Towards a Software-Defined Network Operating System for the IoT

Angelos-Christos G. Anadiotis*, Laura Galluccio*, Sebastiano Milardo†, Giacomo Morabito* and Sergio Palazzo*
*CNIT Research Unit at University of Catania, Catania, Italy
†Dipartimento di Energia, Ingegneria dell’Informazione e Modelli Matematici (DEIM)
Università degli Studi di Palermo, Palermo, Italy

Abstract—The heterogeneity characterizing the Internet of Things (IoT) landscape can be addressed by leveraging network operating systems (NOSs). Current NOS solutions have been developed for traditional infrastructured networks: they do not take the specific features of fundamental IoT components such as wireless sensor and actuator networks into account. Accordingly, in this paper an innovative integrated network operating system for the IoT which is obtained as the evolution of the Open Network Operating System (ONOS) is proposed. This has been appropriately extended to enhance the recently proposed SDN-WISE platform for supporting SDN in wireless sensor networks and let it interact with standard OpenFlow (OF) switches. The proposed integrated system has been prototyped.

Keywords—Software-defined networking; IoT; Network Operating System;

I. INTRODUCTION

According to most researchers, heterogeneity is one of the major characteristics of the Internet of Things (IoT): several platforms are available which employ technological solutions very different from each others. Software developers willing to use resources belonging to different platforms for their applications have hard time in coping with such heterogeneity. In the computer science domain the problem of heterogeneity has been addressed leveraging the use of operating systems.

Thus, in recent years operating systems (OS)s have been proposed which mask the heterogeneity of IoT devices to application developers. Relevant examples of such OSs in the open source arena are Contiki [1] and RIOT OS [2]. By exploiting the above operating system software developers can implement applications which can be used on large number of hardware platforms.

However, such solutions implement specific protocols at the networking layers, usually based on 6LowPAN [3] and IPv6 [4]. Instead, a large body of literature exists demonstrating that there is no one-fits-all networking solution in resource constrained domains like those included in the IoT. Therefore, solutions are needed which allow to program the networking layers in a platform independent way, which is what Network Operating Systems (NOS) do.

In the wired domain several NOSs have been proposed along the years. Early attempts include 4D [5], RCP [6], SANE [7] and ETHANE [8].

After the introduction of the OpenFlow protocol as the solution for software defined networking (SDN), in [9] the need for a NOS, called NOX, that allows management applications to be written as centralized programs over high-level names, has been demonstrated. ONIX [10] represents a further step in the creation of a NOS. ONIX is a platform on top of which a network control plane can be implemented as a distributed system. Control planes written within ONIX operate on a global view of the network, and use basic state distribution primitives provided by the platform.

Recently TUNOS [11] was proposed to provide open device management, cognitive network status, global network view, virtual forwarding space, applications context management, and general network control APIs designed for user-friendly network programming.

Finally, OpenDaylight [12] is a novel SDN controller that supports management and programmability technologies. Using these technologies it is possible to create different southbound plugins for different protocols like OpenFlow, BGP-LS/PCEP, and NETCONF. The northbound of OpenDaylight supports different types of applications/service functions such as Base Network Functions and Service Functions. The first one is related to topology and device management whereas the second one implements a Forwarding Rules Manager for OpenFlow and a PCEP Service function provider for RSVP-TE tunnels.

The Open Network Operating System (ONOS) [13] has characteristics which are similar to OpenDaylight. ONOS is one of the fundamental pillars of our work and will be described in the following section.

Note that all the above network operating systems cannot cope with the specific characteristics of fundamental IoT components such as wireless sensor and actuator networks. For example, they do not consider the limitations in terms of processing and energy resources of such IoT devices. Therefore, they do not handle duty cycles, data aggregation, value based routing, etc. Instead, sensors and actuators are fundamental ingredients of the IoT ecosystem because they generate incessant streams of data that the IoT can use to improve our lives and our businesses in many ways. Sensors, in particular, offer unprecedented access to “granular data that
can be transformed into powerful knowledge”. Integrated analytics platforms will be used to overcome the data burst and avoid that sensor data will just add information overload and noise escalation. In this paper we will extend ONOS in order to consider such features.

The rest of this paper is organized as follows. In Section II, background information on the ONOS architecture and SDN-WISE is provided. The proposed architecture is discussed in Section III, whereas a software prototype is presented in Section IV. Finally, in Section V, the conclusions are drawn.

II. BACKGROUND

A. ONOS Architecture

Open Network Operating System (ONOS) is an open source, distributed network operating system, which has been implemented for managing network operations following an SDN approach. The managed network elements and operations are abstracted in a high level and decoupled from the underlying network architecture, in order to enable interoperability across heterogeneous networks. Even though its design is strongly influenced by OpenFlow, ONOS follows a layered architecture, depicted in Figure 1, which allows integration of several network management protocols, under the same abstractions. Components belonging in multiple layers are vertically integrated in the context of the so-called subsystems, which are services implementing key system functions. Such functions include the management of devices, links, flow rules, topology and more. Typically, communication from high layer components to lower layer ones is performed using the corresponding APIs (NB/SB), whereas communication from low layer components to the higher layer ones is performed by using events.

The components belonging to the Protocols layer are handling communication with the Network Elements, which are the devices connected with ONOS (e.g. OpenFlow switches). More specifically, the Protocols layer includes the drivers implementing each device communication protocol as well as the association between the devices registered in the system with their driver.

In the southbound (SB) part of ONOS, the Providers layer includes the components responsible for translating the abstractions used in higher layers to network management protocol-specific operations and vice versa. In particular, a provider service receives from its south part low-level events and encapsulates them in the proper data format to deliver them to higher layers, where the corresponding information will be further processed or stored. Higher layer components, are using the SB (Provider) API to invoke actions necessary to translate the high level abstractions to protocol-specific formats before sending them to the network elements. In its northbound (NB) part, ONOS includes the components managing the abstractions of the network elements and the available operations. In particular, abstractions of flow rules, incoming and outgoing packets, network devices, hosts and more are represented by dedicated APIs in the NB (Consumer) API layer. Then, these abstractions are used in the context of the Core layer, which provides access to all information maintained by the system, including the devices, the topology and the links, whereas it offers additional functionality based on this information, such as the calculation of forwarding paths between devices deployed in a certain topology.

Finally, the Applications layer components leverage information regarding the network status, provided through the NB API, and performs complex actions requiring several subsystems, such as packet forwarding. Typical ONOS applications are triggered upon specific events coming from the network elements, then, based on certain criteria, they are generating the appropriate flow rules that they send back to the network elements through the corresponding subsystems.

B. SDN-WISE Protocol

Despite the enthusiasm around SDN, less attention has been paid to the extension of the SDN approach to WSNs, even though WSNs can receive great benefits from such approach. Recently, SDN-WISE [14], a Stateful SDN solution for WSNs, has been introduced. Compared to existing works, it advances the state of the art by adopting a stateful approach, which is useful to reduce the amount of information exchanged between the sensor nodes and the Controller, as also discussed in [15].

SDN-WISE clearly separates the data plane run by the sensor nodes and the control plane executed by a software program, i.e. the Controller, running on a server.

The behavior of sensor nodes is mostly determined by the content of the so-called WISE Flow Table, which is filled with information coming from the Controller. Entries of the WISE Flow Table can be divided in three sections: Matching Rules, Action, and Statistics. In the Matching Rules section, which is used to classify packets, up to three logical conditions can be defined. Each condition can be matched against any byte of the packet. If all conditions are verified, then the rule is satisfied and the node treats
the packet as specified in the Action section. Among the envisioned actions there are drop, forward, modify, etc. For example, in case the action is forward, then the node which the packet should be sent to is specified in the Value field. In the Statistics section, information is stored about the usage of the entry.

Sensor nodes execute a simple protocol which enables them to learn their best next hop towards the sink(s). SDN-WISE sensor nodes collect information about their neighbors and periodically report it to the Controller. Similarly to the OpenFlow functioning, when an SDN-WISE node receives a packet for which none of its entries apply, it sends such packet inside a Rule request message to the Controller. Thanks to the periodic reports received by the sensor nodes, the Controller has a complete vision of the network topology and conditions, which it uses to decide how to respond to the received Rule requests. In this way, the Controller can give rules to sensor nodes in such a way that packets can be routed in the network based on any byte they contain, that is, not only considering the destination address.

III. PROPOSED ARCHITECTURE

The proposed architecture is integrated into ONOS, enabling the seamless integration between SDN-WISE and OpenFlow networks. Figure 2 depicts the components required in each one of the ONOS layers in order to achieve both network flow and device control in WSN. Components marked with dark blue have been developed from scratch; the FlowRule API, marked with light blue, has been extended in order to support SDN-WISE features; the rest of the components marked with gray color have been kept in their original ONOS version, even though interacting with different implementations. In the following sections, the proposed architecture will be explained, by following a top-down approach. The components of the northbound and southbound parts will be explained in the context of the following subsystems that they form: SensorNode Subsystem, FlowRule Subsystem, Packet Subsystem and DeviceControlRule Subsystem. Applications and protocols will be separately explained, as they participate in several subsystems.

A. Applications

ONOS Applications layer includes two new components, which perform packet forwarding in SDN-WISE WSNs and remote sensor device management for the devices that have the corresponding driver integrated with the system. More specifically, the SensorNodeForwarding application processes incoming requests and installs the appropriate forwarding rules to the corresponding devices. When a message arrives at the application, the latter will first identify the source and the destination network devices and then, calculate the best path, by considering the global topology, consisting of both OpenFlow and SDN-WISE nodes. Then, depending on the device type (SDN-WISE or OpenFlow), the corresponding FlowRules API will be used in order to create and install the forwarding rules to the devices. At this point it should be noted that it is not necessary to use the extended FlowRules API for forwarding a packet to the next hop even in a WSN; however, as SDN-WISE has been designed specifically for WSNs, it provides functionality, like setting the whole path from source to destination in one message, that cannot be directly supported with the current ONOS features.

The SensorNodeDeviceManagement application goes beyond the standard ONOS functionality, which focuses on the network flows, and enables management operations of the networking devices themselves. We believe that this functionality is vital for a network operating system, since in a WSN, the networking devices have limited resources and their proper management may increase their lifetime and effectiveness. Currently, the SensorNodeDeviceManagement application provides simple primitives to external services in order to perform basic operations on the devices, such as turn them on and off. However, we envision the use of this component in more complex scenarios, where we will optimize resource usage by fine tuning each individual networking device from ONOS, which has a global view of the network requirements in every timeslot.

B. SensorNode Subsystem

The SensorNode Subsystem introduces new components in both the northbound and the southbound part of ONOS and it is responsible for handling information regarding the sensor nodes connected with the system. More specifically, when a message is received by a sensor node in the Protocols layer by SDN-WISE protocol, the latter first extracts the message source (note that ONOS will receive the message from the sink, however the source can be any node) and then, raises an event to notify the SensorNodeProvider that a device has possibly arrived in the system. Then, the SensorNodeManager is notified with all information accompanying the sensor node in order to update the description it currently holds. Such description includes only technical information of the node, such as serial number and operating system version. The SensorNodeManager updates the SensorNodeStore, which is used as an assisting service for persisting data, with the new information. This operation, even though seeming redundant, is very important for managing a WSN, where the network status can be unstable, since it is necessary to keep the network operating system up-to-date on the nodes’ status, so that it can make the right decisions, e.g. for packet forwarding. For this reason, the SensorNodeProvider also offers functionality to actively check the status of a sensor node, instead of waiting for its beacons. This operation is provided through the SensorNodeProvider API and it is useful in case higher layer protocols require accurate information on a node status.
(e.g. battery level or connectivity), before taking a decision and installing a rule.

All data handled in the context of the SensorNode Subsystem are accessed through the SensorNode API, which introduces new data structures designed for the representation of sensor nodes, as the current developments are limited to hosts and switches. The sensor-specific APIs provide access to information such as the battery level of a node, its neighborhood and the signal strength for each neighbor, its coordinates in 3-D space as well as specific data structures for sensor node address representation.

C. FlowRule Subsystem

The FlowRule Subsystem combines existing and new components. More specifically, the new components have been introduced to address requirements that are specific for WSNs and SDN-WISE in particular. This way, the FlowRuleProvider API has been implemented considering SDN-WISE flow tables and messages, since the existing ONOS implementation is provided specifically for OpenFlow, which has different structures. However, the use of the same API allows the transparent use of both the implementations, by leveraging an ONOS Providers layer feature that binds a service with a specific naming scheme throughout the whole platform. Therefore, flow rules meant to be sent to sensor nodes will be always sent through the new provider via the existing API. This is the functionality that enables the SensorNodeForwarding application to create and install the rules transparently, as described earlier in Section III-A.

In the NB API layer, the FlowRule API has been extended to support rules containing conditions and actions that are specific for sensor networks, whereas it has been mostly inspired by SDN-WISE. Most importantly, it adopts the SDN-WISE flexibility to create traffic matching criteria on variable areas in a network packet, instead of using fixed addresses, like the existing API. This is needed, since the IoT is characterized by the dispersiveness of architectures and therefore, it is both unrealistic and inefficient to include every possible option in the API. Rather than that, we provide an extensible component, which can easily be adapted to the specific requirements of each underlying platform. The existing FlowRuleManager can handle the extended actions, so nothing has been added in this context.

D. Packet Subsystem

The Packet Subsystem is extended only in the Providers layer, since the data structures already included in the other layers are generic enough to address the requirements of an IoT ecosystem. The PacketProvider implemented for SDN-WISE WSNs is responsible for handling all possible message types arriving at ONOS. Even though there are several message types handled in this context, in this section we are going to present the two most common ones: the Report and the Data message. Report messages carry information regarding the status of a sensor node, such as its battery level, its neighbors and the associated received signal strength indication (RSSI). This information is very critical for WSNs network operations, such as routing and therefore, ONOS should maintain a consistent view. Therefore, the SDN-WISE-based PacketProvider updates the appropriate services upon the reception of such messages. Data messages are in fact the typical forwarding requests. The PacketProvider wraps them in the format provided by the Packet API and delivers them to the packet processing applications, an instance of which is SensorNodeForwarding.

E. DeviceControlRule Subsystem

The DeviceControlRule Subsystem introduces a new modus operandi in ONOS, by extending its scope to incorporate control not only of the network flows, but also of the

---

**Figure 2. ONOS Extended Architecture for IoT**

- **SensorNode Subsystem**
  - SensorNodeForwarding
  - SensorNodeManager
  - SensorNode Provider API

- **FlowRule Subsystem**
  - FlowRule Manager
  - FlowRule Provider API

- **Packet Subsystem**
  - Packet Manager
  - Packet Provider API

- **DeviceControlRule Subsystem**
  - DeviceManagement
  - DevCtrlRule API

**APP**
**Core**
**NB API**
**SB API**
**Providers**
**Protocols**
network devices. A major design difference, compared to the standard ONOS operation, is that actions in this context are triggered proactively, without any prior request from a network device. This happens when the operating system decides that certain device operations are unnecessary or even impeding other devices operations. This is mostly expected in wireless networks, due to shared medium and power limitations and therefore, the design of this subsystem has been largely based on requirements drawn from this area.

The DeviceControlRule API in the northbound provides the device treatment instructions, with respect to the existing APIs for traffic treatment. Currently, such actions include turning the device on and off, setting the transmission and reception power, loading an executable function and updating the device about its coordinates. The rules created based on these instructions are passed to the DeviceControlRuleManager in the core layer, which is responsible for keeping statistics on the rules and then, forwarding them to the lower layers in order to be sent to the appropriate device.

In the southbound part, the DeviceControlRuleProvider receives requests from higher layers, through its SB API and creates the corresponding messages to be sent to the devices. Then, it uses the SensorNodeDriver to send the messages to the sensor nodes. Note that the flow of operations in this subsystem is only top-down, as no events are expected to come from the network.

F. Protocols

SDN-WISE protocol has a double role: on the one hand it provides access to SDN-WISE-specific data structures, so that ONOS can understand the message format required for communicating with the sensor nodes and, on the other hand, it maintains a mapping between the sensor nodes and their communication driver. Its main responsibility is to listen for incoming sensor connections, pass incoming messages to the higher layers and send messages generated in higher layers, such as flow control rules, to the sensor nodes.

SensorNodeDriver handles the low-level communication with the sensor devices. Its implementation depends on the operating system running on each connected device, such as Contiki or RIOT. In the same respect as SDN-WISE, this protocol provides access to device-specific message formats in order to properly translate the instructions specified in the context of the DeviceControlRule API.

IV. PROTOTYPE IMPLEMENTATION

The software prototype enables packet forwarding across OpenFlow and SDN-WISE networks through a single application, which considers a holistic view of the network provided by ONOS in order to decide the routing path.

The scenario presented in this section shows how the proposed architecture supports communication between a sensor node and a host, which may represent a service running in a data center, storing information generated by the sensor network. As shown in Figure 3, suppose that the sensor node belongs to an SDN-WISE network and its address is 01:00:10, whereas the host is connected to an OpenFlow switch and its address is 10.0.0.1. Both these networks will be represented inside ONOS using common, generic abstractions for devices and links, which allow for the creation of a single topology view of the whole system.

The result is shown in Figure 4, which shows ONOS representation of the topology through its built-in user interface. The gray nodes represent the SDN-WISE nodes, the blue ones the OpenFlow switches and the black ones, the hosts connected to the switches. In order to make the scenario more interesting, we suppose that there are two gateways between the networks and that the sender sensor node is in the middle of the SDN-WISE network, which means that it can use both gateways at the same cost. In the standard operation, the sender would select one of the gateways to send the packet and then, the gateway would send it to the host, regardless the location of the latter. By using the proposed ONOS extended architecture though, the path decision algorithm will be executed considering the whole topology and therefore, the selection of the gateway will be made in an optimal way.

Moreover, gateways do not need to be aware of the services that data should be transferred to, which can become very complex when several providers require access to different parts of the information generated by the WSN. By using ONOS, each application can set its own forwarding rules and the network will perform forwarding accordingly.

V. CONCLUSION

In this paper an architecture of a Network Operating System for the Internet of Things is presented. The proposed architecture starts from ONOS and extends its current scope, which mainly covers OpenFlow networks, to enhance WSNs. In order to do that, it leverages SDN-WISE, a protocol enabling SDN capabilities in wireless sensor nodes. As a result, interaction between SDN-WISE and OpenFlow networks becomes seamless, with the NOS deciding the for-
warding paths considering the whole topology and providing the appropriate commands for each device type.

ACKNOWLEDGMENT

This work has been supported by the European Commission in the framework of the FP7 Network of Excellence in Wireless COMMunications NEWCOM# (Grant no. 318306) and the INPUT (In-Network Programmability for next-generation personal cloUd service supporT) project funded by the European Commission under the Horizon 2020 Program (Call H2020-ICT-2014-1, Grant no. 644672).

REFERENCES


