Capability-Aware SDN Application Models: Dealing with Network Heterogeneity

Felipe A. Lopes[†‡], Robert Bauer[‡], and Stenio Fernandes[†]
[†]Centro de Informática, Universidade Federal de Pernambuco (Recife, Brazil)
[‡]Karlsruhe Institute of Technology (Karlsruhe, Germany)

***Abstract*—Software-Defined Networking (SDN) is a well known key enabler for flexible network operation. However, different types of SDN controllers and SDN switches with a wide range of capabilities (e.g., different protocol versions and support for various hardware features) could limit this flexibility. The not easy task of developing SDN applications becomes even more challenging when we consider such diversity. To soften this issue, we extend our ongoing work in the area of Model-Driven Networking (MDN) to explicitly support infrastructure heterogeneity. More precisely, our extended MDN framework provides the possibility to generate SDN applications from MDN models considering the capabilities of the underlying network elements. In this paper, as an use case, we present MDN to develop a capability-aware network monitoring application.**

I. Introduction

Software-Defined Networking (SDN) is an emerging trend for programmable and flexible network control. However, several challenges still exist regarding the development of new network applications for the SDN paradigm. Even though current SDN programming languages enable application development based on a high abstraction level, the developer still needs to concern with issues like compatibility between SDN programming languages and controllers, rule consistency, and packet header formats.

Moreover, real SDN deployments often consist of a wide range of heterogeneous entities (e.g., constantly evolving protocols, SDN controllers, and switches with different capabilities). Therefore, the lack of tools for easy SDN application development and validation together with the problem of infrastructure heterogeneity composes the perfect storm to hinder flexibility. For instance, if an SDN application considers a specific monitoring action (e.g., NetFlow) and this action needs to be executed on a network element that does not support it, the application may not run properly.

The Model-Driven Networking (MDN) framework created by Lopes et al. [1] can be seen as a feasible step into the direction of a vendor-agnostic tool for high-level SDN application development. This paper presents the contribution of our research and describes the extension to our previous work on MDN so that the modeled applications can deal with network heterogeneity. To achieve this, we include modeling support for different types of packet headers, instruction sets, and switch capabilities. In addition, we incorporate the idea of on-demand capacity and feature delegation presented in [2] into the process of generating capability-aware SDN applications.

II. Model-Driven Networking (MDN)

MDN is a framework and our main contribution to increase the abstraction level in developing SDN applications [1]. It is based on the Model-Driven Engineering (MDE) paradigm, providing a Domain-Specific Modeling Language (DSML) and a Computer-Aided Software Engineering (CASE) tool for enabling the modeling of SDN applications and their underlying environments.

Our overall idea behind MDN arose after analyzing several programming languages proposed to develop SDN applications in the last years. We have noticed that although such languages had brought some benefits for increasing the abstraction level in developing SDN applications, there are still open issues like the lack of tools for developing and validating applications, low level details of SDN protocols (e.g., OpenFlow versions), and error-prone actions regarding flow rules.

Aiming to offer a solution for the problems above, we have designed and built the architecture depicted in Figure 1. The top layer - Application Models - involves the models created in our CASE tool. Such models are based on relationships and constraints that form the layer below named Metamodel & Semantics. After verifying the consistency of an application model, the Code Generation layer is responsible for transforming high-level models into source code used to interact with a target controller. The MDN architecture expects that the selected SDN controller uses the Southbound API to execute the generated code (i.e., SDN application) as instructions and rules for programming the forwarding plane.

Increasing the abstraction level for developing or managing SDN applications is the objective of several research efforts. In [3] we have classified the current approaches into the following categories: programming languages (based on the Domain-Specific Language paradigm) and modeling approaches. The former are distinguished by the structural paradigm applied (e.g., declarative, functional reactive programming). The related modeling approaches are also proposals focusing on modeling and graphical editors [4] [5]. However, MDN provides more than a trivial modeling and CASE tool. As MDN is based on a DSML its features make it feasible to address several current issues in developing SDN applications, such as: low level details of SDN protocols, validation, and dependency between programming languages and controllers.

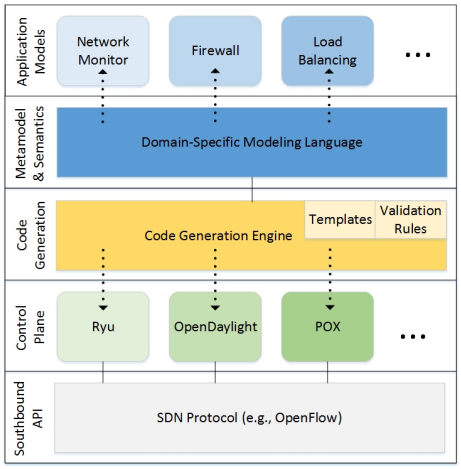

Fig. 1: Elements of the MDN architecture.

III. Generating Capability-Aware SDN Applications

SDN switch heterogeneity is a well known issue [6]. To illustrate the problem, consider the following example: a provider needs to develop a monitoring application for a network composed of two interconnected switches *S1* and *S2*. Those switches have different capabilities, e.g., *S2* is capable of handling NetFlow (for instance) and *S1* is not. However, the provider wants to use NetFlow for the whole network (including *S1*) without having to change the hardware or manually adapt the application. The extension to MDN prior mentioned deals with issues like that above. By adding support for the description of switch capabilities in the MDN metamodel, those capabilities can be considered in the code generation process (cf. Section II). As a result, the generated applications can be extended with mechanisms to deal with capability shortcomings. The new application development workflow including the extension looks as follows:

1) Modeling step, including infrastructure capabilities.
2) Evaluate if the application will work without limitations on the current model.
3) If not, try to define mechanisms for capability-awareness to avoid such limitations.

We currently rely on replication and flow delegation [2] as concrete mechanisms for capability-awareness (step three of the workflow). If, for example, a network device can not provide a certain capability, OpenFlow (OF) rules are used to automatically *delegate* the affected flows to a suitable neighboring device where they can be processed. The required rules and the control logic for flow redirection are included in the Code Generation layer of the MDN framework.

Figure 2 depicts the necessary steps of generating a capability-aware application for the monitoring use case introduced above. It considers a condition in which *S1* does not support NetFlow. The first step in Figure 2 refers to the creation of an application model (i.e., network monitor) by a network developer or operator. Step 2 shows a code snippet (written in Object Constraint Language) of a controller template that is present in the Code Generation layer. Such snippet exhibits how to guarantee flexible code by tagging *S1* flows with a *NetFlow* tag and proactively replicating them to *S2* (we omitted the creation of rules in *S2* for further processing). Step 3 is a snippet of the final application code.

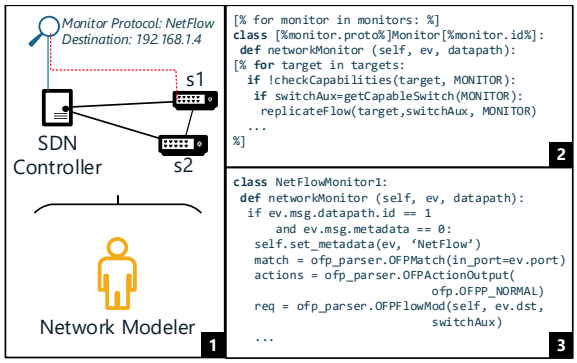

Fig. 2: Generating a capability-aware monitoring application

Our initial performance analysis (compared to the experiments published in [1]) shows a mean increasing of 6% for validation time of the network scenario (including the modeled application). Besides, there is an increasing of 11% in generating applications when considering the verification of all possible packet headers (e.g., OF specifications 1.0-1.5) and monitoring capabilities (e.g., NetFlow) present in network switches. The impact on network performance (e.g., flow table usage, hop count) is an open issue at this moment.

IV. Conclusions and Future Work

Consider applications that need resources or capabilities not available at every network node. Instead of not supporting a service or discarding the utilization of a network node, capability-aware SDN applications look for alternative ways to satisfy specific requirements by considering capabilities of the whole network. This paper proposes an extension to the MDN framework so that capability-aware SDN applications can be automatically created using a high-level, model-driven approach. We therefore integrate information regarding the underlying infrastructure into the MDN metamodel and use flow delegation at the Code Generation layer to realize capability-awareness.

The proactive behavior and the identification of switch capabilities enabled by our extension can improve performance and compatibility of generated applications (e.g., in scenarios where different switches support different OF versions). Application placement recommendations, the impact of real-time network state on application models, and a complete performance evaluation are issues defined for future work.

Acknowledgment

The authors would like to thank the FACEPE and ERASMUS (grants IBPG-1200-1.03/14 and BM15DM0988 for Felipe A. Lopes). This work has been performed in the framework of the CELTIC EUREKA project SENDATE-PLANETS (Project ID C2015/3-1), and it is partly funded by the German BMBF (Project ID 16KIS0460K).

References

[1] F. A. Lopes, L. Lima, M. Santos, R. Fidalgo, and S. Fernandes, "High-level modeling and application validation for SDN," *Proceedings of the NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pp. 197–205, 2016.

[2] R. Bauer and M. Zitterbart, "Port based capacity extensions (pbces): Improving sdns flow table scalability," in *28th International Teletraffic Congress (ITC 28)*, (Wuerzburg, Germany), 2016.

[3] F. A. Lopes, M. Santos, R. Fidalgo, and S. Fernandes, "A Software Engineering Perspective on SDN Programmability," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 2, pp. 1255–1272, 2016.

[4] T. Benson, A. Anand, A. Akella, and M. Zhang, "Visual Network Description: A Customizable GUI for the Creation of Software Defined Network Simulations," in *Proceedings of the European Multidisciplinary Society for Modelling and Simulation Technology / European Simulation Multiconference (ESM 2013)*, no. Lantz, (Lancaster), pp. 149–153, 2013.

[5] B. Pinheiro, R. Chaves, E. Cerqueira, and A. Abelem, "CIM-SDN: A Common Information Model extension for Software-Defined Networking," in *2013 IEEE Globecom Workshops, GC Wkshps 2013*, pp. 836–841, Ieee, dec 2013.

[6] V. Hazlewood, K. Benninger, G. Peterson, J. Charcalla, B. Sparks, J. Hanley, A. Adams, B. Learn, R. Budden, D. Simmel, *et al.*, "Developing applications with networking capabilities via end-to-end sdn (dances)," in *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale*, p. 29, ACM, 2016.