Hindawi
Security and Communication Networks
Volume 2018, Article ID 8734389, 14 pages
https://doi.org/10.1155/2018/8734389

Research Article

WS^3N: Wireless Secure SDN-Based Communication for Sensor Networks

Renan C. A. Alves, Doriedson A. G. Oliveira, Geovandro C. C. F. Pereira, Bruno C. Albertini, and Cíntia B. Margi

Escola Politécnica, Universidade de São Paulo, São Paulo, Brazil

Correspondence should be addressed to Cíntia B. Margi; cintia@usp.br

Received 29 March 2018; Revised 6 July 2018; Accepted 18 July 2018; Published 1 August 2018

Academic Editor: Leandros Maglaras

The Software Defined Networking (SDN) paradigm can provide flexible routing and potentially support the different communication patterns that exist in Wireless Sensor Networks (WSN). However applying this paradigm to resource-constrained networks is not straightforward, especially if security services are a requirement. Existing SDN-based approaches for WSN evolved over time, addressing resource-constrained requirements. However, they do not integrate security services into their design and implementation. This work's main contribution is a secure-by-design SDN-based framework for Wireless Sensors Networks. Secure node admission and end-to-end key distribution to support secure communication are considered key services, which the framework must provide. We describe its specification, design, implementation, and experiments considering device and protocol constraints. The results indicate that our approach has achieved such goals with acceptable overheads up to medium sized networks.

1. Introduction

Wireless Sensor Networks (WSN) are composed of resource-constrained devices with the purpose of gathering information from the environment. These devices can sense, process, and communicate, increasing the information and perception about the world. In fact, WSN constitutes an important tool for real-world applications, improving the information gathering process in many scenarios such as precision agriculture, biodiversity monitoring/research, elderly/disabled monitoring, and health support as well.

There are many applications for WSN in which security services, such as data confidentiality, integrity, and data source authentication, are crucial. Nevertheless, given the limitations of the hardware typically used, the security mechanisms of conventional networks are likely to cause undesirable effects, such as a significant increase in energy consumption and/or communication delay, either from the overhead of processing or from the increment of transmitted data.

Thus, one of the main challenges for the implementation of security mechanisms in WSN is to solve the conflict between minimizing resource consumption and maximizing security [1]. Furthermore, characteristics such as mobility and heterogeneity pose new challenges for deployment of communication protocols.

Routing protocols for WSN evolved over time. Most of the deployments had specific purposes and used routing algorithms tailored for those needs. For instance, directed diffusion was designed for event and interest based networks [2], while Collection Tree Protocol (CTP) is adequate for convergecast exclusive traffic pattern [3]. Then RPL (RFC 6550 [4]) was proposed to address different routing patterns, such as node-to-node, node-to-sink, and sink-to-node communication. However, it is designed to be efficient in node-to-sink traffic.

The issues of routing flexibility and provision of security services can potentially be addressed by the so-called Software Defined Networking (SDN), a paradigm that uses logically centralized software to control the behavior of a network.

OpenFlow [5] is a widely known implementation for the SDN southbound protocol, which has been used on wired networks with devices empowered with relatively more

resources. On the other hand, SDN for a WSN scenario imposes different challenges and requirements. Among the challenges, we highlight the limited resources: energy, processing, memory, and communication. Requirements are related to the applications characteristics (e.g., data frequency and size), as well as to the nodes behavior due to duty cycles, operating systems, and programming approach.

The literature presents two main approaches to SDN applied to WSN-like scenarios. First approach is to consider the sensor nodes as non-SDN components, which must rely on other SDN-capable devices [6, 7]. The second approach considers that WSN nodes are SDN-capable, such as the Flow-Sensor [8], Sensor OpenFlow [9], SDN-Wise [10], and TinySDN [11].

Unfortunately there are drawbacks for these approaches. The ones based on OpenFlow have issues concerning frame sizes, introduced overhead, and use of TCP as underlying communication protocol. SDN-Wise implementation is not completely available for download and use. TinySDN provides the code and related documentation, but it is highly dependable on TinyOS; thus limiting the platforms it could be deployed. Also the use of the ActiveMessage function limits interoperability with other systems.

Furthermore, none of these approaches consider security by design. Control messages are not authenticated, and thus a malicious node could attack the network causing routing holes.

To the best of our knowledge, none of them address secure node admission procedure and end-to-end key distribution, which are modern security requirements.

The main contribution of this work, unlike the existing literature, is the design and implementation of a **secure SDN framework for WSN**, which takes into account (1) node admission, (2) key distribution, and (3) device and protocol constraints typical of WSN networks. This was possible to be achieved due to a careful selection and integration of cryptographic algorithms so that users can explore the trade-off between security and resource usage [12]. As a result, based on both the knowledge of the cryptographic algorithms trade-offs (e.g., security levels, block sizes) and on the broader vision of the devices/resources within an SDN network, the SDN controller is able to make improved routing decisions for establishing communication flows that minimize network energy consumption.

Notice that we choose to focus on security in order to create the basic services that in turn can be used to secure control messages, a critical aspect for SDN. The application which will run on top of this Software Defined Wireless Sensor Network (SDWSN) could benefit from such mechanisms and even use same infrastructure to secure its messages. Providing the basic security services on a combined SDN/WSN approach is *per se* a research topic on its own and the achievement of higher-level services such as privacy should be taken into account by the applications designers, since the granularity and requirements of their messages are application dependent.

The remainder of this paper is organized as follows. Section 2 discusses related work, while Section 3 presents requirements and motivation. Section 4 describes our proposed framework architecture, while Section 5 presents performance analysis. Section 6 concludes this paper, presenting final considerations and future work.

2. Related Work

The concept of network programmability gained attention with Software Defined Networks (SDN) and OpenFlow protocol. In SDN, programmability is achieved by decoupling the forwarding plane from the control plane, thus centralizing routing decisions on specialized controller nodes. These nodes install routing rules on the routing nodes, which is achieved by a so-called southbound interface protocol (e.g., OpenFlow).

Hence, routing devices forward packets by applying previously installed rules to incoming packets. If a packet does not match any of the current rules, a query is sent to the controller. Routing is based on flows, a concept not necessarily driven by destination addresses, since other packet fields may be checked or another abstraction may be employed. For example, OpenFlow rules may check TCP ports or the VLAN tag.

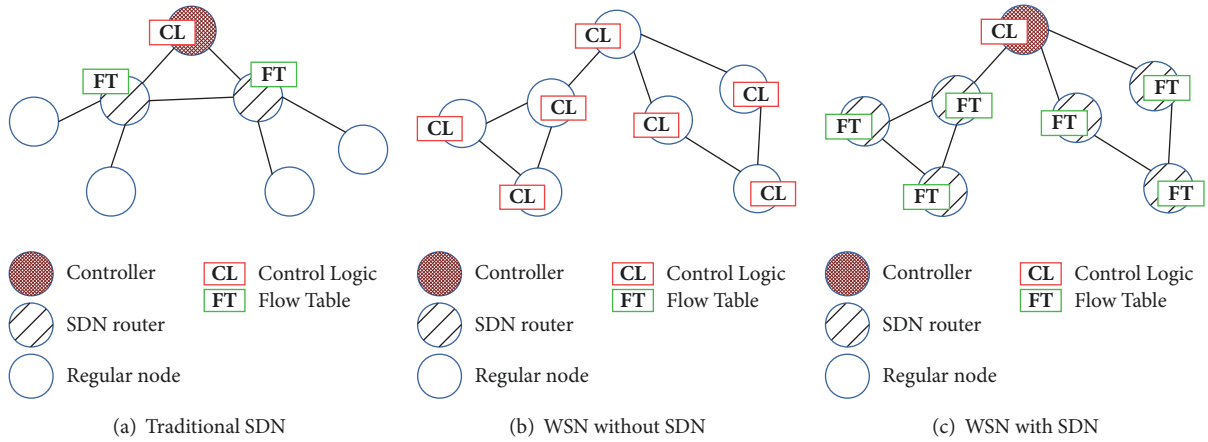
However, enabling network programmability on constrained devices, such as WSN motes, is not a trivial task. One of the challenges is related to limitations in frame sizes (127 bytes considering IEEE 802.15.4). Additionally, every WSN node is a router; therefore every node must be SDN enabled in a software defined sensor network, as depicted in Figure 1. Another challenge is that usually there is a dedicated control channel in wired SDN (out-of-band control), while wireless SDN uses in-band control; i.e., both data and control packets share the same wireless link.

Our literature review was done by using literature search tools with main keywords (such as SDN, WSN, and security) and filtering results to recent work on the subject, which in turn provided references and surveys of the previous literature. We could not find any SDWSN effort that takes security into consideration. However, we could identify a few previous efforts on implementing SDN on WSN and on security in WSN, which we classify as (1) OpenFlow based, (2) new southbound protocol specification, (3) security only, or (4) architectural only.

(1) OpenFlow based. *Sensor OpenFlow* [9] follows an OpenFlow based approach, discussing technical challenges such as control overhead and reliability. But other important aspects are not discussed including topology discovery, information gathering mechanism, and packet size. In addition, there are no experimental results.

A similar work, *Flow-Sensor* [8], is also based on OpenFlow, thus inheriting some of its characteristics such as TCP-based communication and security primitives. However, WSN-specific issues were not discussed, such as layer 2 compatibility, device capability to run TLS-based protocols (required by OpenFlow), and TCP poor performance on Low power and Lossy Networks (LLNs) [13].

Prasad and Ali proposed an identity-based scheme for software defined ad hoc networks [14]. Nonetheless, they use OpenFlow and do discuss devices limitation; therefore their



(a) Traditional SDN (b) WSN without SDN (c) WSN with SDN

Figure 1: Differences of routing approaches.

scheme is not adaptable to the WSN scenario. Also, identity-based schemes are subject to key escrow.

Similarly, another work [15] focuses on securely porting OpenFlow to the ad hoc scenario. But specification details or results based on any implementation of the proposal are not provided.

(2) New southbound protocol specification. These proposals do not rely on OpenFlow architecture. *SDN-Wise* [10] uses a stateful flow table and implements its own neighbor discovery protocol. Up to three packet fields may be matched using comparison operators. The experiments assessed round trip time and control overhead metrics.

TinySDN [11] relies on Collection Tree Protocol (CTP) to allow multiple controllers in the same network. CTP is also used for neighbor discovery purposes. The metrics analyzed are time to reach controller and time to deliver the first data packet.

Data packets are matched according to a flow-identification field (FlowID), which should correspond to an application. For example, suppose there is a FlowID that represents "temperature collection." The controller could identify temperature sinks and set routes for the sink with best metrics.

IT-SDN [23] improves on IT-SDN as its specification is not coupled with any operating system and the underlying discovery algorithms are replaceable. It also implements the concept of FlowID packets matching.

SDN-Wise and TinySDN were actually implemented and tested, but the scenarios considered in the performance analysis are limited in number of nodes (up to 7). Although both designed their own southbound protocol, secure node admission and secure data transmission were not addressed.

IT-SDN is discussed as the routing infrastructure of a secure sensing as a service architecture [24]. The authors argue that postquantum schemes should be used to distribute symmetric cryptographic keys and several schemes are discussed and benchmarked. Although the paper presents IT-SDN performance analysis in networks up to 25 nodes, the details of how to introduce secure mechanisms in the software defined WSN are not discussed nor implemented.

Black SDN [16] attempts to secure resource-constrained SDN by fully encrypting layer 2 frames (including headers). However, there is no real implementation and some fundamental questions are not discussed, such as topology discovery, compatibility to existing IEEE 802.15.4 networks, cryptographic keys distribution, and information about the intended southbound protocol.

(3) Security only. Different authors have proposed cryptography-based security solutions specific to Wireless Sensor Networks. Some of the most prominent are the IEEE 802.15.4 security mechanism [25], TinySec [26], Minisec [27], ContikiSec [28], and, more recently, WSNSec [29] and AdC [30]. All of these solutions focus on hop-by-hop security, since they were designed/implemented to operate at the lower layers of the OSI protocol stack, acting only at the link and/or network layer. While such an approach is suited to provide link-layer security, it unfortunately cannot guarantee end-to-end security. Furthermore, most of these architectures were designed and implemented for specific hardware platforms and operating systems, preventing its wide adoption.

The work by Boudia et al. [31] introduces a scheme to provide end-to-end security services, however the important task of key distribution is not handled. Also, the scheme is based on clustering, which limits the network flexibility.

(4) Architectural only. Other authors discussed architectural aspects of Software Defined Networking on WSN or IoT [17–22], but they do not address protocol specification or implementation issues and therefore do not contain a proof-of-concept of their design. Akram and Gokhale [18] propose a cross-layer design that leverages IEEE 802.15.4 characteristics, including layer 2 encryption. Nonetheless, secure node admission and key establishment were not discussed.

The SDN framework proposed herein, WS^3N (Wireless Secure SDN-based Communication for Sensor Networks), enhances previous proposals as it addresses security issues, such as secure node admission, key distribution, and establishment of a secure control channel. To this end, we designed a southbound protocol to support these operations based on authenticated key agreement [32]. It is fully compliant

Table 1: Comparison between related works.

	SB protocol	Neighbor & Controller discovery	Reconnection	Crypto	Node admission & Key exchange	Implementation
[9]	OF-based	Not informed	Not discussed	OF-based TLS	OF-based TLS	None
[8]	OF-based	Not informed	Not discussed	OF-based TLS	OF-based TLS	Custom simulator
[14]	OF-based	Not informed	Not discussed	OF-based TLS	OF-based TLS	NS3
[15]	OF-based	Not informed	Not discussed	OF-based TLS	OF-based TLS	None
[11]	Own spec	CTP	Not discussed	None	None	TinyOS
[10]	Own spec	Own protocol	Not discussed	None	None	Contiki
[16]	Own spec	Broadcast/ flooding	Not discussed	Yes	None	Custom simulator
[17]	Not informed	Not informed	Not discussed	None	None	None
[18]	Own spec	based on MAC	Not discussed	Yes	None	None
[19]	Not informed	Not informed	Not discussed	Mentions, not detailed	None	None
[20]	Not informed	Not informed	Not discussed	None	None	None
[21]	Not informed	Not informed	Not discussed	Mentions, not detailed	Mentions, not detailed	None
[22]	Not informed	Not informed	Not discussed	Mentions, not detailed	Certificate based	None
[23, 24]	Own spec	replaceable	Not discussed	Mentions, not detailed	Mentions, not detailed	Contiki
WS^3N	Own spec	6lowpan ND	Discussed	Yes	Yes	Contiki

to IEEE 802.15.4 frame length constraints, but it is not coupled with it and could be implemented on top of other standards. The solution has been implemented and tested on Contiki OS, assessing the metrics of packet delivery ratio, control overhead, and time to perform initial procedures (authorization, key bootstrap, key agreement, and data flow setup).

Table 1 contains a comparison between WS^3N and the cited previous work according to the southbound protocol design, neighbor and controller discovery mechanism, how to deal with reconnection, presence of security services by design (symmetric cryptography based: authentication, confidentiality, integrity; asymmetric cryptography based: key distribution, node admission), and the implementation status. Note that WS^3N is the only work that discusses node reconnection and fully specifies how to integrate the cryptographic primitives with the network layer.

3. Requirements and Motivation

WS^3N's main goal is to provide security services for the combined SDN/WSN. Therefore, we identified common requirements from secure WSN frameworks (e.g., [26, 29]), taking into account the WSN limitations, such as the small frame sizes and constrained communication. This section presents the resulting requirements that guided our investigation. Note that in Section 2 we also used criteria that are not requirements but are interesting from a project decision standpoint and well suited to compare our framework against the literature review. Furthermore, oftentimes network protocol design postpones security related issues to a second iteration or protocol version. This may increase the challenge of providing security services or require changes to the initial design. Therefore, we devised a software defined framework with security in mind.

The framework provides the following essential security services: node admission, authenticity, confidentiality, and integrity. Secure node admission is the process that allows new nodes to be authenticated and trusted by all other nodes in the network, usually enabled by asymmetric cryptographic primitives. In contrast, data authenticity, confidentiality and integrity are achievable through symmetric cryptography, such as an Authenticated Encryption with Associated Data algorithm (AEAD). Symmetric key distribution is essential to allow using an AEAD, achieved in our solution by a key agreement procedure.

The constrained wireless network traits hinder the use of other software defined approaches, such as OpenFlow. It is highly desirable that control packets fit in the link-layer transmission unit, typically 127 bytes. In addition, the number of control packets should be kept to a minimum, as they are transmitted in-band (opposed to wired networks, which usually employ out-of-band control).

In the next section we present the framework architecture and how it meets these requirements.

4. System Architecture

The secure SDN architecture for WSN (WS^3N, from Wireless Secure SDN-based Sensor Networks) is described in this section. The software design for sensor nodes and the network controller is presented and other two entities are introduced,

the Key Generation Center (KGC) and the authorization server. Southbound protocol messages are listed along with their purposes. Finally, we describe the node admission procedure and symmetric key establishment steps.

We adopted the flow forwarding model, similar to TinySDN [11]. A flow is identified by a unique identifier (FlowID) and is defined as one or more paths from source nodes to possibly multiple destinations assigned to that flow.

A flow is usually coupled with an application service and may be installed in advance on sensor nodes or it could be dynamically assigned by an application manager. We consider the latter case is out of scope and has been studied previously [33]. We assume that the SDN controller knows the destination of each FlowID and thus it is able to calculate flow paths with a global view of the network.

4.1. Security Notes. A secure communication channel between the controller and the SDN nodes is of high importance. Malicious nodes could inject fake packets into the network causing damage like denial-of-service by changing flow definitions (e.g., draining nodes' batteries), or obtain sensitive or confidential information if upper layers do not offer adequate protection.

Also, applications may want to communicate securely in an end-to-end manner. Link-layer security is not sufficient for that purpose, since data is decrypted and re-encrypted at each hop. On the other hand, easy-to-use SDN APIs could provide security at network level, taking the additional key management burdens away from the upper layers. It is assumed that the SDN controller is trusted by the network users since it will be the responsible for configuring all the network rules.

Symmetric key distribution is a challenging problem in the context of WSN. Traditional Public Key Infrastructure (PKI) schemes used in the Internet are not suitable due to large certificates and heavy processing requirements [32].

Authenticated Key Agreement (AKA) solutions are a more lightweight approach. The literature offers identity-based schemes as an alternative; however it has the drawback of key escrow. In addition, security flaws were discovered in special pairing-friendly curves required by those schemes [32].

The certificate-less paradigm could be an alternative, but it was shown that it is not the most efficient solution. Therefore we choose an implicitly certified paradigm, the iSMQV protocol, in particular [32], which combines both escrow-less capability and performance efficiency.

The obtained symmetric key may be used with an Authenticated Encryption with Associated Data algorithm (AEAD) to provide confidentiality, integrity, and authenticity services, i.e., essential security requirements in the context of WSN and the Internet of Things (IoT) [34].

4.2. System Entities. There are three special entities in our SDN architecture, namely, SDN controller, Key Generation Center (KGC), and authorization server. It is assumed that these three entities are able to communicate with each other regardless of WSN connectivity.

The controller is a commonplace entity across SDN architectures, responsible for managing the network control plane in a centralized fashion. It defines the routing behavior by dynamically installing forwarding rules in SDN nodes flow tables.

The controller executes algorithms over a local representation of the network state (such as topology, link state, and remaining energy) in order to calculate suitable flow table entries according to current routing policy. The resulting rules are informed to the nodes by the southbound protocol described in Section 4.4.

The controller may also be provided with a northbound interface, to enable external applications to influence network behavior, but we consider such protocol out of scope, as we focus on the wireless network segment.

The Key Generation Center (KGC) is a trusted entity responsible for generating authenticated key pairs for the SDN entities by using an Implicitly Certified Authenticated Key Agreement protocol (such as iSMQV [32]). Note that although the KGC participates in the key generation procedure, the key pairs are not subject to escrow.

The Authorization Server is responsible for receiving and answering authorization requests from joining SDN nodes. Node access is granted upon checking its credentials in a database or requesting user interaction. We choose the node credential to be based on AKA protocol parameters. When a node is authorized, its ID/Credential is also stored in a local database called authorization list.

The other entities are SDN-enabled network nodes, responsible for data plane execution by consulting local flow tables that are managed by the SDN controller. An SDN-enabled node simply forwards packets according to rules defined by the controller and may also create packets according to its application. SDN-enabled nodes use predefined FlowID values to reach the three special entities in the system, namely, "To KGC," "To Auth," and "To Controller" FlowIDs.

4.3. Software Architecture. The secure SDN software architecture is based on RFC 7426 guidelines and terminology [35], with the addition of security services, such as node admission and confidentiality, as aforementioned in Section 4.1.

Figure 2 displays the software architecture envisioned for SDN-enabled nodes. It aims at clearly defining the system boundaries and its submodules.

The SDN module provides an interface with the application layer for data transmission and data reception services. On the other hand, the SDN module relies on lower layer services, including link quality indicators and physical data transmission/reception.

The main SDN submodule is called **Decision Module**. It handles packet queuing, incoming/outgoing packets, conversion between internal data structures, and raw packets and coordinates the other submodules. For instance, the Decision Module converts the raw packet to the SDN module internal data structures (*Raw packet Handler* internal submodule). Next, the packet is forwarded according to its type to the operational plane or forwarding plane.

The **forwarding plane** handles data packets according to the actions defined in the flow table; for example, the

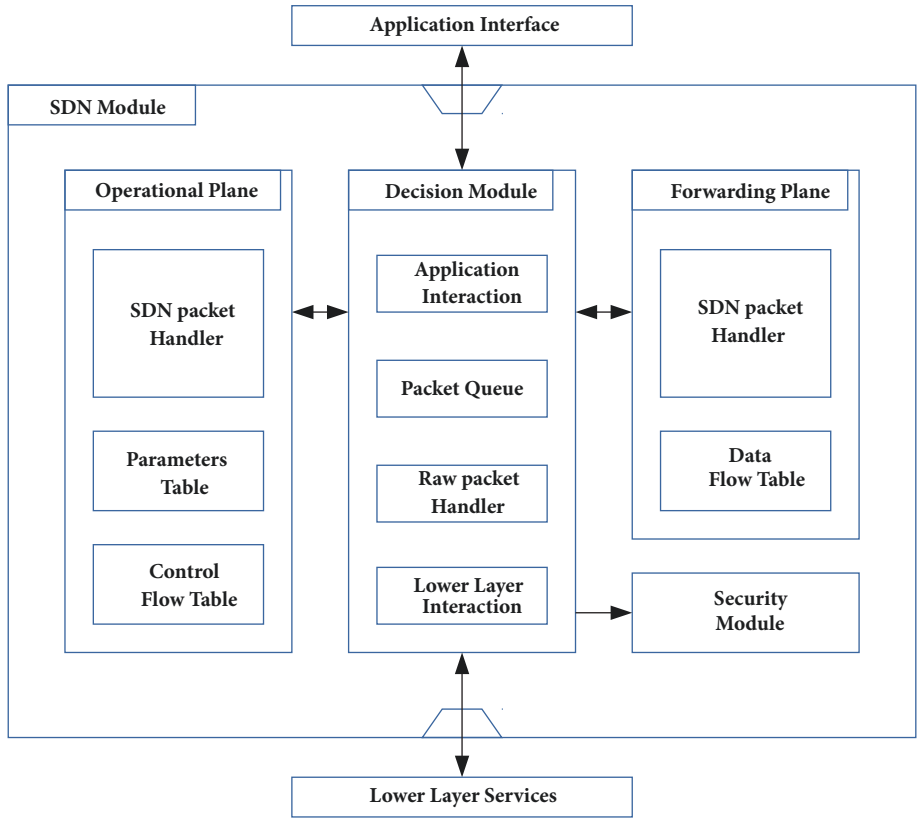


Figure 2: SDN-enabled node software architecture.

packet may be forwarded to the next hop, dropped, or sent to the upper layer in case it has reached its destination. In case there is no matching rule, it is responsible for querying the controller about how to handle the packet. Unmatched packets are stored until the response arrives.

The **flow table** itself is a passive data structure that stores packet forwarding information, such as actions associated to each FlowID. If the packet contains any layer 3 headers other than the SDN-specific headers described in Section 4.4, the forwarding plane should be able to extract such information through specialized submodules.

Upon a packet reception, the **Decision Module** checks whether it is encrypted or not. Encrypted packets must be decrypted before further processing, which is accomplished by the **Security Module**. Once the plaintext packet is obtained, it is processed by the SDN layer or sent to upper layers.

The **Security Module** handles all cryptographic operations, such as key agreement procedures, data encryption/decryption, and stores sensitive information. The objective of this module is to provide an interface to the **Decision Module** that eases handling the underlying cryptographic algorithms, providing high level primitives such as *encrypt packet*, *decrypt packet*, and *request symmetric key*. The internal database stores the node asymmetric key pair, temporary key agreement values, and symmetric keys that are eventually exchanged with other nodes. Section 4.1 discusses the classes of algorithms which are suitable to perform each operation, while Section 5.1 contains the specific choices of algorithms, their parameters, and security level.

The **operational plane** module is responsible for managing and updating relevant information about the node state. For instance, it may check neighboring status or the amount of available energy, and report the controller through specialized control packets. These parameters are stored in a **parameter table** within the operational plane and may also be polled by the controller or other SDN-enabled nodes. An auxiliary protocol may be used to fulfill some of these tasks. For example, TinySDN uses the Collection Tree Protocol (CTP) so nodes can reach the controller and gather link quality information. In our implementation, we use IPv6 ND as the underlying neighbor discovery protocol [36, 37]. SDN control packets destined to specific nodes are handled by the operational plane by means of a **control flow table**.

The SDN controller plays a central role in the SDN architecture, since it takes care of all network related decisions. It relies on an SDN module to provide connectivity with network nodes. It may provide an interface to the user, for example, to show current routes or battery levels, and an API to allow configuration by external applications.

In addition to a **Decision Module**, the SDN controller itself is composed of the **management and control planes**, as displayed in Figure 3. The **Decision Module** handles the interface with the SDN module by transforming between internal and external data formats and maintains a packet

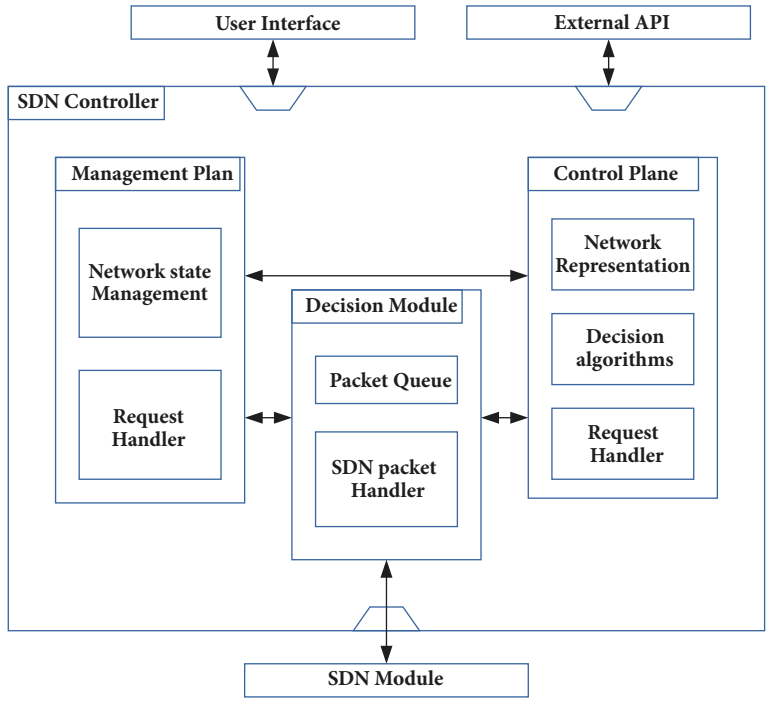


Figure 3: SDN controller high level software architecture.

queue. It also decides whether a request is destined for the management plane or the control plane.

The **management plane** is the module responsible for maintaining network information up-to-date; i.e., it feeds the network representation of the **control Plane** with new information. It interacts mostly with the **Operational Plane** of the SDN-enabled nodes by sending information request packets and receiving node status updates. In order to avoid unnecessary overhead, some node parameters may be estimated by the management plane instead of actually querying the network (for instance, the node battery level [38]).

The **control plane** runs decision algorithms in order to determine the best routes for the network. The flow configuration may be installed in the network nodes upon a request (e.g., no matching entry in flow table) or a reconfiguration may be triggered due to network changes (for instance, link quality changes).

The routing algorithms use information consolidated by the network representation submodule. Many different strategies may be employed; for instance, it is possible to avoid low battery nodes or to prioritize nonbattery-powered nodes. However, the actual routing approach is not part of WS^3N framework scope. An example of network representation is illustrated in Figure 4, including links between nodes, active flows, and node data.

Authorization server and KGC architectures are simpler than the nodes and controller. The authorization server only needs a database of authorized nodes and the capability of informing other network elements of their authorization status. The KGC is responsible for bootstrapping nodes asymmetric keys; therefore it only receives bootstrapping

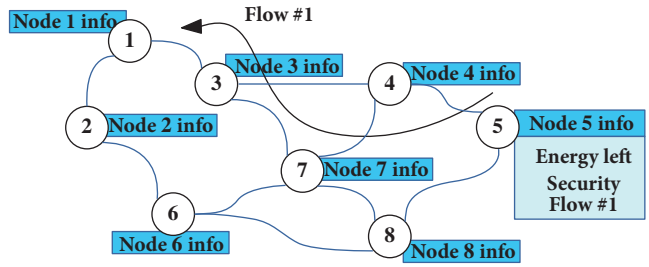


Figure 4: Network representation example (node data is exemplified for node 5).

requests and answer the parameters accordingly. It does not hold temporary values or symmetric keys.

4.4. Southbound Protocol Messages. The southbound protocol is composed of a set of message types to perform SDN network tasks. The framework specification is flexible to adopt any network layer packet format (as long as the forwarding plane implementation is able to deal with it), but we adopt the convention of RFC 4944 [39] that all packets start with an 8 bit field that defines the packet type, as indicated in Figure 5.

The first two bits are reserved for 6lowPAN compatibility and should be set to zero. The third bit is reserved and should be set to zero. The fourth bit indicates whether the packet is encrypted or not. The remaining bits indicate one of the 12 types of SDN packets, namely, data, flow request, flow set up, status response, status request, neighboring information, iSMQV bootstrap request, iSMQV bootstrap response,

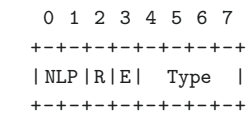


Figure 5: Packet type field. NLP stands for Not a Low PAN packet. R is reserved. E indicates encryption. Type is the packet type identifier.

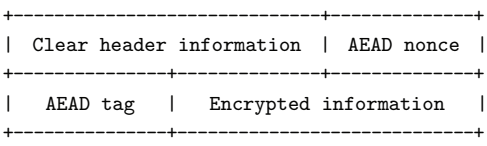


Figure 6: Generic authenticated packet format.

iSMQV key agreement request, iSMQV key agreement response, authorization request, or authorization response.

The SDN data packet is used to send application data, its fields relate to addressing information (FlowID and source address).

Flow request and flow set up packets are used for control flow and data flow configuration. The main information needed for requesting a new flow is addressing information (network address or FlowID). Flow request packets must be sent to *To Controller* FlowID. The flow set up packet configures nodes flow tables and are sent in response to flow requests. It contains packet matching rules, action to execute upon packet match and action parameters.

The controller will be able to calculate routes only if information about network topology is available. This information is transmitted to the controller by neighboring information update packets. The FlowID should be set to *To Controller* and the address of the node must also be present, followed by a list of neighbors and their status (link quality metric, such as ETX).

However, the controller may use information other than neighboring for route calculation. Such information may be battery level, physical characteristics, or security services availability. This information may be gathered by status request/response packets. They contain addressing information and a sequence of information IDs. In the case of status response messages, the values of such information are also transmitted.

The remaining packets are related to security operations. The authorization request packet fields are FlowID (should be *To Auth Server*), source address, node security configuration, and the node authentication ID. After the SDN-enabled node is properly authorized, the authorization server replies with an authorization response. It is destined to the requesting node and contains the authorization status.

The next step is to establish an asymmetric key pair according to the AKA protocol. An iSMQV bootstrap request message is used by SDN-enabled nodes to start the key setup procedure. The fields of this packet are the FlowID (must be *To KGC* FlowID), node address, and necessary parameter for executing the protocol. The KGC replies with an iSMQV bootstrap response message, the contents of which are, in addition to the node address, the parameters necessary for finishing the key authentication procedure and establishing the asymmetric key. Similarly, key agreement request and response packets are used to establish a symmetric key, filled with addressing fields and protocol parameters.

If security services are enabled, flow configuration, status, neighbor information and data packets have two additional fields. The first one is the encryption nonce related to the AEAD algorithm and is used for providing semantic security; the second one is the AEAD tag (for authentication). Those fields are depicted in Figure 6.

4.5. Node Admission and Key Bootstrap Messages. This subsection describes the node admission and key bootstrap procedure from a network point of view. The following steps are executed, as illustrated in Figure 7:

(1) Node transmits authorization request packet;

(2) Authorization module recognizes the node and reports back an authorization response message;

(3) In the meanwhile, the controller may set up the reply flow in its control flow table and authorization module flow table;

(4) Intermediate nodes relay the authorization response message to the node;

(5) Node receives the authorization response message and fires an iSMQV bootstrap request to the KGC;

(6) KGC receives the packet and checks the credentials within the authorization server;

(7) KGC replies to the node iSMQV bootstrap response;

(8) Intermediate nodes relay the response message to the node;

(9) The node is now able to establish authenticated secure communication with the Controller and is able to send flow requests;

(10) After the key exchange, the controller adds the node to the list of secure nodes.

4.6. Node Availability and Message Delivery. It is important to define the WS^3N behavior in boundary conditions such as permanent or temporarily unavailability.

The underlying neighbor discovery algorithm is responsible for detecting the absence of nodes. In our implementation, Neighbor Unreachability Detection (NUD) is used with this purpose [36]. In the case a node becomes unavailable (for example, due to lack of energy, or its removal), all of its neighbors will detect its absence from the current neighborhood and inform the controller.

Next, the unavailable node will be removed from the flow tables and all packets destined to it will be dropped. If the unavailable node was a relay node to other destinations, the controller will recalculate and assign the new routes according to its representation of the network.

If a node is temporarily unavailable (for example, due to duty cycling or reboot), any messages sent to it will be lost. The first mechanism to recover from such failures is

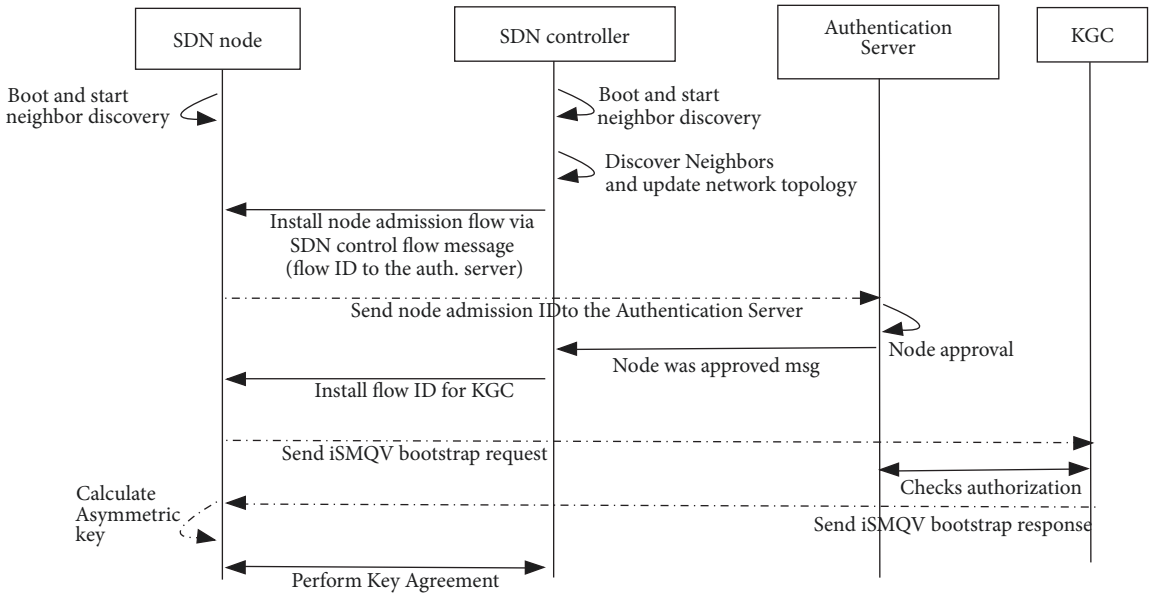


Figure 7: Node admission and key bootstrap messages.

the retransmission mechanisms imbued at the MAC layer. However, if the packet requires reliability at the network layer, retransmissions are triggered after a timeout. In addition, the loss of packets decreases the link quality metric, possibly triggering route recalculations and alleviating congestion around the designated node.

5. Performance Analysis

5.1. Architecture Implementation. This section presents the WS^3N implementation described in Section 4 and the experiments used to evaluate the performance of the framework. The SDN-enabled node implementation is based on Contiki OS [40], a prominent open-source OS for WSN and IoT. We were able to compile and test on two platforms: wismote (MSP430-based) and CC2650 (ARM-based) binaries.

The SDN layer was implemented as a "network driver," following Contiki stack design. An intermediate layer between MAC and SDN was introduced to allow switching packets to the SDN or IPv6 stack according to the packet type field, allowing both stacks to coexist. In fact we use IPv6 neighbor discovery as the underlying ND protocol. RPL routing protocol was also enabled, in order to demonstrate cooperability and to aid keeping neighborhood information up-to-date.

The SDN controller was implemented on Linux using C language. Both authorization server and KGC were embedded in the same piece of software for simplicity. The controller calculates routes using Dijkstra algorithm over a network graph weighted with ETX metric [41]. Route recalculation is triggered automatically on a predefined weight threshold.

Both SDN-node and controller use RELIC [42], a C library for elliptic curve arithmetic, for implementing the iSMQV key agreement protocol (compiled for different platforms). We choose an elliptic curve providing 80-bit security level (namely SECP160r1 NIST curve [43]). LetterSoup AEAD [44] was employed to provide symmetric security services.

5.2. Method and Scenarios. In order to conduct the performance analysis experiments, we used Cooja simulation/emulation environment [45]. Cooja is both a simulator and emulator as it emulates real Contiki code, using a third-party tool named MSPSim, and simulates the radio environment.

The metrics of interest are packet delivery rate, regarding both data and control packets, control overhead, and time that it takes to perform important framework operations. These time metrics are calculated from the moment of request to arrival of response, with respect to the following operations: obtaining authorization, asymmetric key bootstrap, performing key agreement with the controller, and obtaining application data flow. The time to obtain the controller route is also measured, but using the node boots time as reference.

The nodes were arranged in a square grid topology, the controller placed in the center. We vary the number of network nodes from 9 to 64, including the controller. Each node boots at a random time, uniformly distributed from 0 to 10 seconds, to prevent artificial synchronization and increase of packet loss due to collisions.

In addition to the number of nodes, we also varied the MAC protocol primitive. First we used Contiki nullrdc, an always-on MAC approach. The second set of simulations used ContikiMAC protocol, which turns the radio on periodically to check for incoming packets [46]. We configured ContikiMAC channel check rate to 8 Hz.

Every node in the network produced data packets at every 5 seconds towards a common sink, which is placed two hops away from the controller. Data packets are transmitted for 20 minutes and only after the network is established to avoid data

Table 2: Simulation parameters summary.

Parameter	Values
Number of nodes	9, 25, 36, 49, 64
Data traffic rate	one packet every 5 s per node
Data traffic time	20 minutes
Topology	grid (controller at center)
MAC protocol	Nullrdc, ContikiMAC (8 Hz)

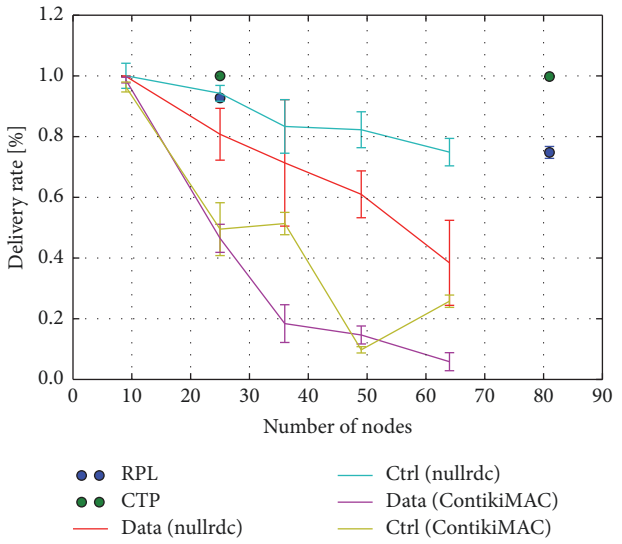


Figure 8: Packet delivery ratio results.

and control packets interference. A summary of simulation parameters is presented in Table 2.

5.3. Results. Each scenario was simulated 10 times, to obtain statistical relevance. Every graph in this section presents 99% confidence intervals.

The first metric to discuss is packet delivery ratio, exhibited in Figure 8. We observe that delivery rates are fairly high for small to medium sized (up to 36 nodes) networks and nullrdc MAC approaches, ranging from 70% to 100% for data and from 85% to 100% for control packets.

As the number of nodes increases, extra control traffic causes packet loss due to collision and buffer overflow (limited to 8 packets at SDN layer). Another issue is the limited capacity of node flow tables (10 in the simulations), which causes recurrent flow table overwriting. Those limits were imposed solely by the memory capacity of the experimental node device and are not limited conceptually by design. For instance, consider the 49-node scenario: each controller neighbor would need 11 control flow table entries, if routes are evenly distributed. At the time a packet needs to be routed to the 11th node, the oldest flow table entry will be overridden. Therefore, the next time a packet is routed to that overridden destination, another flow request will be issued, increasing control overhead.

Finally, nodes start to transmit data packets immediately after receiving a shared-key with the controller. Therefore, all

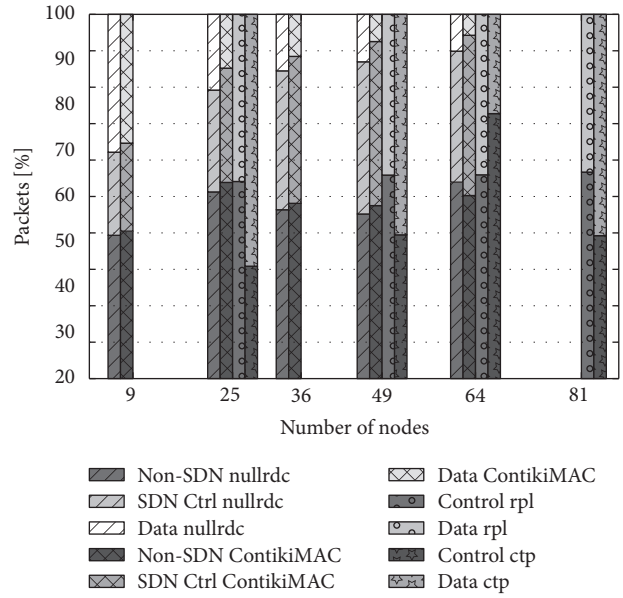


Figure 9: Control overhead.

transmission attempts fail until the data flow setup message arrives, which can take several seconds, as discussed in the end of this section (Figure 14).

Despite the inferior performance, larger networks are still able to operate with nullrdc. Operation with ContikiMAC is not so encouraging, as packet delivery ratio may get below 10% in the 64-node scenario. For the case of small networks, i.e., 9-node scenario, results indicate nearly 100% delivery. For larger networks, we suggest more robust duty cycling algorithms or a cross-layer mechanism to improve delivery rates, but this discussion is beyond this work.

Figure 8 also shows data delivery rate results for RPL and Collection Tree Protocol from Ogawa et al. [47]. We observe better performance for large networks from these protocols than to SDN. Note that we rely on IPv6 ND as the underlying topology discovery mechanism. Therefore, in addition to SDN control overhead, IPv6 and RPL control overhead are also present, lowering our expectation over delivery rate.

The percentage of data and control (further classified in SDN and non-SDN) packets is displayed in Figure 9 (for 9, 25, 36, 49, and 64 nodes). RPL and CTP results are extracted from Ogawa et al. (25, 49, 64, and 81 nodes) [47]. We noticed a large percentage of control packets in all scenarios, highlighting the non-SDN control packet fairly high rate (40% to 55%). Nevertheless, it is comparable to results from Ogawa et al. [47]. This suggests that a more efficient underlying neighbor discovery protocol could be used instead of IPv6 ND. Also, as expected, the proportion of control packets raises as network size increases.

In fact, nodes farther from the controller may not be able to fully join the network due to low packet delivery ratio, as illustrated in Figure 10. Scenarios with 9, 25, and 36 nodes are able to fully connect independently of the MAC primitive. Regarding larger nullrdc networks, nodes are likely to be able to get a controller key in the simulation timespan,

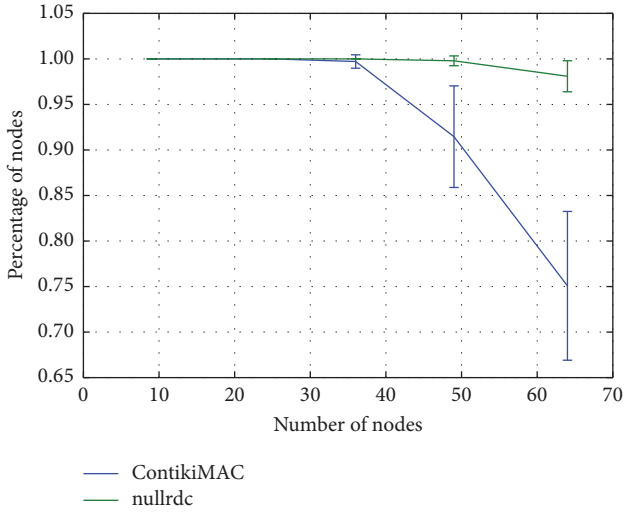


Figure 10: Percentage of nodes that completed key agreement procedure.

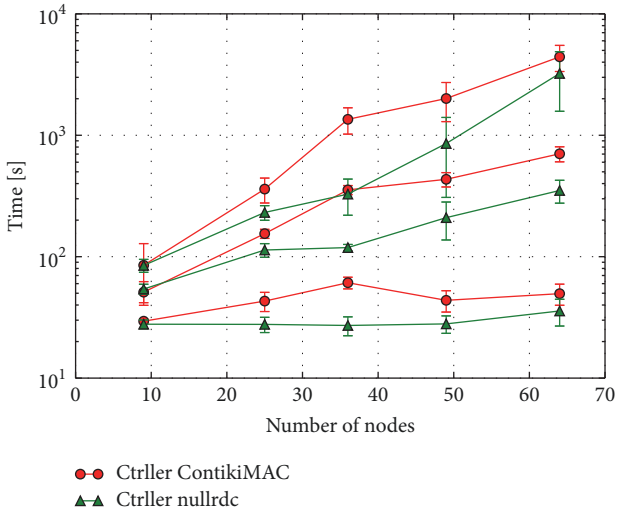


Figure 11: Time to obtain controller route from boot. Curves represent minimum, average, and maximum values.

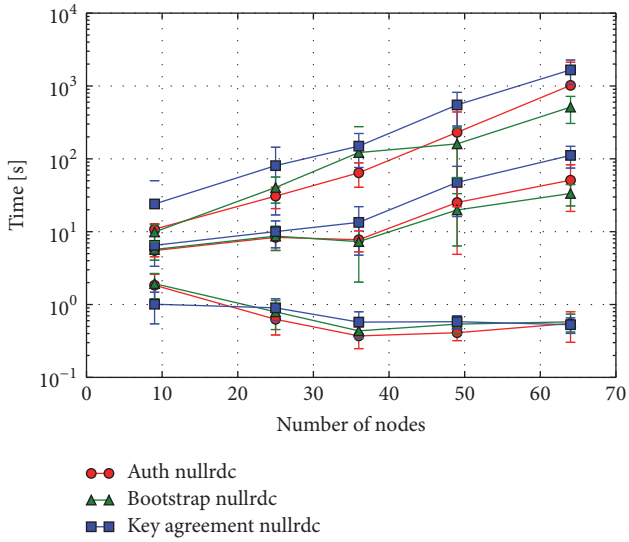


Figure 12: Time to perform initial procedures with nullrdc MAC—authorization, bootstrap, and controller key agreement. Curves represent minimum, average, and maximum values.

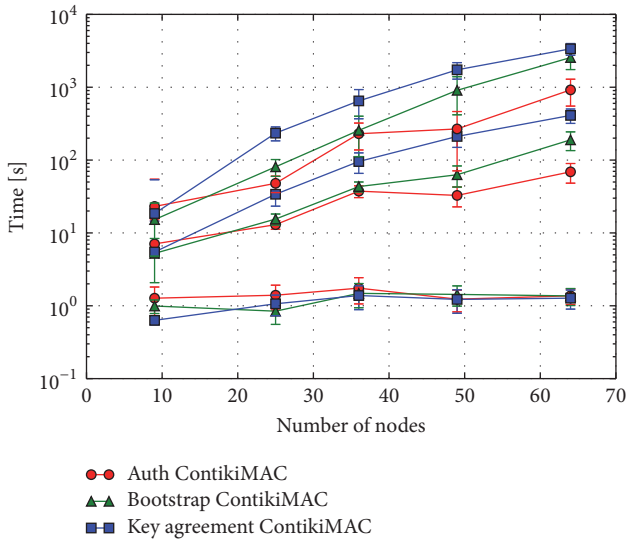


Figure 13: Time to perform initial procedures with ContikiMAC—authorization, bootstrap, and controller key agreement. Curves represent minimum, average, and maximum values.

as opposed to ContikiMAC. This happens due to the larger number of hops to reach a distant node and the reduced delivery probability at the MAC layer.

The effect of duty cycling can also be observed in the time to obtain controller route, displayed in Figure 11 (note that this time is computed from node boot time). For all scenarios above 9 nodes, ContikiMAC takes about twice the time. It is also interesting to observe that, for either MAC, the minimum time does not change significantly, which means that the nodes near the controller are always able to get this route timely (around 30 seconds for nullrdc and 50 seconds for ContikiMAC).

As the network size increases, the peripheral nodes take longer to obtain the controller route, causing the increasing trend both in the maximum and average time values. The nullrdc average values increase in an exponential trend from 55 to 350 seconds (note that the graph is in logarithmic scale), while ContikiMAC ranges from 50 to 700 seconds.

The delay between the request and the response for mandatory procedures in our SDN framework, namely, node authorization, asymmetric key bootstrapping, and controller key agreement, is shown in Figures 12 and 13 for nullrdc and ContikiMAC.

Observe that each procedure takes about the same amount of time to be accomplished. This is because all tasks are similar in nature: they are basically a request to and an answer from the controller. Note that, in this set of experiments, controller, KGC, and authorization server are all

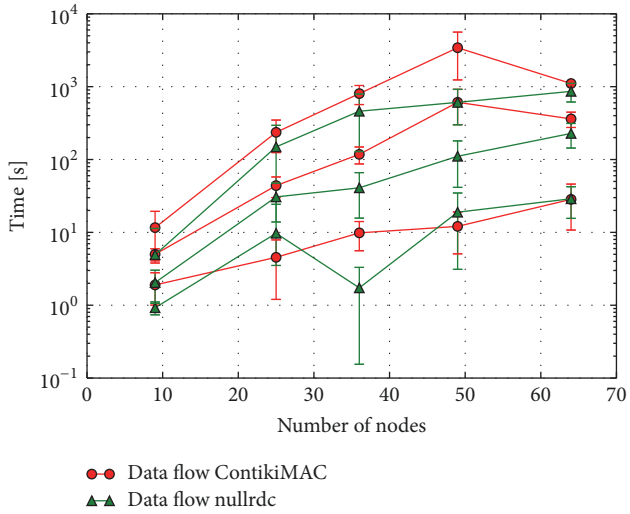


FIGURE 14: Time to obtain application flow table entry. Curves represent minimum, average, and maximum values.

in the same computer. Each operation takes a little longer, due to two main reasons: the latter operations are more complex and take more time and the network gets busier as nodes join the network. Note that all minimum times fluctuate around 1 second for both nullrdc and ContikiMAC.

Also, as expected, nullrdc outperforms ContikiMAC in all scenarios larger than 9 nodes regarding time metrics. The 9-node scenario presents similar values for both MAC primitives; however ContikiMAC presents larger variability. Nullrdc average delay increases slowly in the first three scenarios, with values around 10 seconds. In the last two scenarios, node flow tables get overflown frequently, causing new requests, thus lowering performance (up to 200 seconds on average). ContikiMAC performance drops at the second scenario due to the duty cycle restrictions, obtaining response time of a few minutes.

Another characteristic is that maximum values are notably larger than average values (around 2.5 times for smaller and 10 times for larger networks). This is expected since nodes outlying the controller require more hops to reach it.

The last metric regards data flow request, depicted in Figure 14. This metric analysis is similar to the previous metrics (Figures 12 and 13), since the data flow request is similar in nature to the other operations: it is a request to as a response from the controller. This request can only be executed after a symmetric key has been properly established between the node and the controller.

As the time to obtain the data flow route increases, more data packets are dropped since the transmission queue eventually overflows. This effect is more intense as distance from controller increases. Therefore, the time to obtain data flow helps explain packet delivery ratio results presented in Figure 8. However, these drops only occur at the beginning of the data transmission. Therefore, on the long run, the packet delivery ratio should increase.

6. Conclusions

Security and flexibility are key elements to enable the modern Wireless Sensor Networks and the Internet of Things. We argue that Software Defined Networks are capable of providing the desired flexibility. However, security was not a concern in the literature related to SDN application in constrained devices.

We proposed WS^3N, a secure SDN framework that is able to handle node admission and symmetric key distribution. Cryptographic protocols and algorithms were combined with an SDN protocol to provide such services. Packet headers were crafted to fit the IEEE 802.15.4 frame size and to be compatible with 6LoWPAN networks.

We validated our proposal through an experimental setup, designed to reproduce typical IoT scenarios. Results show that initial network setup overhead, inherent to SDN, takes from 1 to 5 minutes on average, depending on the scenarios and configuration. Delivery rate results show the applicability of the proposed framework up to medium size networks, which suffice to most personal area applications (i.e., domestic usage).

Future work includes replacing the current IPv6 neighbor discovery by a more efficient algorithm and adding cooperation between MAC duty cycling and routing process in order to improve delivery rates and energy efficiency. Moreover, the establishment of flows based on decisions considering metrics such as the available (estimated) battery per sensor, data exchange rate of the applications, and type of cryptographic parameters (e.g., distinct blocks sizes) is also interesting to be investigated through new experiments.

Data Availability

The company that sponsored this project required a Non-Disclosure Agreement to be signed, and the authors are not allowed to publicly distribute the implementation they developed. If code or simulation traces are requested, they will forward the request to the company.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was supported by *Fundação para o Desenvolvimento Tecnológico da Engenharia* (FDTE) under Grant no. 1450 sponsored by LG Electronics through *Lei de Informática/MCTI*. C. B. Margi is supported by CNPq Research Fellowship no. 307304/2015-9. The authors thank their colleagues that contributed to the prototype implementation: Tiago R., Tiago S., Roberto, Artur, and F. Leno.

References

[1] F. Hu and N. K. Sharma, "Security considerations in ad hoc sensor networks," *Ad Hoc Networks*, vol. 3, no. 1, pp. 69–89, 2005.

[2] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for

sensor networks," in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MOBICOM '00)*, pp. 56–67, August 2000.

[3] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys '09)*, pp. 1–14, New York, NY, USA, November 2009.

[4] T. Winter, P. Thubert, A. Brandt et al., "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," RFC Editor RFC6550, 2012.

[5] N. McKeown, T. Anderson, H. Balakrishnan et al., "OpenFlow: enabling innovation in campus networks," *Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[6] O. Flauzac, C. Gonzalez, A. Hachani, and F. Nolot, "SDN based architecture for IoT and improvement of the security," in *Proceedings of the 29th IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA '15)*, pp. 688–693, March 2015.

[7] V. R. Tadinada, "Software Defined Networking: Redefining the Future of Internet in IoT and Cloud Era," in *Proceedings of the 2014 2nd International Conference on Future Internet of Things and Cloud (FiCloud)*, pp. 296–301, Barcelona, Spain, August 2014.

[8] A. Mahmud and R. Rahmani, "Exploitation of OpenFlow in wireless sensor networks," in *Proceedings of the International Conference on Computer Science and Network Technology (ICCSNT '11)*, pp. 594–600, Harbin, China, December 2011.

[9] T. Luo, H.-P. Tan, and T. Q. S. Quek, "Sensor openflow: enabling software-defined wireless sensor networks," *IEEE Communications Letters*, vol. 16, no. 11, pp. 1896–1899, 2012.

[10] 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 *Proceedings of the 34th IEEE Annual Conference on Computer Communications and Networks, IEEE INFOCOM 2015*, pp. 513–521, May 2015.

[11] B. T. de Oliveira, C. B. Margi, and L. B. Gabriel, "TinySDN: Enabling multiple controllers for software-defined wireless sensor networks," in *Proceedings of the 2014 6th IEEE Latin-American Conference on Communications (LATINCOM)*, pp. 1–6, November 2014.

[12] G. C. C. F. Pereira, R. C. A. Alves, F. L. da Silva, R. M. Azevedo, B. C. Albertini, and C. B. Margi, "Performance evaluation of cryptographic algorithms over IoT platforms and operating systems," *Security and Communication Networks*, vol. 2017, Article ID 2046735, 16 pages, 2017.

[13] V. Kawadia and P. R. Kumar, "Experimental investigations into TCP performance over wireless multihop networks," in *Proceedings of the Proceeding of the 2005 ACM SIGCOMM workshop, E-WIND '05*, pp. 29–34, New York, NY, USA, August 2005.

[14] B. V. Prasad and S. Salman Ali, "Software – defined networking based secure rout-ing in mobile ad hoc network," *International Journal of Engineering & Technology*, vol. 7, no. 1.2, p. 229, 2017.

[15] M. Alqallaf and B. Wang, "Software defined collaborative secure ad hoc wireless networks," in *Proceedings of the 2015 International Conference on Collaboration Technologies and Systems (CTS)*, pp. 196–203, Atlanta, GA, USA, June 2015.

[16] S. Chakrabarty, D. W. Engels, and S. Thathapudi, "Black SDN for the Internet of Things," in *Proceedings of the 2015 IEEE 12th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pp. 190–198, Dallas, TX, USA, October 2015.

[17] A. De Gante, M. Aslan, and A. Matrawy, "Smart wireless sensor network management based on software-defined networking," in *Proceedings of the 27th Biennial Symposium on Communications (QBSC '14)*, pp. 71–75, Kingston, Canada, June 2014.

[18] H. Akram and A. Gokhale, "Rethinking the Design of LR-WPAN IoT Systems with Software-Defined Networking," in *Proceedings of the 2016 International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pp. 238–243, Washington, DC, USA, May 2016.

[19] Y. Jararweh, M. Al-Ayyoub, A. Darabseh, E. Benkhelifa, M. Vouk, and A. Rindos, "SDIoT: a software defined based internet of things framework," *Journal of Ambient Intelligence and Humanized Computing*, vol. 6, no. 4, pp. 453–461, 2015.

[20] A. El-Mougy, M. Ibnkahla, and L. Hegazy, "Software-defined wireless network architectures for the Internet-of-Things," in *Proceedings of the 2015 IEEE 40th Local Computer Networks Conference Workshops (LCN Workshops)*, pp. 804–811, Clearwater Beach, FL, October 2015.

[21] K. S. Sahoo, B. Sahoo, and A. Panda, "A secured SDN framework for IoT," in *Proceedings of the 2015 International Conference on Man and Machine Interfacing (MAMI)*, pp. 1–4, Bhubaneswar, India, December 2015.

[22] O. Salman, I. Elhajj, A. Chehab, and A. Kayssi, "Software Defined IoT security framework," in *Proceedings of the 2017 Fourth International Conference on Software Defined Systems (SDS)*, pp. 75–80, Valencia, Spain, May 2017.

[23] R. C. A. Alves, D. Oliveira, G. N. Segura, and C. B. Margi, "IT-SDN: Improved architecture for SDWSN," in *XXXIV Simpósio Brasileiro de Redes de Computadores*, 2017, http://www.larc.usp.br/~cbmargi/it-sdn/.

[24] C. B. Margi, R. C. A. Alves, and J. Sepulveda, "Sensing as a service: Secure wireless sensor network infrastructure sharing for the internet of things," *Open Journal of Internet Of Things (OJIOT)*, vol. 3, no. 1, pp. 91–102, 2017.

[25] IEEE. 802.15.4-2011 - IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs), 2011.

[26] C. Karlof, N. Sastry, and D. Wagner, "Tinysec: a link layer security architecture for wireless sensor networks," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pp. 162–175, Baltimore, MD, USA, November 2004.

[27] M. Luk, G. Mezzour, A. Perrig, and V. Gligor, "MiniSec: a secure sensor network communication architecture," in *Proceedings of the 2007 6th International Symposium on Information Processing in Sensor Networks*, pp. 479–488, New York, NY, USA, April 2007.

[28] L. Casado and P. Tsigas, "ContikiSec: a secure network layer for wireless sensor networks under the Contiki operating system," in *Identity and Privacy in the Internet Age*, vol. 5838 of *Lecture Notes in Computer Science*, pp. 133–147, Springer-Verlag, Berlin, Germany, 2009.

[29] N. Bandirmali and I. Erturk, "WSNSec: a scalable data link layer security protocol for WSNs," *Ad Hoc Networks*, vol. 10, no. 1, pp. 37–45, 2012.

[30] A. L. M. Neto, A. L. Fernandes, I. Cunha et al., *AdC: um Mecanismo de Controle de Acesso para o Ciclo de Vida das Coisas Inteligentes*, 2016.

[31] O. R. M. Boudia, S. M. Senouci, and M. Feham, "Secure and efficient verification for data aggregation in wireless sensor networks," *International Journal of Network Management*, vol. 28, no. 1, 2017.

[32] M. A. Simplicio Jr., M. V. M. Silva, R. C. A. Alves, and T. K. C. Shibata, "Lightweight and escrow-less authenticated key agreement for the internet of things," *Computer Communications*, vol. 98, pp. 43–51, 2016.

[33] Silva. Henrique, Hahn. Andre Pereira, Yuka. Solano, T. Bruno, and B. Cíntia, "WARM: WSN application development and resource management," in *XXXIV Simpósio Brasileiro de Telecomunicações e Processamento de Sinais*, Sociedade Brasileira de Telecomunicações, Santarém, Brazil, 2016.

[34] L. Atzori, A. Iera, and G. Morabito, "From 'smart objects' to 'social objects': the next evolutionary step of the internet of things," *IEEE Communications Magazine*, vol. 52, no. 1, pp. 97–105, 2014.

[35] E. Haleplidis, K. Pentikousis, S. Denazis, J. Hadi Salim, D. Meyer, and O. Koufopavlou, "Software-Defined Networking (SDN): Layers and Architecture Terminology," RFC Editor RFC7426, 2015.

[36] T. Narten, E. Nordmark, W. Simpson, and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)," RFC Editor RFC4861, 2007.

[37] Z. Shelby, S. Chakrabarti, E. Nordmark, and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)," RFC Editor RFC6775, 2012.

[38] G. A. Segura and C. B. Margi, "Duty cycle based energy management tool for wireless sensor networks," in *XXXIV Simpósio Brasileiro de Telecomunicações e Processamento de Sinais*, Sociedade Brasileira de Telecomunicações, Santarém, Brazil, 2016.

[39] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944 (Proposed Standard), September 2007. Updated by RFCs 6282, 6775.

[40] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki—a lightweight and flexible operating system for tiny networked sensors," in *Proceedings of the 29th IEEE Annual International Conference on Local Computer Networks (LCN '04)*, pp. 455–462, November 2004.

[41] D. S. De Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," in *Proceedings of the the 9th annual international conference*, pp. 134–146, New York, NY, USA, September 2003.

[42] D. Aranha and C. Gouvêa, *RELIC is an Efficient LIbrary for Cryptography*, 2015, http://code.google.com/p/relic-toolkit/.

[43] Certicom Research, "SEC 2: Recommended Elliptic Curve Domain Parameters," in *Standards for Efficient Cryptography*, 2000.

[44] M. A. Simplicio, B. Pedro Aquino, P. S. L. M. Barreto, T. C. M. B. Carvalho, and C. B. Margi, "The Marvin message authentication code and the LetterSoup authenticated encryption scheme," *Security and Communication Networks*, vol. 2, no. 2, pp. 165–180, 2009.

[45] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with COOJA," in *Proceedings of the 31st Annual IEEE Conference on Local Computer Networks (LCN '06)*, pp. 641–648, November 2006.

[46] A. Dunkels, "The ContikiMAC Radio Duty Cycling Protocol," Tech. Rep. T2011:13, Swedish Institute of Computer Science, 2011.

[47] H. Ogawa, B. T. de Oliveira, T. Rodrigues, B. C. Albertini, and C. B. Margi, "Energy consumption and memory footprint evaluation of RPL and CTP in TinyOS," in *XXXIV Simpósio Brasileiro de Telecomunicações e Processamento de Sinais*, Sociedade Brasileira de Telecomunicações, Santarém, Brazil, 2016.