Towards a Fully Programmable Internet of Things

Amaury Bruniaux, Julien Montavont, Thomas Noël, Georgios Papadopoulos, Nicolas Montavont

Towards a Fully Programmable Internet of Things

Amaury Bruniaux
IMT Atlantique, IRISA
ICube Laboratory (CNRS)
University of Strasbourg
Rennes, France
amaury.bruniaux@imt-atlantique.fr

Julien Montavont
ICube Laboratory (CNRS)
University of Strasbourg
Strasbourg, France
montavont@unistra.fr

Thomas Noel
ICube Laboratory (CNRS)
University of Strasbourg
Strasbourg, France
noel@unistra.fr

Georgios Z. Papadopoulos
IMT Atlantique, IRISA
Rennes, France
georgios.papadopoulos@imt-atlantique.fr

Nicolas Montavont
IMT Atlantique, IRISA
Rennes, France
nicolas.montavont@imt-atlantique.fr

***Abstract*—The devices of Internet of Things (IoT) networks can be deployed for extended period of time in inaccessible locations. After their deployment, new features, bug fixes, or changes in the client needs can require an update of the node's behavior, and this update has to be transmitted over the radio medium. Proposals have been made in the literature to apply the Software Defined Network (SDN) paradigm to wireless sensor networks but they focus on the packet forwarding layer of the stack. In this work, we propose to extend the programmability to the whole stack with a fully programmable device architecture able to handle the runtime programming of every protocol on any hardware. We detail the use of extended finite-state machines (XFSM) on which the architecture is based and their conversion to compact executable bytecode that can be sent over radio. Simulations demonstrate that this architecture can reproduce a protocol from the literature and enable interoperability between programmable nodes and legacy nodes.**

***Index Terms*—Programmability, SDN, Wireless sensor networks, XFSM**

I. Introduction

Numerous communication and networking related protocols have been designed to support the various services offered by Internet of Things (IoT)-enabling technologies [1]. However, selecting and tuning a protocol suite for one application remain complex tasks, especially if the optimal values are outside the range of recommended ones [2]. Unfortunately, optimizing a configuration generally involves a fastidious trial and error process as optimal parameters depend on the network topology, the radio environment, and the traffic model, among others. In addition, typically the IoT networks are time-variant, meaning that a configuration may become invalid after some time. As a result, a constant reconfiguration is required to converge the network to the optimum performance.

In addition to these factors, the services offered over an IoT network can be continuously customized and adapted to the clients' needs, or multiple service providers can share the same infrastructure. We are shifting IoT from application-centric deployment to IoT as a Service (IoTaaS) paradigm. The ability to change on the fly the behavior (e.g., features, protocols, or configuration) of the network devices - the *programmability*, should be a key feature of IoT technology. However, IoT devices are often hard to access once deployed, and human interventions are costly, preventing on-site firmware flashing from being a practical solution for device reprogramming at large scale. We will employ the term *programmability* when referring to the ability to modify the behavior of a device through the wireless medium. Programmable networks have different scales, ranging from solutions focusing on changing the whole firmware of a node Over-The-Air (OTA) [3] to solutions that make the data plane programmable [4].

OTA firmware update permits the distribution of software updates to smart devices. However, firmware image size can be large, and transmitting large quantities of bytes over a wireless multi-hop network can be challenging. On the other hand, Software Defined Network (SDN) is the *de facto* solution for programming the data plane. The SDN paradigm was first introduced in wired networks [5], and it consists in removing the control plane, i.e., how frames or packets should be handled, from the nodes to a logically-centralized entity called the controller. As a result, the controller pushes simple commands or rules to nodes for moving frames or packets from sources to destinations. SDNs were first implemented to remove human errors from network configuration and allow simpler management, easier interoperability, and a faster pace of innovation. However, wired-based networks often benefit from point-to-point and reliable links, while wireless networks manage a shared and rather a lossy medium. Therefore, the Medium Access Control (MAC) layer is of crucial importance for achieving the best performance. Current SDN approaches for IoT are limited in the sense that most of the MAC layer operations are still viewed as a monolithic block from the controller. For example, the controller can modify a TSCH schedule on the fly, as proposed in [4], but it has no control over the intra-timeslot organization, retransmission, carrier sense, etc.

In this context, we propose to push forward the SDN paradigm

with a novel device architecture enabling the programming of the whole communication stack in IoT networks. This architecture brings the flexibility, reusability, and programmability required to make IoT a key element of the future Internet. In this paper, we propose to describe any communication protocol with an Extended Finite State Machine (XFSM) [6] whose transitions are based on a set of elementary abstractions. Our architecture allows both stateless and stateful operations and relies on devices being able to execute a provided XFSM translated into bytecode. As a result, network nodes become simple MAC/network processors that execute elementary commands enforced by the controller. The specific contributions of this paper include:

1) The design of a fully programmable device architecture;
2) The design of the machine interface language based on elementary abstractions representing the actions, conditions, and events expressing the behavior of the device;
3) The bytecode encoding and decoding, structuring the XFSM and the implementation of those algorithms;
4) The proof of concept validation using OMNeT++, including an interoperability test with the X-MAC protocol [7]. X-MAC is a well-known contention-based MAC protocol using preamble sampling that is supported by OMNeT++.

The rest of the article is organized as follows. A brief introduction of SDN and programmable approaches is provided in Section II. Section III presents our fully programmable architecture and Section IV details our simulation campaign to validate our approach. Finally, the article concludes in Section V.

II. Related work

A. Software Defined Networks

SDN are networks where a centralized controller is in charge of the control plane while the network nodes only take care of the data plane [5]. The control plane refers to the functions that determine how frames or packets should be processed. By contrast, the data plane is the actual forwarding process. Generally, the controller collects information about the current network status (topology, link characteristics, application requirements, etc.) and pushes actions to apply to incoming frames or packets on network nodes via a southbound API. Typical actions are discarding packets, forwarding packets on specific egress interfaces, or encapsulating packets with some extra headers. In IoT networks, removing the network intelligence from network nodes makes sense, mainly because of their computational power, memory, and energy constraints.

The actions defined in SDN-WISE [8] add the possibility to match any sequence of bytes in packet headers instead of being limited by the fields of well-known protocols. In addition, it is a stateful solution in the sense that nodes can store and use local variables as conditions, extending the action space. However, it relies on a typical match/action pipeline that prevents the controller from having complete control of the communication stack of the nodes. SDN-TSCH [4] allows a controller to define the TSCH schedule regarding the application needs and guarantees flow isolation. However, this solution is dedicated to scheduled networks and does not expose basic MAC/network primitives to the controller. Wireless networks being time-variant, the control traffic exchanged over the southbound API can be intense to maintain a global view of the network on the controller. As a result, several SDN solutions focus on optimizing this aspect. Atomic-SDN [9] proposes a scheduling scheme that enables the controller to configure the nodes with synchronous flooding. On the other hand, IT-SDN [10] introduces source routing for flow table entries coming from the controller, reducing the number of packets necessary to install a routing path. Finally, VERO-SDN [11] proposes a secondary communication channel (via a second radio interface) dedicated to the traffic between the nodes and the controller.

As we can see, current SDN solutions for IoT networks are limited in several aspects. Actions are mainly triggered by incoming packets, and more complex interactions with hardware components, such as clocks, radios, interrupts, and sensors are not considered. Even though some approaches are stateful, they do not take advantage of the various metrics available locally, such as link quality indicators, packet delivery ratios, or sensor measurements. Finally, SDN solutions for IoT networks mainly focus on frame/packet forwarding and disregard the diversity of ways of sending a frame/packet on a shared wireless medium.

B. Over-the-air firmware updates

In order to add new features or fix security breaches, IoT devices require frequent updates [3]. OTA firmware update protocols enable a firmware image to be compiled, transmitted, and installed from a base station in order to change the behavior of an already deployed device. The images typically consume dozens of kilobytes of memory, and sending a whole image over a multi-hop wireless network is costly because it mobilizes radio resources at the expense of data traffic. Efforts have been made towards incremental programming schemes that reduce the size of update packets, and make it possible to reuse old images to create new ones. However, there are still challenges to be addressed by OTA firmware update protocols. IoT nodes have a limited memory size and may not handle more than one image. In addition, the new features can not be dynamically linked to the system and require a reboot that takes time, forcing nodes to bootstrap again the communication stack. Some approaches tackles those issues by using in-place patching and memory space trampolines [12]. On the other hand, writable pieces of code can be stored in RAM [13].

C. Programmable MAC

Tinnirello et al. [14] already paved the way toward a programmable communication stack with a reprogrammable MAC for wireless networks. Similar approaches can be classified under three categories of programmability [15]:

- monolithic implementations can only change the whole MAC protocol at once;
- parametric implementations enable external entities to change the values of intra-system parameters;
- modular implementations are the most flexible and can rearrange independent software functions.

Modular designs rely on a set of abstracted functionalities used as building blocks of a finite state machine that needs to be represented in machine language to be run by the devices. In [16], the MAC layer is described in a C-like meta-language before being parsed and converted into a linked list of actions that are executable by the nodes. In [14], an XFSM representing a 802.11 MAC protocol is converted into bytecode that is executed by the device. Such an approach is the lightest and most natural way to describe a system with a high degree of granularity and modularity. As a result, we propose to extend the use of the XFSM representation to the whole behavior of the device.

III. Fully Programmable Device Architecture

A. Scope and objectives

We aim at designing the building blocks of a general programmable node that supports any hardware platform. An architecture that provides a fully programmable protocol suite should include the following properties:

- *full programmability* - the ability to program any behavior that can be run by the physical components of nodes;
- *compactness* - the size of updates has to be as light as possible;
- *hotplugging* - the possibility to link new functions at runtime without interrupting the system, preventing service interruptions;
- *modularity* - the ability to extend or modify parts of the protocol stack running on network nodes.

This paper proposes an architecture that focuses on the two first properties and opens the way to offer the two last properties in the near future.

B. Architecture overview

We designed a general architecture adapted to XFSM based devices regardless of their connected peripherals. This architecture contains two kinds of entities: physical components (radios, clocks and data sources) and memory structures. The architecture does not impose a specific implementation of the control interface and of the memory management system as long as it is interoperable with the interface and the various data structures of the architecture are available.

C. Extended Finite State Machines

XFSMs are Turing-complete [6]. As a result, they can be used to program any Turing machine, and more precisely any communication protocol. XFSMs have been used by [14] to run 802.11 networks with a programmable MAC layer. We extend this work by going beyond a single layer and consider a fully programmable communication stack. We also extend the bytecode structure presented in [14] and add new concepts such as templates, hosts, tables, and variable types. An XFSM is a finite state machine in which transitions between states are defined by event, condition, and action. It also includes memory registers where variables can be stored or read. A transition is only performed if the event occurs while all the required conditions related to this transition are met. A transition completes with commands sent to the entities in charge of executing the specified actions. Fig. 1 illustrates the XFSM of the sender part of the send and wait protocol.

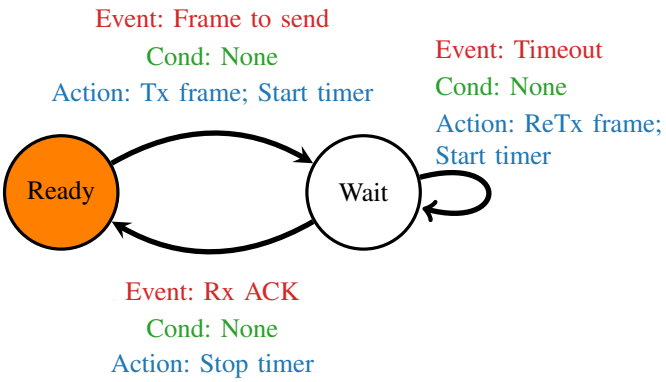


Fig. 1: Send and wait protocol in XFSM format

An event, such as the reception of a packet or the end of a timer, consists of a signal sent by a physical component and received by the XFSM engine. A condition takes the form of a comparison expression using relational operators, such as equal, greater than, or less than. Operands are variables saved in the memory structures. Since every variable present in the node can be accessible that way, we avoid the issue of current IoT SDN whose statefulness only applies to artificial variables. An action is a command to be executed by a component and can take arguments detailing how to execute the command. For example, the action that sets a new value to a global variable takes as arguments the location of the variable and the new value. All variables are identified by a type that enables the XFSM engine to choose the proper way to read the two bytes that describe them. Variable types are reported in Table I.

TABLE I: Variable types defined in our XFSM

Type	Detail
Global	simple variable storing a value
Table	array that links values to specific keys
Identifier	number associated to a component that is used to send a command to this component
Random	random value between two values
Timer	value that indicates if a timer has been started or has expired
Access	several variable types for various ways of pointing to a specific byte in a buffer

D. Abstractions

The set of identified events, conditions and actions is summed up in Table II. As long as conflicts of identifiers are avoided,

a device only needs to know the abstractions that it applies to its hardware. Furthemore, new abstractions can be added to the set to handle new hardware capacities without impacting old devices, it is the responsibility of the controller to only reference abstractions that target devices are capable of handling. In some cases, an action is represented by several action identifier, each representing a variation of the action; or a combination of two subsequent actions can be merged into a new action. For instance, an action REFRESH_TIMER is a sequence of STOP_TIMER and START_TIMER. This allows to save space in the bytecode description of the XFSM.

Relying on a set of abstractions implies that all supporting devices expose their set of abstractions to the engine running the XFSM. While this is out of the scope of this paper, we can imagine that the device communicates their available abstractions to the controller so that it can accordingly design a suitable program without including abstractions unknown of the device.

States that are the source of transitions triggered by the event VIRTUAL are virtual events, they are immediately left once entered. This enables to skip the limitations of three transitions with three actions each per state.

TABLE II: Abstractions defined in our XFSM

Events	**Actions**
VIRTUAL	START_TIMER
TIMER_EXPIRES	STOP_TIMER
CCA_BUSY	REFRESH_TIMER
CCA_FREE	CCA
COLLISION	SET_VAR
RX_COMPLETE	ADD_ROW
TX_COMPLETE	COPY_PACKET
MEASURE_COMPLETE	DEQUEUE
TX_MODE_SWITCH	MEASURE
RX_MODE_SWITCH	SET_FIELD
RADIO_SLEEP_SWITCH	INCREASE_VAR
Conditions	RADIO_TX_MODE
NO_CONDITION	RADIO_RX_MODE
GREATER	RADIO_SLEEP_MODE
EQUALS	EMPTY_RX_BUFFER
QUEUE_LONGER	SEND_PKT
QUEUE_SHORTER	PARSE_PKT
RADIO_MODE_IS	COPY_TEMPLATE

E. Bytecode structure

The bytecode representing the XFSM is structured in five regions, each containing several instances of smaller sequences representing the abstractions composing the program. The number of instances of each region is present at the very beginning of the bytecode in order to delimit the end of each region. We make many design choices, and one could compress further the bytecode, but this first version is already far more compact than OTA firmware images, as shown in Section IV-A.

1) Transition region: this region contains all the transitions of the XFSM grouped by exit state. For each state, a preliminary byte specifies the number of transitions exiting the state and the number of actions for each transition. This first byte is followed by a transition sequence for each transition exiting the state. A transition sequence contains an event ID, the source of the event, a condition to verify (optional), the next state to enter, and the list of IDs of the actions to execute. Upon reception, a node first parses the bytecode and stores the memory addresses of the head of each state so that potential transitions can be evaluated quickly.

2) Action region: this region contains a description of each action that can be triggered when performing a transition. Actions are described with 6 bytes: the ID of the elementary action and up to two variables, including their types. They are similar to calling a function with two arguments in typical programming language. Finally, two bytes describe each variable.

3) Condition region: similarly to the action region, this region contains a description of each condition that can be verified when evaluating a transition. The format is exactly the same as an action entry, except that ID refers to the elementary condition.

4) Parameter region: this region details how many global variables to store and the structure of the tables. Initial values can also be set directly in the bytecode.

5) Template region: this region defines the packet structure, meaning the organization of the various packet headers. A template contains the number of fields, the size of each field, and optional default values. Actions can set any field of a template once it has been copied to a transmission buffer.

IV. Validation

A. XFSM and bytecode size

We designed the XFSM of the XMAC protocol along with a basic application protocol for simple data traffic by basing on its legacy implementation in OMNeT++. Transitions, composed actions and conditions, and templates have been arranged in CSV files that are read by our bytecode creator script. A simplified version of the output of the creator is displayed on Figure 2, with actions being removed or renamed for clarity. Orange circles represent virtual states. Its size is 699 bytes, which means it could be handled by fragmentation protocols such as the IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) protocol [17] and transmitted to the devices by a controller. It is also much smaller than other implementations of the protocol and images of OTA protocols as shown in Table III. This is a significant advantage of the XFSM representation, potentially avoiding long periods of traffic load on IoT networks performing an image update.

TABLE III: Comparison of image sizes

Approach	**Implemented image**	**Size in bytes**
This paper	XMAC based node behavior	699
[16]	Monolithic XMAC	1242
[16]	Toolchain XMAC	3326
[18]	6LoWPAN router	47772

B. Simulation

We implemented our bytecode engine in an OMNeT++ node, an ran simulations on two networks: one of programmable

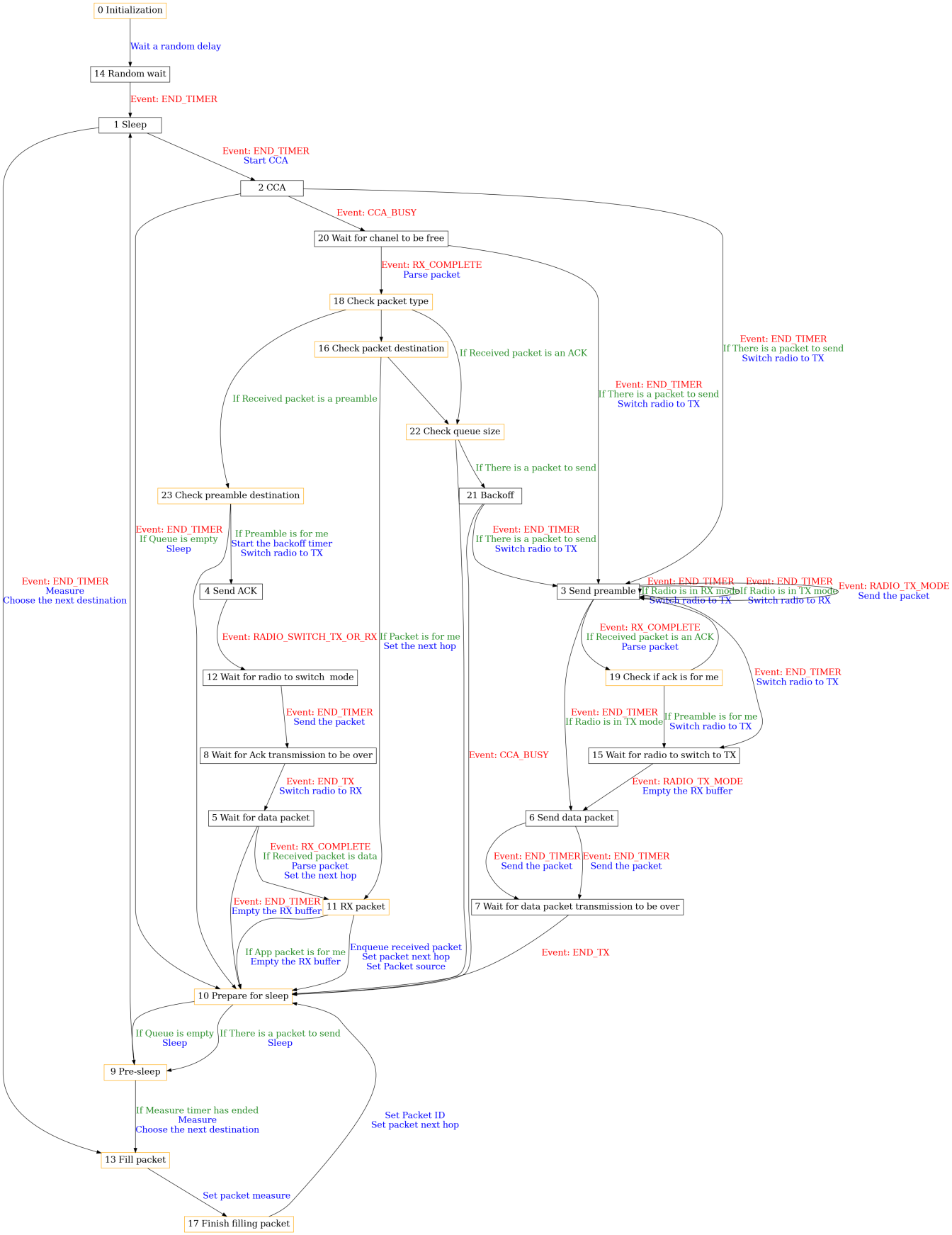
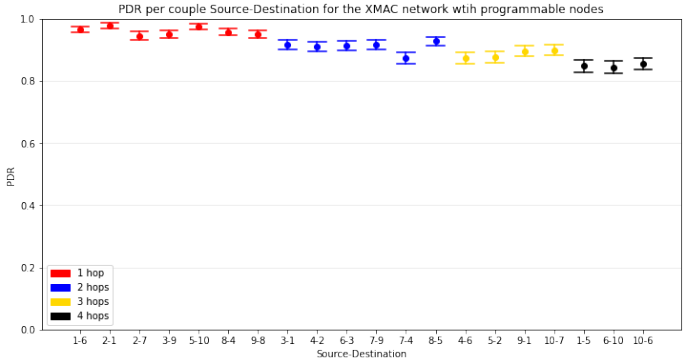
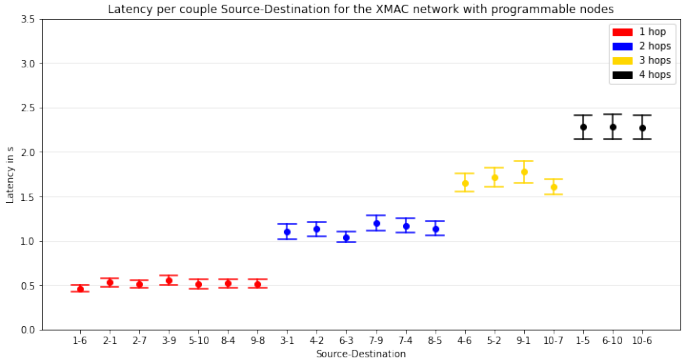


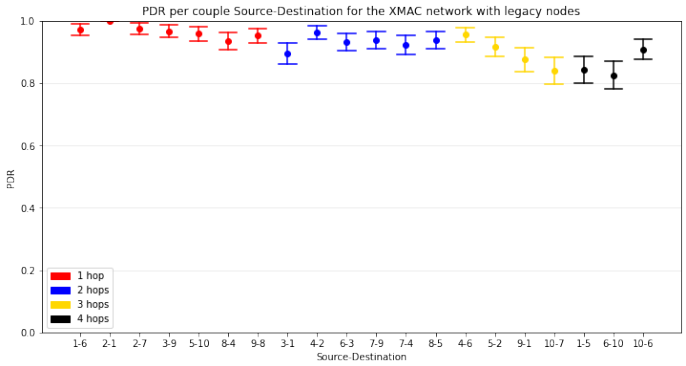
Fig. 2: XFSM representation of an XMAC-based device



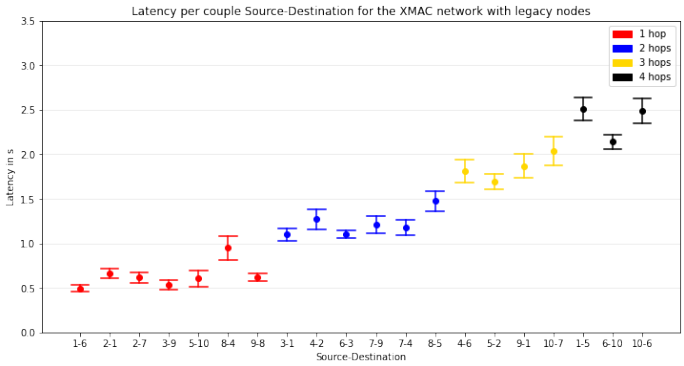
(a) PDR for the programmable network



(b) Latency for the programmable network



(c) PDR for the legacy network



(d) Latency for the legacy network

Fig. 3: Performance indicators of the simulated networks

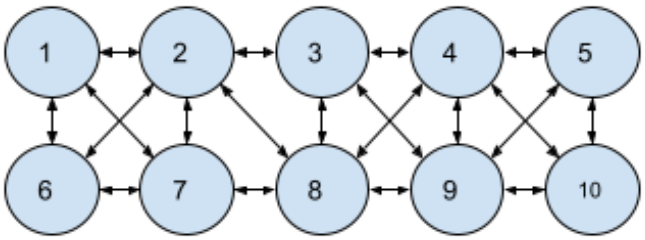


Fig. 4: Topology of the simulated networks

TABLE IV: Simulation settings

Setting	**Value**
Number of runs	8
Run duration	1000 minutes
Application traffic period	Uniformly randomized between 16 and 30 seconds
Effective link quality	100%
Number of retransmissions	0

nodes and the other of legacy nodes from the inet package. We set the topology shown in Figure 4 with the same parameters in both networks and expected that they behave identically. The protocol and simulation settings can be found in Table IV and the performances of both networks are displayed on Figure 3.

As expected, both the Packet Delivery Ratio (PDR) and the latency are following the trend of respectively decreasing and increasing with the number of hops between the source and the destination. Despite some individual discrepancies like the latency of the pair (6,10) or the PDR for the pair (3,1), the confidence intervals are overlapping for each plot point and we deduce that both networks have the same behavior.

V. Conclusion

In this work, we addressed the issue of programmability in wireless sensor networks, and more specifically the challenge of allowing a controller to have full control over the behavior of the devices forming the network. Previous works on programmability achieved either limited programmability or relied on images of considerable size that entailed drawbacks such as an increased traffic load during the updates. Our architecture relying on XFSMs enables to design protocols from scratch that can be run on any hardware running the XFSM engine, and simulation results show an example of protocol being run on this engine.

Our next steps are addressing the bootstrapping phase of the network by including a controller responsible of providing

a brand new image to nodes that join the network; and modularity: devices should not receive a whole bytecode when only a portion of it is modified.

REFERENCES

[1] A. Kumar, M. Zhao, K. Wong, Y. Guan, and P. Chong, "A Comprehensive Study of IoT and WSN MAC Protocols: Research Issues, Challenges and Opportunities," *IEEE Access*, vol. 6, 2018.

[2] G. Anastasi, M. Conti, and M. Di Francesco, "A Comprehensive Analysis of the MAC Unreliability Problem in IEEE 802.15.4 Wireless Sensor Networks," *IEEE Transactions in Industrial Informatics*, vol. 7, no. 1, 2011.

[3] K. Arakadakis, P. Charalampidis, A. Makrogiannakis, and A. Fragkiadakis, "Firmware Over-the-air Programming Techniques for IoT Networks - A Survey," *ACM Computer Survey*, vol. 54, no. 9, 2022.

[4] F. Veysi, J. Montavont, and F. Théoleyre, "SDN-TSCH: Enabling Software Defined Networking for Scheduled Wireless Networks with Traffic Isolation," in *proc. of the IEEE Symposium on Computers and Communications (ISCC)*, 2022.

[5] T. Luo, H. Tan, and T. Quek, "Sensor OpenFlow: Enabling Software-Defined Wireless Sensor Networks," *IEEE Communications Letters*, vol. 16, no. 11, 2012.

[6] C. Wang and M. Liu, "A Test Suite Generation Method for Extended Finite State Machines Using Axiomatic Semantics Approach," in *proc. of the IFIP TC6/WG6.1 International Symposium on Protocol Specification, Testing and Verification*, 1992.

[7] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-MAC: A Short Preamble MAC Protocol for Duty-Cycled Wireless Sensor Networks," in *proc. of the ACM International Conference on Embedded Networked Sensor Systems (SenSys)*, 2006.

[8] L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, "SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for WIreless SEnsor networks," in *proc. of the IEEE Conference on Computer Communications (INFOCOM)*, 2015.

[9] M. Baddeley, U. Raza, A. Stanoev, G. Oikonomou, R. Nejabati, M. Sooriyabandara, and D. Simeonidou, "Atomic-SDN: Is Synchronous Flooding the Solution to Software-Defined Networking in IoT?" *IEEE Access*, vol. 7, 2019.

[10] R. Alves, D. Oliveira, N. S. G.A., and C. Margi, "The Cost of Software-Defining Things: A Scalability Study of Software-Defined Sensor Networks," *IEEE Access*, vol. 7, 2019.

[11] T. Theodorou and L. Mamatas, "A Versatile Out-of-Band Software-Defined Networking Solution for the Internet of Things," *IEEE Access*, vol. 8, 2020.

[12] C. Zhang, W. Ahn, Y. Zhang, and B. Childers, "Live code update for IoT devices in energy harvesting environments," in *proc. of the Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, 2016.

[13] W. Dong, Y. Liu, C. Chen, L. Gu, and X. Wu, "Elon: Enabling efficient and long-term reprogramming for wireless sensor networks," *ACM Transactions on Embedded Computing Systems*, vol. 13, no. 4, 2014.

[14] I. Tinnirello, G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, and F. Gringoli, "Wireless MAC processors: Programming MAC protocols on commodity Hardware," in *proc. of the IEEE International Conference on Computer Communications (INFOCOM)*, 2012.

[15] P. Isolani, M. Claeys, C. Donato, L. Granville, and S. Latré, "A Survey on the Programmability of Wireless MAC Protocols," *IEEE Communications Surveys and Tutorials*, vol. 21, no. 2, 2019.

[16] X. Zhang, J. Ansari, L. Martinez, N. Linio, and P. Mähönen, "Enabling rapid prototyping of reconfigurable MAC protocols for wireless sensor networks," in *proc. of the IEEE Wireless Communications and Networking Conference (WCNC)*, 2013.

[17] G. Montenegro, N. Kushalnagar, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," IETF RFC 4944, Sep. 2007.

[18] H. Park, J. Jeong, and P. Mah, "Non-invasive rapid and efficient firmware update for wireless sensor networks," *UbiComp 2014 - Adjunct Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pp. 147–150, 09 2014.