Towards visual programming abstractions in Software-Defined Networking

Rojas, Elisa; Ollora Zaballa, Eder; Noci, Victoria

Published in:
Internet Technology Letters

Link to article, DOI:
10.1002/itl2.358

Publication date:
2022

Document Version
Peer reviewed version

Towards Visual Programming Abstractions in Software-Defined Networking

Elisa Rojas*[1] | Eder Ollora Zaballa*[2] | Victoria Noci[1]

[1] Universidad de Alcalá, Dpto. de Automatica, Alcalá de Henares, Spain
[2] DTU Fotonik, Technical University of Denmark, Kongens Lyngby, Denmark

Correspondence
*Elisa Rojas - Email: elisa.rojas@uah.es,
*Eder Ollora Zaballa - Email: eoza@fotonik.dtu.dk
Elisa Rojas and Eder Ollora Zaballa should be considered joint first author.

Present Address
Corresponding addresses are *Dpto. de Automatica, Universidad de Alcalá, Spain* and *DTU Fotonik, Technical University of Denmark*, respectively.

Summary

Since Software-Defined Networking (SDN) emerged, the research community and industry have developed numerous projects and fostered novel use cases. However, engineers now need to learn how to program the control and data planes, which might slow down technology acceptance. To accelerate it, visual programming abstractions facilitate the incorporation of SDN technologies and assist in creating new applications. So far, very little effort has been made in this field. This letter presents an early-stage SDN visual abstraction initiative based on the Scratch/Blockly programming framework, initially aimed at kids. The objective is to illustrate how this work could be extended to provide value-added resources for network programming.

KEYWORDS:
Software-Defined Networking, Scratch, Blockly, Visual Programming, Network Programmability, Human-Defined Networking, Softwarized Networks

1 | INTRODUCTION

In 2008, the report publication of the OpenFlow protocol started a revolution in network programming, later on, entitled as Software-Defined Networking (SDN)[1]. Although the topic was far from being new, control plane logical centralization and softwarization, along with the use of OpenFlow (to communicate the control and the data planes), encouraged researchers to develop new use cases[2]. The limitations of OpenFlow fostered data plane programming, emerging as the latest evolution in network management together with P4[3] as its reference language. As data plane programming evolved, another Control-Data Plane Interface (CDPI), popularly known as Southbound Interface (SBI), protocol was needed. P4Runtime became, at this point, the *de facto* protocol for data planes programmed in P4.

While centralized control planes became a popular topic among researchers, creating a Proof-of-Concept (PoC) control plane for specific tests was possible but time-consuming, and particularly challenging for industry[4]. Creating reliable, resilient, and optimized control planes for data centers or telco-grade applications was only feasible for experienced programmers and developers with a networking-specific background. The complexity of creating control plane applications was –and is– still evident. Due to the steep learning curve, we identify a gap in SDN programming, that is, the missing possibility to fully or partially provide a reliable visual programming abstraction for the SDN control and/or data planes. At this point, we believe that a visual framework for network programming could serve as a software *wizard* or assistant, capable of setting up the very few initial parts of any application, thus, softening that curve. At the same time, industry innovation processes would benefit from it, as engineers could leverage it to learn about the field and quickly acquire some basic knowledge about it.

The letter is structured as follows: Section 2 reviews the current state of the art regarding visual programming. Section 3 depicts the abstractions for control and data planes. Section 4 presents ScratchingSDN, an application of visual programming in SDN. Next, Section 5 discusses how this work can be extended and improved, and it concludes in Section 6.

2 | RELATED WORK

Two main topics are related to the idea conveyed in this letter: visual programming and SDN. In the case of the former, Scratch[5] and Blockly[6] are well-known sets of tools and libraries that facilitate programming by using a visual code editor, although others like Snap! and EduBlocks also follow the same approach. The conceptually connected blocks represent logical expressions, including variables or loops, and can be converted to various languages. In this way, the learning curve is flattened by allowing code bootstrapping that can be rapidly generated using visual entities. The use cases involving Blockly are numerous.

Currently, published work focuses on a wide variety of Blockly-based topics and use cases. Ashrov *et al.*[7] present a behavioral programming study based on JavaScript and a Blocky-based architecture. The authors try to apply the aforementioned tools to client-side web and smartphone applications. Nguyen *et al.*[8] introduce BlocklyAR, a visual programming interface to create Augmented Reality (AR) applications. The authors managed to create an interface that can replicate existing applications but implied less development effort. Weintrop *et al.*[9,10] leverage a block-based programming interface for industrial robotics. The goal of the study is to make robotics programming more accessible. A low-scale test shows that programmers with no previous experience can properly program a virtual robot.

On the other hand, so far, SDN has not been of great focus when developing high-level programming abstractions based on visual programming; probably because visual and block-based programming are generally applied to learn either basic programming or provide an accessible programming interface to non-expert professionals. Still, when developing high-level programming abstractions based on visual programming, a few researchers have contributed to the field by integrating these abstractions for SDN. For example, Schultz *et al.*[11] present a Graphical User Interface (GUI) named OpenGUFi. This platform provides a visual abstraction of the underlying network and enables network actions via the control interface. Using gestures, network managers can trigger network handovers and visualize traffic flows involving wireless access points and mobile clients. Other approaches consider that providing a human-like interface is critical for the evolution of SDN[4]. In this regard, StreaMon[12] is an XML-based abstraction for monitoring in SDN, while OpenFunction[13] is an abstraction for SDN middleboxes. Two recently published pre-prints present SeaNet[14] and Lucid[15], focused on high-level abstractions for control and data planes, respectively. Additionally, focusing on the data plane, Graph-to-P4[16] allows the user to define a parse graph using blocks that represent parse states. Each parse state extracts the header it carries as a name, and the compiler will translate the graphical representation to the P4 pipeline code (including the parser). Similarly, P4click[17] facilitates data plane programming based on data plane modules. Also, μP4[18] abstracts from the underlying hardware and their lower-level architecture, but it still requires considerable understanding of data plane programming. Finally, Michel *et al.*[19] survey programmable data plane abstractions and questions how to measure the tradeoff between supported functionality, resulting in performance, and API simplicity.

Therefore, though many of these works focus on high-level abstractions for SDN, none achieves a truly visual abstraction. Accordingly, the main contribution of our letter is the proof that visual programming and SDN can be merged to provide a drag-and-drop interface for network managers.

3 | ABSTRACTIONS FOR CONTROL AND DATA PLANES

The SDN architecture typically comprises three layers or planes: data or infrastructure, control, and application (see Figure 1). Elements from each plane communicate with the adjacent layer elements using different protocols. The boundary between each adjacent layer is an interface. The Northbound Interface (NBI) defines the boundary between the control and application planes. Some example protocols of the NBI include HTTP/S (REST) mainly for external applications, but also for internal ones that can use programming APIs (Python, Java, Go, etc.). Similarly, the SBI is the common boundary of the control and the data plane. The elements that belong to the control and data plane communicate using protocols such as OpenFlow, P4Runtime or NETCONF (among others). To provide visual programming abstractions, it is necessary to define the control and data plane requirements and how these map to the elements that belong to each plane.

3.1 | Defining the control plane structure

The control plane structure depends on the actual SDN controller implementation. For example, the Ryu SDN controller is characterized by offering a simple approach to developing SDN applications. The architecture of Ryu provides an event queue and event loop that further forwards the events to the appropriate internal handlers. New control plane applications in Ryu

inherit from the base application that allows developers to create methods with decorators that handle new events, such as *PACKET_IN* events. Similarly, other popular controllers like the Open Network Operating System (ONOS) offer a similar approach to handle new events but try to provide a higher abstraction to disengage the SDN architectural planes clearly. The ONOS SDN controller uses an Open Services Gateway initiative (OSGi) framework to include modular applications. Creating new applications in ONOS offers the possibility to generate independent *oar* packages that can be installed at runtime. This separates the controller's core layer and core applications and the possible applications that might be aggregated at runtime. These applications are also called *bundles* and can be installed, uninstalled, started, and stopped without modifying the state of the runtime OSGi. OpenDaylight offers a similar architecture to ONOS but is based on models instead of specific programming APIs. Due to their modularity, architecture, and performance, both ONOS and OpenDaylight are considered production-grade controllers but require a steeper learning curve, making it complicated to develop new applications for newcomers efficiently.

Figure 2(a) shows how visual programming abstractions relate to the control and data planes. In this figure, the control plane (as previously defined in Figure 1) is divided into three layers, first, the controller-dependent code applications (either based on Ryu or ONOS), then the core (which includes basic services, such as topology discovery), and finally the SBI protocols and drivers. Although we consider Ryu and ONOS as relevant examples of SDN controllers, many others exist, like NOX/POX or Floodlight[20], and they all share similar architectural definitions.

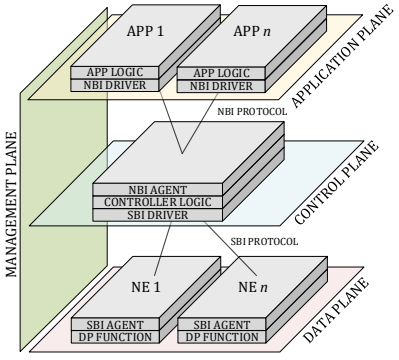

FIGURE 1 A visual representation of the SDN architecture. *NE* stands for Network Element, *APP* for Application, *NBI* for Northbound Interface and *SBI* for Southbound Interface.

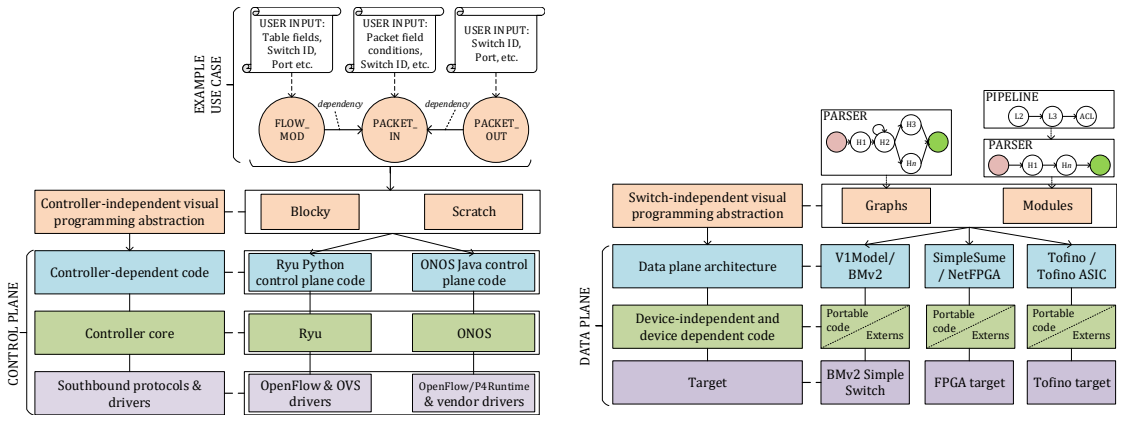

FIGURE 2 Mapping of (a) visual abstractions to control plane components, and (b) visual abstractions to data plane components.

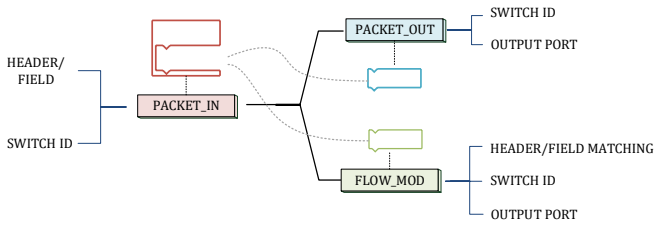

FIGURE 3 Block-based example case. A *PACKET_IN* handler that outputs *PACKET_OUT* and *FLOW_MOD* messages based on user input parameters.

3.2 | Defining the data plane structure

As seen in Figure 1, the data plane comprises the bottom-most part of the architecture; e.g., OpenFlow/P4Runtime switches, P4-programmable FPGAs or P4 SmartNICs. While OpenFlow data plane hardware has become popular, no custom pipeline can be programmed on these switches. The recent advances in programmable Application-Specific Integrated Circuits (ASICs) and Field-Programmable Gate Arrays (FPGAs) offer the possibility to define pipeline processing code. Nowadays, software and hardware switches can be programmed using P4, which opens new gaps for network developers. While P4 offers a standard specification, different architectures exist, complicating creating a program that works on every target and architecture.

Figure 2(b) illustrates how a graph or module-based framework could be used to generate a visual representation of the data plane application or to generate the modular pipeline (ultimately translated into the P4 code of the selected target).

4 | SCRATCHINGSDN

ScratchingSDN (available in GitHub[21]) is a Blockly-based Ryu visual programming framework for the control plane. Its main objective was to develop a framework that could be used to program a network simply via drag-and-drop actions. Its architecture is founded on a set of new blocks for the Blockly editor, following the same idea of Scratch for kids learning how to program. These blocks, when combined, generate handlers for the Ryu controller. Ryu was selected due to its simplicity and, as it mainly uses OpenFlow as its SBI protocol. In particular, we modeled four OpenFlow messages: *PACKET_IN* and *PORT_STATUS* (switch-to-controller messages), and *PACKET_OUT* and *FLOW_MOD* (controller-to-switch messages). Although OpenFlow outlines many messages, the reason for those was that they are, statistically, the most frequently leveraged by SDN engineers to define runtime network functionality. Once the messages were designed, we edited the HTML and JavaScript code of Blockly, customizing the Python editor –as Ryu is Python based– and, afterwards, providing an empty Ryu app template, which would be later on populated with the translated content of the blocks.

The typical SDN workflow encompasses the reception of a switch-to-controller message and the action to be applied in response to it (controller-to-switch). Figure 3 illustrates the visual structure that handles a *PACKET_IN* message. All these blocks offer a text input for developers in order to define which parameters to filter in the handler, e.g., the destination MAC for a *PACKET_IN* message. Once a set of blocks is defined, ScratchingSDN translates them online in their web-based editor and allows the download of a piece of Ryu code (Python-based), which is directly executable in the Ryu framework. Hence, the code is already following a visual approach, in which the network administrator should not write a single line of Python code.

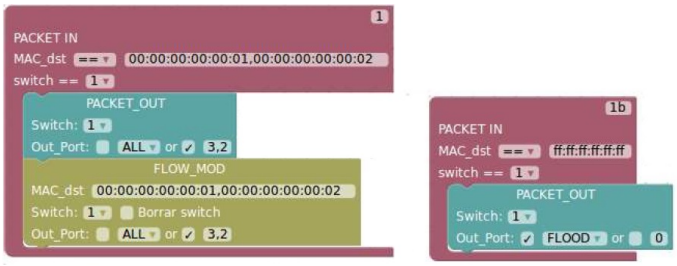

FIGURE 4 Screenshot of ScratchingSDN containing part of the code developed via drag and drop.

To evaluate ScratchingSDN, we created a simple topology with two hosts connected via two different paths. We managed to install a default route from one to the other, which was diverted to the second one when the main one failed. The whole scenario was developed strictly via drag-and-drop actions, without further intervention or code writing. As an example, Figure 4 depicts a piece of the developed code. Once the PoC was developed and evaluated, ScratchingSDN was successfully extended to support ONOS. ONOS is focused on industry-based deployments and possesses higher network abstractions, but still, ScratchingSDN seems to facilitate the creation of SDN projects. The amount of generated code per block project is more extensive as well, as Ryu requires a single Python file, while ONOS needs a Java bundle, which will be compiled afterward.

5 | DISCUSSION AND FUTURE WORK

ScratchingSDN facilitates initial developments in SDN frameworks, simply by drag-and-drop actions and without specific knowledge about the supporting programming language. Its features are still clearly limited, and it could be extended to support additional headers and fields to decide when to execute a handler, such as the *PACKET_IN* handler. The same can be used to support *PACKET_OUT* or *FLOW_MOD* messages. Moreover, this block-based framework can be leveraged for multiple controllers. Future research could also focus on including handlers for all the headers supported in the latest CDPI protocols. On the other hand, the main drawback that we found was the dependence of the controllers' NBI with its SBI (particularly with Ryu). In this regard, it is critical to disengage both entities for portable –and truly visual– programming.

For these reasons, although we intended to generate a genuinely visual layer above SDN controllers, we believe this is not scalable in the long term, as controllers are continuously growing and being updated. However, ScratchingSDN provides an alternative advantage: the ability to deploy an initial piece of code from scratch that can be later modified and extended, i.e., to avoid starting from a blank file, which is particularly challenging for newcomers in the area. It is important to note that many developers in the field copy and paste code from other examples. Therefore, ScratchingSDN could grant an easy and clean starting point, even if developing the complete functionality appears to be challenging at this stage.

6 | CONCLUSION

This letter has introduced the existing projects and works that use visual programming abstractions. The reduced number of research works explains why little work leverages visual programming in SDN. As a PoC, we implemented ScratchingSDN, which, to the best of our knowledge, is the first proposal that merges the Blockly framework with an SDN controller (Ryu and ONOS) for network programming based on visual abstractions. We believe this PoC accomplishes the objective of serving as a starting point for network programmers. However, as explained in Section 5, there is likely more research to be done in this field: the frameworks exist but lack extended features and a more comprehensive range of functionalities. For the control plane visual abstractions, frameworks need to support other controllers, other OpenFlow (even P4Runtime) messages, or specific API calls. For the data plane, new targets have to be supported and simple visual abstractions to define data plane logic functions.

ACKNOWLEDGMENTS

This work was funded in part by grants through projects TAPIR-CM (S2018/TCS-4496) and IRIS-CM (CM/JIN/2019-039) of Comunidad de Madrid, and ONENESS (PID2020-116361RA-I00) of the Spanish Ministry of Science and Innovation.

References

1. Kreutz D, Ramos FMV, Veríssimo PE, Rothenberg CE, Azodolmolky S, Uhlig S. Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE* 2015; 103(1): 14-76. doi: 10.1109/JPROC.2014.2371999

2. Garrich M, Moreno-Muro FJ, Bueno Delgado MV, Pavón Mariño P. Open-Source Network Optimization Software in the Open SDN/NFV Transport Ecosystem. *Journal of Lightwave Technology* 2019; 37(1): 75-88. doi: 10.1109/JLT.2018.2869242

3. Bosshart P, Daly D, Gibb G, et al. P4: Programming Protocol-Independent Packet Processors. *SIGCOMM Comput. Commun. Rev.* 2014; 44(3): 87–95. doi: 10.1145/2656877.2656890

4. Rojas E. From Software-Defined to Human-Defined Networking: Challenges and Opportunities. *IEEE Network* 2018; 32(1): 179-185. doi: 10.1109/MNET.2017.1700070

5. Scratch - Imagine, Program, Share. https://scratch.mit.edu/; . (Accessed on 09/11/2021).

6. Blockly | Google Developers. https://developers.google.com/blockly; . (Accessed on 09/11/2021).

7. Ashrov A, Marron A, Weiss G, Wiener G. A use-case for behavioral programming: An architecture in JavaScript and Blockly for interactive applications with cross-cutting scenarios. *Science of Computer Programming* 2015; 98: 268-292. Special Issue on Programming Based on Actors, Agents and Decentralized Controldoi: https://doi.org/10.1016/j.scico.2014.01.017

8. Nguyen VT, Jung K, Dang T. BlocklyAR: A Visual Programming Interface for Creating Augmented Reality Experiences. *Electronics* 2020; 9(8). doi: 10.3390/electronics9081205

9. Weintrop D, Shepherd DC, Francis P, Franklin D. Blockly goes to work: Block-based programming for industrial robots. In: B&B'17. ; 2017: 29-36.

10. Weintrop D, Afzal A, Salac J, et al. Evaluating CoBlox: A Comparative Study of Robotics Programming Environments for Adult Novices. In: CHI '18. Association for Computing Machinery; 2018; New York, NY, USA

11. Schultz J, Szczepanski R, Haensge K, Maruschke M, Bayer N, Einsiedler H. OpenGUFI: An Extensible Graphical User Flow Interface for an SDN-Enabled Wireless Testbed. In: CIT '15. ; 2015: 770-776.

12. Bonola M, Bianchi G, Picierro G, Pontarelli S, Monaci M. StreaMon: A Data-Plane Programming Abstraction for Software-Defined Stream Monitoring. *IEEE Transactions on Dependable and Secure Computing* 2017; 14(6): 664-678. doi: 10.1109/TDSC.2015.2499747

13. Tian C, Munir A, Liu AX, Yang J, Zhao Y. OpenFunction: An Extensible Data Plane Abstraction Protocol for Platform-Independent Software-Defined Middleboxes. *IEEE/ACM Transactions on Networking* 2018; 26(3): 1488-1501. doi: 10.1109/TNET.2018.2829882

14. Zhou Q, Gray AJ, McLaughlin S. Towards A Knowledge Graph Based Autonomic Management of Software Defined Networks. *arXiv preprint arXiv:2106.13367* 2021.

15. Sonchack J, Loehr D, Rexford J, Walker D. Lucid: A Language for Control in the Data Plane. In: SIGCOMM '21. Association for Computing Machinery; 2021; New York, NY, USA: 731–747

16. Ollora Zaballa E, Zhou Z. Graph-To-P4: A P4 boilerplate code generator for parse graphs. In: ANCS'19. ; 2019: 1-2.

17. Ollora Zaballa E, Franco D, Berger MS, Higuero M. *A Perspective on P4-Based Data and Control Plane Modularity for Network Automation*: 59–61; New York, NY, USA: Association for Computing Machinery . 2020.

18. Soni H, Rifai M, Kumar P, Doenges R, Foster N. Composing Dataplane Programs with uP4. In: SIGCOMM '20. Association for Computing Machinery; 2020; New York, NY, USA: 329–343

19. Michel O, Bifulco R, Rétvári G, Schmid S. The Programmable Data Plane: Abstractions, Architectures, Algorithms, and Applications. *ACM Comput. Surv.* 2021; 54(4). doi: 10.1145/3447868

20. Ahmad S, Mir AH. Scalability, Consistency, Reliability and Security in SDN controllers: A Survey of Diverse SDN Controllers. *Journal of Network and Systems Management* 2021; 29(1): 1–59.

21. Noci V. ScratchingSDN. https://github.com/NETSERV-UAH/TFGs/tree/master/201906-VictoriaNoci; . (Accessed on 09/11/2021).

How to cite this article: Rojas E., Ollora Zaballa E. and Noci V. (2021) , Towards Visual Programming Abstractions in Software-Defined Networking, *Internet Technology Letter*, .