# Survey and Comparison of SDN Controllers for Teleprotection and Control Power Systems

Silvio E. Quincozes, Arthur A. Zopellaro Soares, Wilker Oliveira, Eduardo B. Cordeiro,
Robson A. Lima, Débora Muchaluat-Saade, Vinicius C. Ferreira, Yona Lopes,
Juan Lucas Vieira, Luana M. Uchôa, Helio N. C. Neto, Leonardo F. Soares,
Natalia C. Fernandes, Diego Passos, and Célio Albuquerque
*Laboratório MídiaCom*
*Universidade Federal Fluminense*
Niterói/RJ, Brazil
sequincozes@gmail.com, {arthurazs,wilkerrj,bastos,robsonal,debora}@midiacom.uff.br,
{viniciusferreira,yonalopes,juanlucasvieira,luchoa,heliocunha,leonardofiorio,nataliacf}@id.uff.br,
{dpassos,celio}@ic.uff.br

***Abstract*—The advent of smart grid promotes the upgrade of substation automation technology, adding processing and communication capabilities to control and protection devices. Thereby, research on substation data communication networks is pressured to find new, efficient, and reliable ways to support such scenario. In particular, the IEC 61850 standard defines stringent temporal requirements for the power systems communication comprising teleprotection schemes. Given that the power grid is a critical infrastructure, availability is also a strong requirement. In that context, Software-Defined Network (SDN) may provide powerful tools to fulfill those requirements at acceptable costs. In this paper, we survey and compare several available SDN controllers and their applicability to the teleprotection scenario.**

***Index Terms*—Software-Defined Networks, Controllers, Teleprotection, IEC 61850, Reliability**

## I. Introduction

Power transmission lines extend through large geographical areas, thus being vulnerable to a number of events that are potentially harmful to the electrical power grid. The duration of a fault is a crucial performance aspect for the electrical grid [1]. Hence, since 2000, working group 34/35.11 of the Brazilian National Committee for Production and Transmission of Electrical Energy (CIGRÉ) [2] considers the usage of communication in line protection schemes as an efficient option. The rationale is that communication may decrease the time to perform a trigger command, which must occur from 2 to 3 cycles after the fault at any point of the line [3]. Using communication to enhance protection is known as *teleprotection* and its general structure is defined by the IEC 60834 standard [4]. Teleprotection schemes can improve selectivity and response time by allowing protection Intelligent Electronic Devices (IEDs) to exchange logical information between terminals of a transmission line. Hence, the decision time for an IED to block both external and internal faults is reduced.

This work is supported by CAPES, CNPq, FAPESP, FAPERJ, INERGE, and TAESA.

While IEC 61850 [5] seeks to improve the performance of protection schemes, it imposes stringent communication requirements to ensure the correct behavior of teleprotection functions. Not only communication delay must be minimized, but also any down time in the communication network. Otherwise, we risk jeopardizing teleprotection, which may result in financial — and even human — losses.

Therefore, a fast and reliable communication network that can quickly recover from failures is essential. Methods such as Rapid Spanning Tree Protocol (RSTP) [6], Parallel Redundancy Protocol (PRP) [7], and High-availability Seamless Redundancy (HSR) [7] are commonly employed to provide fault recovery in those networks, but they are either expensive to deploy (PRP and HSR require special topologies) or not fast enough for teleprotection (RSTP can result in long recovery times).

Software-Defined Network (SDN) [8] appears as a promising solution to fulfill the performance and resilience requirements of such networks. A centralized controller would be able to establish efficient paths for the communication of IEDs — possibly with traffic differentiation mechanisms to ensure priority for critical messages —, while minimizing flooding and, therefore, optimizing network resource usage. The controller might also proactively setup fallback paths for critical traffic, reducing recovery time when network components fail. By the same token, such a solution would allow data replication by multiple paths as a zero-recovery-time solution for failures, similarly to how HSR and PRP work, but with less specific topological requirements.

Despite the potential benefits, it is fair to wonder whether the centralized controller would not represent a bottleneck that hampers performance or robustness in such a demanding scenario. Currently, there are many SDN controllers available with varying levels of functionality, performance, reliability, and fault-tolerance. Plenty of literature compares several popular controllers [9]–[14], but research has mostly focused on performance aspects — specifically, the capacity to handle large volumes of packet-in messages and the associated re-

978-3-903176-23-2 

sponse time. Teleprotection, however, poses very particular requirements and use cases for the controller, presenting a unique set of aspects of interest. To the best of our knowledge, such a targeted evaluation of controllers has not yet been presented.

In that spirit, this paper evaluates popular SDN controllers under the light of the specific requirements of teleprotection applications. We start with an overview of teleprotection, as defined by IEC 61850, in order to properly establish the most pressing requirements. We then provide a review of several SDN controllers, ranging from popular, general purpose ones — such as Opendaylight and ONOS — to domain-specific controllers — such as SEL-5056. Finally, we use that review to study the applicability of each controller to teleprotection.

This text is organized as follows. Section II provides a primer on the IEC 61850 standard and discusses benefits and requirements of an SDN solution. Section III surveys SDN controllers, highlighting their main characteristics. Section IV evaluates those controllers, matching their characteristics to the requirements discussed in Section II. Section V, then, presents our conclusions and ideas for future work.

## II. Teleprotection with IEC 61850 and SDN

IEC 61850 [5] is a set of manufacturer-agnostic specifications developed by the International Electrotechnical Commission (IEC) to standardize communication among devices in control, protection, and automation for electric power systems. Modern relays, known as IEDs, have become micro-processed, being able to record, process, and transmit information about events, measurements, and other kinds of values. The collaboration among IEDs achieved through communication allows more efficient control and protection of the power system.

IEC 61850 is based on four pillars: information modeling; service modeling; communication protocols; and configuration languages. Information modeling defines classes and naming conventions for relevant data exchange (*e.g.*, measurements, such as voltage and current, commands, events). Service modeling defines actions to be performed on that data. Communication protocols define how data is exchanged among elements of the system. Configuration languages define a standard way to express those elements' configurations. For the remainder of this section, we will focus on communication protocols.

The standard defines two basic communication models. Two Party Application Association (TPAA) is used for service request and reply, using a reliable connection-oriented bidirectional information flow. Multicast Application Association (MCAA), on the other hand, uses a publisher-subscriber architecture for disseminating information from a source to a set of interested parties. The standard further defines three protocols that operate under those models. The Manufacturing Message Specification (MMS) implements TPAA, while Generic Object Oriented Substation Event (GOOSE) implements MCAA. Sampled Values (SV), in turn, may operate under either model.

MMS was originally standardized in ISO 9506 [15] for communication between programmable devices in Computer Integrated Manufacturing environments [16]. It was later incorporated by IEC 61850 for control and supervision of automation devices in the power system. In a nutshell, MMS allows a Supervisory Control and Data Acquisition (SCADA) system to send commands and receive replies from those devices. It can work over both TCP/IP and ISO/OSI stacks — it runs on top of TCP, in the former case —, and uses Abstract Syntax Notation One (ASN.1) to encode information in a platform-independent manner.

GOOSE is used to send event information. It groups and transmits the values of certain internal variables of the IED, which allows the receiving device to be warned — and possibly react — to events detected by others. GOOSE is, therefore, intimately related to protection schemes. Messages are asynchronous, triggered by specific events and transmitted using MAC multicast addresses, as GOOSE runs directly on top of Ethernet.

SV is used periodically sampling current and voltage data. Samples are digitized by transformers and transmitted to one or more IEDs, which, in turn, process them to check for abnormal behavior of the system.

Due to the critical nature of teleprotection, all three protocols have real-time requirements associated with their messages. MMS messages have the least stringent requirements, varying from 100 ms to over a second, depending on the application. Because SV and GOOSE are directly involved in fault detection and reaction, their time requirements are much stronger: for certain messages, no more than 3 ms is tolerated. Additionally, SV may generate large volumes of data because of the sampling frequency.

Those characteristics put severe stress on the communication network. Moreover, as the power system must have very high reliability and availability, so does the underlying communication network. Under those circumstances, SDN can bring a number of advantages. For example, GOOSE's multicast traffic can be forwarded much more efficiently, given a centralized controller with a complete view of the network topology and a previous knowledge of interested receivers. Other broadcast traffic, such as ARP, can also be handled much more efficiently, avoiding flooding. As such, resource waste is minimized, saving network capacity to fulfill the temporal requirements of critical messages and the throughput requirements of SV traffic.

SDN can also improve network reliability and reduce failure recovery time. There are two additional protocols for that purpose: HSR and PRP. Both use specially-crafted topologies to define independent alternative paths between devices that exchange critical messages — HSR uses a ring topology, while PRP uses separate Local Area Networks (LANs). The idea of both is to proactively replicate each packet in each path so that, in case of a failure in one of the paths, a copy may still reach the destination through the other. That results in zero-time recovery. Those topological requirements, however, make HSR and PRP much harder and expensive to deploy. An SDN-based solution could establish alternative *flows* in a more generic topology to achieve a similar effect.

## III. A Review of SDN Controllers

SDN's control plane is responsible for the management of flows that traverse switching devices [8]. At the heart of the control plane lies the network controller, which usually runs on a server connected to the network. Choosing an SDN controller that properly matches the network needs is an important deployment task. Here, we survey a number of SDN controllers, highlighting their main characteristics.

### A. NOX

NOX [17] provides a programming platform to control one or more OpenFlow switches. It gained popularity as one of the first able to control OpenFlow networks. Moreover, NOX is an open platform, which allows the development of management functions for both enterprise and domestic networks. NOX aims to provide the capacity of managing large networks at rates of gigabits per second without requiring special hardware to run the controller [17].

Its Northbound Interface is accessible through both C++ and Python and offers a centralized programming model in which an application may take forwarding decisions with a complete view of the topology. That seeks to simplify application development. In addition to the Application Programming Interface (API), NOX provides a Graphical User Interface (GUI) with three basic elements: a log viewer, a topology viewer and a console widget.

NOX supports Python 2.7 and C++ for application development. Several Linux distributions are officially supported. The GUI is implemented in Python using the Qt library and communicates with the controller's core through JSON messages, allowing both components to be executed in different hosts if needed.

NOX is published under the Apache 2.0 license, which allows its code to be freely modified and redistributed. While that could foster interest in the continuation of the project, NOX has witnessed a decrease in popularity, perhaps due to its performance not matching that of more recent controllers. The lack of support for multi-threading helps explain that, as NOX is unable to efficiently explore the currently common multicore processors. Moreover, it is not possible to use multiple distributed controllers. Because of all those factors, the development of NOX seems to have stalled.

### B. POX

POX is an open source controller written in python and distributed under the Apache 2.0 license. It started as an OpenFlow-specific controller, but today it provides a general framework for management of switches using both OpenFlow and Open vSwitch Database Management Protocol (OVSDB). POX is particularly popular as a teaching and researching tool [10], and it also facilitates fast prototyping of new management applications, due to its simplicity. It is an interesting alternative to NOX, if performance is not a critical requirement.

POX is developed for Python 2.7 and officially supports Windows, MacOS, and Linux. It is distributed with a few standard applications, such as simple layer-2 and layer-3 forwarding. There is no official GUI for POX, although third-party projects, such as POXDesk[1], exist. POXDesk, in particular, offers basic functionality, such as visualizing flow tables, logged events and the network topology. The communication between POXDesk and the POX's core uses a REpresentational State Transfer (REST) API available with the controller.

Like NOX, POX lacks support for distributed controllers and multi-threading. Moreover, it does not support Transport Layer Security (TLS) for secure OpenFlow communication with switches. The development of POX also seems stalled. Furthermore, literature has repeatedly shown that POX is outperformed by other more robust controllers [10], [12], [13].

### C. Beacon

Beacon is a modular, open source OpenFlow controller that supports both event-oriented execution and multi-threaded execution. Created in 2010 at Stanford University, it was widely used in academia, both for research and education. It served as the basis for the Floodlight controller.

One of the goals of Beacon was to improve productivity by allowing an administrator to start, alter, and interrupt applications in execution time [18]. It also provides a set of standard applications that implement several common functionalities of the control plane. High performance was also a goal [18].

Beacon is written in Java, using several frameworks such as Open Services Gateway Initiative (OSGi) and Spring. As such, Beacon can run in several platforms ranging from robust Linux-based servers to Android-powered smartphones. The controller is distributed under GNU General Public License (GPL) v2 and Free Open Source Software (FOSS) License Exception v1.0.

Like several other controllers, Beacon has a basic GUI that lists network nodes and shows the topology. It is also possible to inspect the flow table of each switch. Also similarly to the other controllers discussed so far, the code base of Beacon has not been updated for several years. One of the strong suits of Beacon, however, is the good quality of its code documentation.

To provide support for OpenFlow, Beacon uses the OpenFlowJ Java library, which is an object-oriented implementation of OpenFlow 1.0. Because of that, Beacon lacks support for newer versions of the OpenFlow specification. That is perhaps the main limitation of this controller.

### D. Ryu

Ryu was created by Nippon Telegraph and Telephone Corporation (NTT) and follows a component-oriented design to facilitate modification and extension of modules in response to new demands from applications. In turn, applications use components supplied by the controller in order to interface with switches and install flow-handling rules.

Some basic applications are distributed with Ryu, including an SDN-based implementation of a self-learning switch on

[1] https://noxrepo.github.io/pox-doc/html/#poxdesk-a-pox-web-gui

top of OpenFlow. A simple monitoring application, which allows an administrator to follow the current state of ports and flows, is also available. Ryu is quite flexible in terms of the southbound API. It supports several protocols, including OpenFlow versions 1.0 and 1.2 – 1.5, OFConfig, Network Configuration (NETCONF), and Nicira extensions[2].

Ryu provides a very basic web-based GUI. It exhibits topology and flow information, but it does not allow any modification of the switches' flow tables. However, to facilitate the debugging of new applications, Ryu includes a REST API that supports collecting flow statistics and topology information, as well as manipulating flow tables dynamically.

Source code is written in Python and is freely available through a GitHub repository[3] under the Apache 2.0 license. Like Beacon, Ryu excels in terms of documentation, including several application development tutorials. Unlike the previously discussed controllers, Ryu remains under active development, as evidenced by its support to recent versions of OpenFlow.

Another positive aspect of Ryu is the native support to multi-threading, although it does not support any kind distributed controller setup. While multi-threading allows Ryu to perform better under heavy loads, its performance is still considered poor in comparison to more recent controllers according to several benchmarks [19].

### *E. Floodlight*

While many SDN controllers have academic roots, Floodlight[4] is maintained by Big Switch Networks[5] and is considered a professional controller. Created in 2011, Floodlight is written in Java and supports versions 1.0 to 1.4 of OpenFlow, as well as other southbound APIs. It is distributed under the Apache 2.0 license.

Floodlight is compatible with Linux and currently requires JDK 8 for the development of applications in Java. Applications can also be written in Python using an specific library. Floodlight uses a highly modular architecture that facilitates application development.

Like Ryu, Floodlight supports multi-threading to improve performance under heavy loads. Indeed, several authors have found good results with Floodlight in terms of capacity of handling large volumes of requests [13], [20], although it is still outperformed by other controllers. Like all other controllers described thus far, however, Floodlight has no support for distributed controllers.

Basic pre-installed applications of Floodlight include an SDN-based self-learning level-2 switch and a proactive routing application that learns the network topology and installs flows for the shortest paths between network elements. A third application, called *Static Flow Entry Pusher* allows an administrator to statically install flows in each network switch. A variation of the application, called *Circuit Pusher*, allows the manual definition of complete virtual circuits.

Floodlight provides a web-based GUI that allows visualizing network topology, including hosts connected to each switch. This interface also shows detailed information about each network component, such as the configuration of Network Interface Cards (NICs) and flow tables of switches.

### *F. OpenDayLight*

OpenDayLight (ODL) was created in 2013 and is maintained by The Linux Foundation. Several companies contribute with the project, including Cisco, HP, IBM and NEC. ODL is distributed under the Eclipse Public License (EPL) v1.0, which is a more restrictive license[6] compared to the ones used by the previously discussed controllers.

This controller is written in Java and is compatible with several southbound protocols, such as OpenFlow, Simple Network Management Protocol (SNMP), NETCONF, OVSDB, Border Gateway Protocol (BGP), and Path Computation Element Communication Protocol (PCEP). It uses an architecture called Model-Driven Service Abstraction Layer (MD-SAL), and it defines a layer with common application patterns, a model for messages exchanged between applications, as well as the Yet Another Next Generation (YANG) language, which is used for data modelling. This highly modular architecture is built on top of Karaf and a configuration layer. Together, they work as the base for the applications. Among other advantages, they allow applications to be loaded and interrupted dynamically.

ODL comes with three standard applications. The first, called *Simple Forwarding*, implements a very basic forwarding mechanism based on OpenFlow. This application uses ARP traffic to identify hosts connected to the network. The second application provides a load-balancing scheme between backend servers. Balancing is achieved by means of the reactive installation of flow rules mapping the source address of packets to paths to one of the available servers. Both round robin and random policies can be used. Finally, the third application allows an administrator to monitor network statistics using a web interface.

ODL has a rich documentation, including source code documentation, installation tutorials, and general usage. While comprehensive, documentation is somewhat outdated in some aspects. For instance, as of this writing, documentation still refers to the DLUX GUI and the L2switch module, both of which were discontinued.

One major advantage of ODL with respect to the other controllers described so far is the support for distributed controllers. ODL can be configured to perform load-balancing among multiple controllers or to use a secondary controller as a backup to the primary one. It also supports multi-threading.

### *G. ONOS*

Open Network Operating System (ONOS) is an open source controller developed by Open Networking Foundation (ONF),

[2]See documentation: https://osrg.github.io/ryu/

[3]https://github.com/osrg/ryu

[4]http://www.projectfloodlight.org/

[5]https://www.bigswitch.com/

[6]Available in: https://www.eclipse.org/legal/epl-v10.html

a non-profit organization that promotes the usage of SDN and OpenFlow. ONOS is highly extensible and modular, and natively supports both distributed and multi-threaded execution. The controller core and applications are developed in Java. The main design goal of ONOS was to support the management of large-scale and fast networks, targeting mainly the ISP market segment [12]. According to Berde *et al.* [21], ONOS was created with the following requirements in mind: high throughput (up to 1 million requests per second); low latency (event processing should take between 10 and 100 ms); support to a global view of large networks (up to 1 TB of data); and high availability (up to 99,99%).

The core of the controller and the basic services are executed on the Java Virtual Machine (JVM) through the OSGi component system. That allows new modules to be installed and started dynamically in a single JVM. Moreover, the portability of Java allows ONOS to be executed in several different platforms.

The open source nature of ONOS — it is distributed under the Apache 2.0 license — has attracted a fairly large development community. As a consequence, a large number of applications are available — 187 at release of version 2.1.0. Features provided by those applications range from simple topology discovery tools and basic forwarding schemes to complex traffic handling. ONOS also supports several southbound protocols, including OpenFlow, OpenConfig, NETCONF, and P4. Additionally, driver components can be loaded to provide support to more specific devices.

ONOS has a very resourceful web-based GUI. It supports the visualization of the network topology, including discovered hosts, and exhibits details of the managed devices, links' states, flow statistics, among other information. In particular, for OpenFlow switches, the GUI allows monitoring installed flows, connected devices, and usage of each port.

This controller also provides a REST API that allows the integration of ONOS with other systems. That is useful, for example, for third-party systems that require certain network information, such as a list of connected devices or usage statistics. The REST API also allows the manipulation — installation and deletion — of switches' flow tables.

### *H. SEL-5056*

Different from the other solutions discussed in this paper, SEL-5056 SDN Flow Controller is a domain-specific controller. It is a commercial solution developed by Schweitzer Engineering Laboratories (SEL), which targets the management of SDN networks for critical infrastructure. In fact, this controller was specifically designed to work with the SEL-2740S switch manufactured by the same company.

SEL-5056 is distributed either as a Microsoft Windows application or pre-installed in a specially crafted SEL computer — SEL-3355 — that runs Windows Server 2012 R2. The software is available to download from the company's website and can be used with no cost for up to four switches. Above that limit, the software requires a one-time license fee.

Regardless of whether SEL-5056 is executed as a Windows application or on the SEL-3355 hardware, the user interfaces with the controller through a web-based GUI. This GUI allows configuring SEL-2740S switches, visualizing the network topology, as well as installing and deleting flow rules in each device. One important aspect of this controller is the fact that in earlier versions all flows had to be manually configured by an administrator. However, in the newer version of SEL-5056 a set of substation flow rules can be created and reused.

### *I. Cisco Open SDN Controller*

Cisco Open SDN Controller[7] is a commercial distribution of ODL. This version, however, is tested, validated, and officially supported by Cisco. Still, this controller obviously shares several characteristics with ODL.

Besides the official support and validation, Cisco also provides certain applications that facilitate — or complement — the usage of this controller with devices and platforms commercialized by the company. That creates an specialized SDN environment for the manufacturer. As of the writing of this paper, the following lines of products are supported: Cisco ASR 9000, Cisco Nexus 3000, and Cisco Catalyst 4500X. Nevertheless, other devices, regardless of manufacturer — including those compatible with OpenFlow — can be controlled as well.

The Cisco Open SDN Controller can be executed as a Virtual Machine, by means of an Open Virtual Appliance (OVA) that can run on both VMWare's ESXi and Oracle's VirtualBox. However, Cisco recommends users to observe a number of suggested hardware requirements for the host system.

Similarly to ODL, Cisco Open SDN Controller is compatible with a number of northbound and southbound protocols. In particular, a REST API can be used to interface with applications. For southbound communication, the controller supports OpenFlow — versions 1.0 and 1.3 —, Cisco MPSL, OVSDB, NETCONF, BGP-LS, and PCEP. The controller requires a license to be used.

Cisco Open SDN Controller is managed through a web-based GUI. The interface allows remote configuration of network elements, visualizing the network topology, accessing network statistics, and manipulating switches' flow tables, among other features.

The current version is based on ODL "Helium" (v.2), which is more than five years old. By the end of 2016, Cisco announced the discontinuation of the controller, which explains this lag. Instead, Cisco is currently investing in complete solutions — comprising hardware and software — for specific scenarios, such as cloud management (Cisco DNA center / SD-Branch) and access networks (SD-Access). Therefore, Cisco has halted the development of more general purpose adaptable SDN controllers.

[7]For more details, please refer to the datasheet available in: https://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/open-sdn-controller/datasheet-c78-733458.html

### J. Performance Comparison

Many authors have conducted and reported on performance comparisons between controllers [9]–[14]. NOX, POX, Ryu, ONOS, ODL, and Floodlight are the most often evaluated. Perhaps due to its domain-specific nature, to the best of our knowledge, SEL-5056 has not been included in any performance comparison with other controllers in the literature.

While details vary, most comparisons resort to the Cbench tool to simulate the generation of large volumes of packet-in messages, opting to measure controllers' performance under saturated conditions. Usual interest metrics are throughput and response-time (in terms of number of packet-in messages replied and the time to generate those responses). Among the controllers surveyed in this section, most comparisons agree that ONOS, ODL, Floodlight, and Beacon outperform NOX, POX, and Ryu. Within those first four controllers, however, conclusions vary depending on the evaluated metric and on the particular methodology.

One should observe, however, that not many comparisons are performed on real hardware, or even in a full network simulation environment. While Cbench is a valuable tool to stress controllers, network interactions (such as packet losses and excessive delays due to congestion) might influence results. Moreover, important aspects for the teleprotection scenario, such as long term availability and reliability, are not often evaluated. Finally, comparisons tend to focus on open source controllers, while commercial ones are generally not considered.

## IV. Qualitative Analysis

We now present a qualitative analysis of SDN controllers considering the specific requirements of teleprotection — in particular, of IEC 61850. We analyze all controllers discussed in Section III in terms of the following 13 characteristics: (i) availability and quality of documentation; (ii) availability, friendliness and completeness of a GUI; (iii) programming language in which the controller is developed; (iv) existence of standard applications with basic functionalities; (v) support to distributed controllers (load-balancing); (vi) support to backup controllers (failover); (vii) supported platforms; (viii) northbound APIs; (ix) southbound APIs and other protocols; (x) license; (xi) multi-threading support; (xii) frequency of updates; (xiii) TLS support. Table I summarizes these characteristics for each evaluated controller.

Documentation is an important characteristic of any software, but it becomes even more relevant for an SDN controller used in a teleprotection scenario, because professionals of the energy sector might have to interact with the controller. For this evaluation, we considered both official and unofficial (*e.g.*, community generated) documentation. In Table I, this characteristic is evaluated in a numerical scale from 1 to 5, which represents the sum of 5 binary properties: (i) existence of official documentation; (ii) documentation is consistent with the newest version of the controller; (iii) documentation has a beginners' guide; (iv) documentation is easy to follow; and (v) controller has an active user community (*e.g.*, discussion forums). According to our survey, only the documentations for ONOS and Ryu perfectly fit all five criteria. Beacon is particularly poor in that regard, which is perhaps explained by its overall lack of development for several years now.

While a GUI is not essential for an SDN controller, it certainly facilitates the learning and deployment of the technology, especially as controllers become more and more complete and complex. Similarly to a good documentation, a good GUI is particularly relevant in the teleprotection scenario, as it facilitates the interaction of non-specialized people. We evaluate this aspect using a numerical scale very similar to the one used for documentation. Here, the following five binary properties were considered: (i) controller has an official GUI; (ii) the GUI supports manipulating flow tables; (iii) the GUI exhibits the network topology; (iv) the GUI shows a log of events; and (v) the GUI allows modifying the behavior of the controller (*e.g.*, starting a new application). While most evaluated controllers have some kind of useful GUI, ONOS, Beacon, SEL5056, and Cisco are the only ones that perfectly fit all five criteria.

The programming language in which the controller is written can be an important factor as well. Languages such as Python and Java tend to guarantee better levels of portability. Moreover, Python is widely regarded as an easy language to learn, which might bode well if one plans to develop applications. On the other hand, because they are interpreted or based on a virtual machine, one might speculate on whether controllers written in Python or Java perform poorly in comparison to one written in C++, for instance. Java is the language used for most of the evaluated controllers, especially the more recent ones.

In terms of standard applications, most controllers include them. Usually, at least a basic level-2 forwarding application is distributed with each controller. The only exception in this evaluation was SEL-5056, which comes with no automatic flow configuration applications and requires flows to be manually configured by an administrator.

Support to distributed controllers is especially important for very large networks, as a tool to improve scalability. Among the evaluated controllers, only ODL, ONOS, and Cisco support this functionality. In a teleprotection scenario, however, the usefulness of this property is tightly coupled with fault tolerance: in a system in which faults can be catastrophic, an SDN controller should not be a single point of failure. Therefore, the usage of distributed controllers provides an automatic fault response mechanism. Nevertheless, load-balancing among controllers is not strictly necessary for fault recovery. Instead, the possibility of having a backup controller that can seamlessly take over when the primary fails is enough. In any case, support for backup controllers is only present in ODL, ONOS, and Cisco, the same controllers that support a distributed architecture.

The platforms supported by the controller are another important aspect. Controllers that are able to run on an open source Operating System (OS), such as Linux, may be desirable because they potentially decrease deployment costs. How-

TABLE I
LIST OF FEATURES USED FOR THE QUALITATIVE COMPARISON OF THE EVALUATED CONTROLLERS.

| **Controller** | **NOX** | **POX** | **Beacon** | **Ryu** | **FloodLight** | **ODL** | **ONOS** | **SEL-5056** | **Cisco** |
|---|---|---|---|---|---|---|---|---|---|
| Version | Classic | v0.5.0 | v1.0.4 | v4.30 | v1.2 | v0.9.2 | v2.1 | v2.0.0.0 | v1.2 |
| Docs | 2/5 | 3/5 | 1/5 | 5/5 | 4/5 | 3/5 | 5/5 | 2/5 | 2/5 |
| GUI | 3/5 | 3/5 | 5/5 | 3/5 | 3/5 | 4/5 | 5/5 | 5/5 | 5/5 |
| Language | C++/Python | Python | Java | Python | Java | Java | Java | C++ | Java |
| Standard Applications | Yes | Yes | Yes | Yes | Yes | Yes | Yes | No | Yes |
| Distributed Controllers | No | No | No | No | No | Yes | Yes | No | Yes |
| Backup Controllers | No | No | No | No | No | Yes | Yes | No | Yes |
| Platforms | Linux | Linux, MacOS, Windows | Linux, MacOS, Windows | Linux | Linux | Linux, MacOS | Linux, MacOS | Windows | ESXi, VirtualBox |
| Northbound APIs | REST | REST | - | REST | REST | REST | REST, gRPC | REST | REST |
| Southbound APIs | OpenFlow (1.0) | OpenFlow (1.0) | OpenFlow (1.0) | OpenFlow (1.0, 1.2-1.5), NETCONF | OpenFlow (1.0-1.4) | OpenFlow (1.0, 1.3), NETCONF | OpenFlow (1.0-1.5), NETCONF | OpenFlow (1.3) | OpenFlow (1.0, 1.3), NETCONF |
| Other Protocols | - | Nicira Extensions | - | Nicira Extensions | - | SNBI, LACP, OVSDB | TL1, SNMP, CLI, BGP, RESTCONF | - | BGP-LS, Cisco MPLS, OVSDB |
| License | Apache 2.0 | Apache 2.0 | BSD v1.0 | Apache 2.0 | Apache 2.0 | EPL v1.0 | Apache 2.0 | Proprietary | Proprietary |
| *Multi-threading* | No | No | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| Updates | None | Occasional | None | Frequent | Occasional | Frequent | Frequent | Frequent | None |
| TLS | No | No | No | Yes | Yes | Yes | Yes | Yes | Yes |

ever, devices placed inside a power plant or substation often need to fulfill a number of special requirements, such as having no moving parts [22]. Thus, specially crafted hardware, such as SEL-3355, is often employed in lieu of general purpose computers. Manufacturers of such specialized hardware often ship those devices with a pre-configured software platform. Thus, perhaps more important than support for Linux — or other open source/free OS — is the ability to run in several different platforms. In that sense, Beacon and POX are the two best options among the surveyed controllers.

In terms of northbound APIs, the majority of controllers support some kind of native internal API in one or more programming languages — usually, including the same language in which the controller is written. Nevertheless, controllers also often support other "external" APIs, which allow the execution of completely separate applications — perhaps, even in separate hardware — that simply send requests to the controller. As shown in Table I, that is the case for all surveyed controllers with the REST API, except for Beacon. ONOS, on the other hand, also supports gRPC.

Since OpenFlow is such a widely accepted industry standard, it is no surprise that all surveyed Controllers support it. However, there is clearly difference in terms of the supported versions of the specification. As expected, controllers that have had their development interrupted tend to support only earlier versions, but even some of the most modern options, such as ODL and SEL-5056 have limitations in that regard. An older version may limit functionality somehow. For example, only OpenFlow 1.3 and above allow installation of flows with QoS properties. Under this aspect, ONOS and Ryu are clearly the best options.

Besides OpenFlow, it is common for controllers to support other Southbound APIs or even interfacing with classical protocols, such as BGP and SNMP. ODL, ONOS, and Cisco are particularly comprehensive in this regard.

Licensing might also play a role when choosing a controller, particularly because of deployment costs. As such, open source controllers are clearly desirable, especially because the number of controlled devices might be substantial in a teleprotection environment. Indeed, most surveyed controllers fit that description, except Cisco and SEL-5056. Aside from the financial motivation, an open source controller also allows the inspection of the source code, which might be important for security reasons, especially in such a critical infrastructure. Furthermore, licenses that permit code modification may be of interest because they allow the controller to be modified to better fit a particular scenario, if needed.

While we do not anticipate a very heavy load on a controller used in a teleprotection scenario, multi-threading support is certainly beneficial and can perhaps have an impact — even if small — on response time in certain cases. Most modern controllers present such support.

The frequency of updates is of paramount importance in the teleprotection scenario. While an older controller might perhaps adequately fulfill functional requirements of a certain deployment, regular updates are important to ensure protection

against zero-day attacks, as well as possible bugs that may cause network instability. Given the critical nature of teleprotection, those two aspects cannot be overlooked. Among the controllers still under development, Ryu, ODL, SEL 5056, and ONOS seem to be the most active projects.

By the same token, ensuring secure communication between controller and switches is fundamental. Despite the numerous potential benefits of the SDN paradigm, it does open potential attack vectors to the network, as devices become more remotely configurable. OpenFlow, in particular, provides a native security solution that is based on the usage of TLS for the secure communication of devices and controller. However, even some popular controllers fail to provide support for TLS. That is the case, for example, of NOX, POX, and Beacon. In general, however, our survey shows that modern controllers tend to support it.

## V. Conclusions

For the past several years, the energy sector has been gradually shifting towards introducing processing and communication capabilities to protection and control devices that comprise the power grid. That paradigm shift, popularly known as Smart Grid, has the potential to increase the grid's efficiency in several aspects. In particular, the concept of teleprotection, standardized in IEC 60834 (and then in IEC 61850), allows a more effective power systems protection, which ultimately results in better robustness, availability, and an overall lower probability of damages to such a critical infrastructure.

In this paper we provided a short introduction to the communication aspects of IEC 61850, including an overview of the involved protocols, traffic patterns, and temporal requirements for the most important messages exchanged in a teleprotection scenario. Based on that, we conducted a brief survey of popular SDN controllers that may be used in an SDN-based solution for the communication network to support teleprotection. Furthermore, we performed a qualitative comparison of those controllers, with particular focus on aspects that are relevant for that application. That comparison suggests that ONOS may be the best choice for such a scenario, due to its good balance between supported features, documentation, and development pace. Given those criteria, ONOS seems to overcome even commercial controllers targeted at that particular domain, such as SEL-5056. Overall, it fits most requirements for teleprotection, being robust, open source, and reasonably friendly for non-specialized personnel.

Because of the well-defined strong temporal requirements of IEC 61850 traffic, however, we believe this work should be complemented with a quantitative comparison. Nevertheless, different from what other generic comparisons have done, this analysis should focus on a more realistic testbed and on metrics that are of particular interest to the teleprotection scenario.

## References


[1] International Electrotechnical Commission, "IEC 61850-90-1: Use of IEC 61850 for the communication between substations," IEC, Tech. Rep., 2010.

[2] A. Adamson, "Protection Using Telecommunications," CIGRÉ Joint Working Group 34/35.11, Tech. Rep., 2000.

[3] International Electrotechnical Commission, "IEC 61850-5: Communication requirements for functions and device models," IEC, Tech. Rep., 2003.

[4] ——, "IEC 60834: Teleprotection Equipment of Power Systems - Performance and Testing," IEC, Tech. Rep., 1999.

[5] ——, "IEC 61850: communication networks and systems in substations," IEC, Tech. Rep., 2013.

[6] ——, "IEC 62439-1: General Concepts and Calculation Methods (Including RSTP)," in *Industrial Communication Networks - High Availability Automation Networks*. IEC, 2010.

[7] ——, "IEC 62439-3: Parallel Redundancy Protocol (PRP) and High-availability Seamless Redundancy (HSR)," in *Industrial Communication Networks - High Availability Automation Networks*. IEC, 2010, pp. 1–62.

[8] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[9] S. A. Shah, J. Faiz, M. Farooq, A. Shafi, and S. A. Mehdi, "An architectural evaluation of SDN controllers," in *2013 IEEE International Conference on Communications (ICC)*. IEEE, Jun. 2013, pp. 3504–3508.

[10] A. L. Stancu, S. Halunga, A. Vulpe, G. Suciu, O. Fratu, and E. C. Popovici, "A comparison between several Software Defined Networking controllers," in *2015 12th International Conference on Telecommunication in Modern Satellite, Cable and Broadcasting Services (TELSIKS)*, Oct 2015, pp. 223–226.

[11] Y. Zhao, L. Iannone, and M. Riguidel, "On the performance of SDN controllers: A reality check," in *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, nov 2015, pp. 79–85.

[12] O. Salman, I. H. Elhajj, A. Kayssi, and A. Chehab, "SDN controllers: A comparative study," in *2016 18th Mediterranean Electrotechnical Conference (MELECON)*, Apr. 2016, pp. 1–6.

[13] S. Rowshanrad, V. Abdi, and M. Keshtgari, "Performance evaluation of SDN controllers: Floodlight and OpenDaylight," *IIUM Engineering Journal*, vol. 17, no. 2, pp. 47–57, 2016.

[14] R. Jawaharan, P. M. Mohan, T. Das, and M. Gurusamy, "Empirical evaluation of sdn controllers using mininet/wireshark and comparison with cbench," in *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, July 2018, pp. 1–2.

[15] "ISO 9506-1:2003:Industrial Automation Systems – Manufacturing Message Specification (MMS) – Part 1: Service Definition," International Organization for Standardization, Standard 2, 2003.

[16] "ISO 9506-2:2003:Industrial Automation Systems – Manufacturing Message Specification (MMS) – Part 2: Protocol Specification," International Organization for Standardization, Standard 2, 2003.

[17] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: Towards an operating system for networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, Jul. 2008.

[18] D. Erickson, "The Beacon Openflow Controller," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13, 2013, pp. 13–18.

[19] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced study of SDN/OpenFlow controllers," in *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia on - CEE-SECR '13*, 2013, pp. 1–6.

[20] Z. K. Khattak, M. Awais, and A. Iqbal, "Performance evaluation of OpenDaylight SDN controller," in *Parallel and Distributed Systems (ICPADS), 2014 20th IEEE International Conference on*. IEEE, 2014, pp. 671–676.

[21] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, and W. Snow, "ONOS: towards an open, distributed SDN OS," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 1–6.

[22] "IEEE 1613: Environmental and Testing Requirements for Communications Networking Devices in Electric Power Substations," Institute of Electrical and Electronics Engineers, Standard, 2003.