct impact on performance, and therefore on the suitability for production networks [4].

[3] [4]

Fast performing controllers like Trema [12], FloodLight [13], and OpenDaylight [14] all support Open-
Flow version 1.0, but lack the ease of implementation

Algorithm 2: PFIM running at a SDN Controller: Learning phase

```
1 include PFIM_table; // Ref. to Algorithm 1 table 2 define i;
// max. num. of entries 3 define λ; // std. threshold 4 define α; //
pre-empt value 5 new μ; // average of entries 6 new σ; // standard
deviation of entries 7 new timer(t,f); // installs rule at t for flow f
8 for flow in PFIM_table do
9     list = PFIM_table[flow];
10    if list.size() > i then
11        μ = average(list);
12        σ = std(list);
13        if σ < λ then
14            next_installation = μ - α;
15            timer.set(next_installation, flow);
```

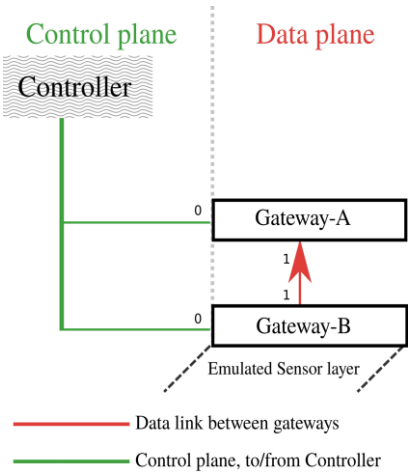

Fig. 3. Topology of the experimental setup. Numbers beside data plane links are used for identification purposes only. Gateways access the control channel through interface 0.

provided by the Python-based POX [10] and Ryu [11]; which in turn do not scale through multiple threads. As this work implements new functionality, the selected Controller is the one able to:

- Provide interfaces for application-Controller interaction.
- Include comprehensible documentation supported by a community.
- Allow fast prototyping by employing high level programming languages and abstractions.
- Is block/module based, so SDN applications can be reused or integrated with others.
- Supports most of OpenFlow's optional features.

As this work shares the same requirements as [9], Ryu has been selected as the SDN Controller for the work presented in this paper.

B. Testbed description

In order to implement Extended PFIM, a real SDN IoT Gateway layer was built using COTS devices, such as the Raspberry Pis 3 model B. Open vSwitch

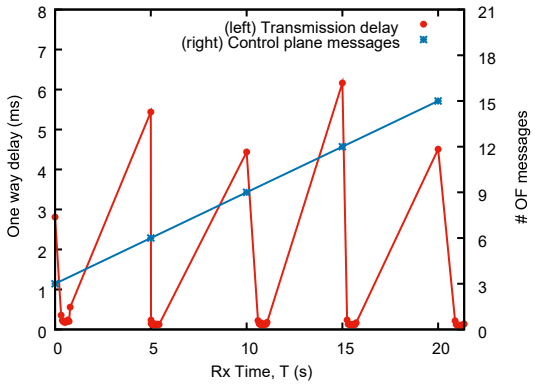

Fig. 4. Reactive flow rule installation with Periodic traffic. (lefty) One way delay, (right-y) accumulated number of OF messages related to the flow rule installation procedure.

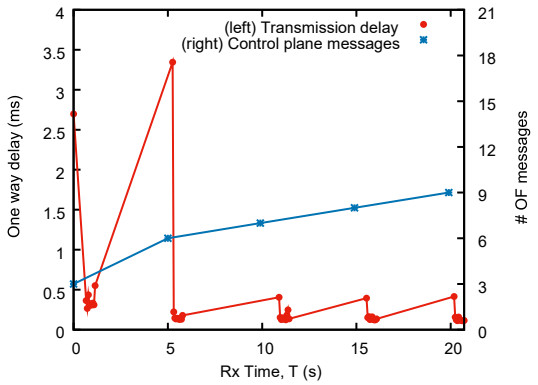

Fig. 5. PFIM pro-active flow rule installations for Periodic traffic. After *i* entries in the PFIM table, periodicity is detected and rules are pro-actively installed.

(OVS) version 2.7.0 provides the OpenFlow Agent to Gateways. Traffic from real sensor nodes[5] is set to follow the parameters specified in Table I. Ryu v4.4 SDN Controller (using OpenFlow v1.4) runs an Extended PFIM implementation on a PC with Ubuntu
16.04[6].

Table I also provides PFIM parameters used in the experiments (see Algorithms 1 and 2 for reference): λ, t_{idle}, and the Duty cycle (c) of periodic traffic are derived from [7]; i and α are lower than proposed in

[5] Two Zolertia Z1 motes were used. https://github.com/Zolertia/Resources/wiki/The-Z1-mote

[6] PC details: 16GB of RAM and an Intel® Core™ i5-4690 CPU @ 3.50GHz.

the aforementioned reference, but performed as expected[7].

The resulting testbed, whose network topology is shown in Figure 3, allows the emulation of sensor node traffic and its transport from one Gateway to another, mimicking a part of the process of forwarding sensor data towards upper layers of an example SDN IoT platform (like Figure 1). To derive accurate delay measures, the whole network is tightly synchronized using Precise Time Protocol (PTP, an IEEE Std 15882008 [15] implementation for Linux). The Controller sends multicast PTP synchronization messages to Gateways via the control channel.

TABLE I
PFIM and traffic characteristics

PFIM		
Parameter		**value**
i		3
λ		1% μ
α		250 ms
Traffic		
	Periodic	**Periodic/Bursty**
Duty cycle (c)	5 s	3 s / int(rand(1,5)) s
Packets per cycle (p)	10	10
t_{idle}	1 s	1 s

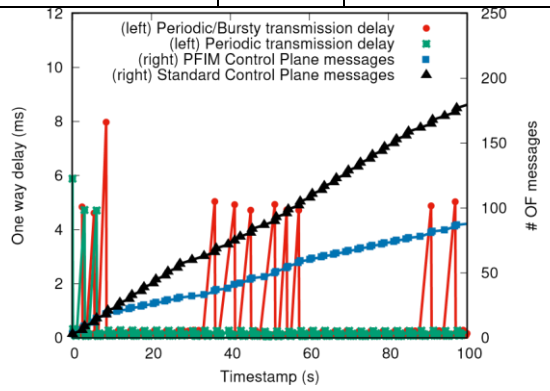

Fig. 6. A solely Periodic, and a Periodic/Bursty flows traverse a Gateway. Extended PFIM is able to identify periodicity and install flow rules pro-actively. When periodicity breaks, proactive flow rule installations are suspended for such flow.

C. Implementation of PFIM

The conventional reactive flow installation procedure and the impact on the delay of periodic flows is shown in Figure 4. The left y-axis shows the period between transmission and reception of a packet (from Gateway-B to Gateway-A in Figure 3), or one way delay. The right y-axis provides the accumulated number of OF messages related to the flow installation procedure, that is, packet_in, flow_mod and send_msg packets (as in Figure 2).

As the head of flow goes through the reactive flow installation process, is traverses the Gateway after d = 4 ms on average, while the rest of the flow (having a rule already installed at the Gateway) just experiences an average one way delay of d = 0.12 ms. Further, as each reactive flow installation process implies the exchange of 3 OF messages (see Section II-C), the accumulated number of such packets traversing the control channel grows linearly with each duty cycle.

Figure 5 shows the same metrics but this time using PFIM. Contrasting with Figure 4, after the period of a flow is learned only flow_mod messages are sent from Controller to Gateway, reducing the number of OF flow installation messages to a third. Moreover, as rules are already installed the head of flow experiences a one way delay reduction of around 90% (an average d = 0.4 ms).

D. Extended PFIM with heterogeneous traffic patterns

Extended PFIM makes it possible to identify periodic flows, but also to stop installing rules pro-actively if the periodicity disappears. Figure 6 shows the one way delay for two types of traffic patterns, namely Periodic and Periodic/Bursty (see Table I for details).

The sensor assigned with the Periodic/Bursty traffic pattern starts by transmitting p = 10 packets every c = 3 seconds, i = 10 times (periodic phase). After that, the periodicity of the generated flow is purposefully broken by changing to a variable duty cycle ($c \leftarrow$ int(rand(1,5))) for the next i = 10 iterations (bursty phase). Later, the sensor goes back to the periodic phase. This periodic/bursty behavior aims at emulating traffic from an arbitrary sensor that sends periodic reports, but also reacts to events.

Figure 6 shows (left-y axis) the one way delay, and (right-y axis) the accumulated number of OF messages associated to the aforementioned two types of traffic from real sensors. Extended PFIM is able to learn the duty cycle of Periodic flows. Later, when the periodicity is broken (at around T = 30 s in the Periodic/Bursty curve), reactive flow installations are used (evidenced by the increase in the delay of the head of flow).

Extended PFIM is able to forget periodic flows and identify them again. Results show the expected reduction in the delay of head of flows once periodicity is detected, and also the expected

[7] Reducing i and α even further caused PFIM to malfunction.

decrease in the number of OF messages observed through the control channel.

V. Conclusions and Open Subjects

For SDN IoT's Gateways serving an underlying Sensor layer, the reactive flow rule installation process increases the average delay of periodic flows. Further, as the rate of new flows per second grows the resulting control channel congestion increases the chances of packet loss, wasting sensors' very constrained energy resources.

This work presents experimental results from a proactive flow rule installation mechanism for periodic flows. Further, it introduces extensions that select between reactive and pro-active flow installation procedures according to the learned periodicity of a flow and the number of flow rule matches. Experimental results show that even with multiple transmission periods, pro-active flow rule installation techniques are both possible and useful for reducing control channel congestion; showing an average delay reduction of 90% for head of flows from periodic transmissions, such as the generated to report data from a wide variety of sensors in IoT deployments.

Future research on pro-active flow installation strategies may take advantage of OpenFlow (OF) Bundle messages, which allows for a group actions to be executed with a single control message. Furthermore, by performing deep packet inspection [16] and using OF's experimenter messages is possible to extract meaningful information from flows and network links, opening the way for energy consumption minimization in radio heterogeneous IoT networks. Finally, finetuning PFIM's learning algorithm for context-aware forwarding under variable network conditions, wireless control channels, and massive IoT traffic requiring very fast flow rule installations [17] are left as subjects for future studies.

References

[1] X. Ma and W. Luo, "The analysis of 6LoWPAN technology," in *Pacific-Asia Workshop on Computational Intelligence and Industrial Application, PACIIA'08*, vol. 1. IEEE, 2008, pp. 963–966.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[3] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "On the effect of forwarding table size on SDN network utilization," in *2014 Proceedings IEEE INFOCOM*. IEEE, 2014, pp. 1734–1742.

[4] H. Kim, J. Kim, and Y. B. Ko, "Developing a cost-effective OpenFlow testbed for small-scale Software Defined Networking," in *16th International Conference on Advanced Communication Technology*, Feb 2014, pp. 758–761.

[5] 5GPPP, "5G and the Factories of the Future," https://5gppp.eu/wp-content/uploads/2014/02/5G-PPP-WhitePaper-on-Factories-of-the-Future-Vertical-Sector.pdf, 2015.

[6] A. Wang, Y. Guo, F. Hao, T. Lakshman, and S. Chen, "Scotch: Elastically scaling up SDN control-plane using vswitch based overlay," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. ACM, 2014, pp. 403–414.

[7] P. Bull, R. Austin, and M. Sharma, "Pre-emptive Flow Installation for Internet of Things Devices within Software Defined Networks," in *2015 3rd International Conference on Future Internet of Things and Cloud*, Aug 2015, pp. 124– 130.

[8] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced study of SDN/OpenFlow controllers," in *Proceedings of the 9th central & eastern European software engineering conference in Russia*. ACM, 2013, p. 1.

[9] R. Khondoker, A. Zaalouk, R. Marx, and K. Bayarou, "Feature-based comparison and selection of Software Defined Networking (SDN) controllers," in *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*. IEEE, 2014, pp. 1–7.

[10] "POX Controller," https://openflow.stanford.edu/display/ONL/POX+Wiki, 2013, [Online].

[11] "Ryu Controller," https://osrg.github.io/ryu/, 2017, [Online].

[12] "Trema Controller," https://trema.github.io/trema/, 2016, [Online].

[13] "Floodlight Controller," http://www.projectfloodlight. org/floodlight/, 2017, [Online].

[14] "OpenDaylight Controller," https://www.opendaylight. org/, 2017, [Online].

[15] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, pp. 1–269, July 2008.

[16] G. Li, M. Dong, K. Ota, J. Wu, J. Li, and T. Ye, "Deep Packet Inspection Based Application-Aware Traffic Control for Software Defined Networks," in *2016 IEEE Global Communications Conference (GLOBECOM)*, Dec 2016, pp. 1–6.

[17] A. Nguyen-Ngoc, S. Lange, S. Gebert, T. Zinner, P. TranGia, and M. Jarschel, "Performance evaluation mechanisms for FlowMod message processing in OpenFlow switches," in *2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)*, July 2016, pp. 40–45.