A High Level Rule-based Language for Openflow

A Proof of Concept based on Floodlight

Mehdi Mohammadi
Computer Science Department
Western Michigan University
MI, USA
mehdi.mohammadi@wmich.edu

***Abstract*—This paper proposes XML-Defined Network policies (XDNP) a new high level language based on XML notation to describe network control rules in Software Defined Network environments. We rely on existing OpenFlow controllers specifically Floodlight but the novelty of this project is to separate complicated language- and framework-specific APIs from policy descriptions. This separation makes it possible to extend the current work as a northbound higher level abstraction that can support a wide range of controllers who are based on different programming languages. By this approach, we believe that network administrators can develop and deploy network control policies easier and faster.**

Keywords—Software Defined Networks; Openflow; Floodlight; SDN compiler; SDN programming languages; SDN abstraction.

I. INTRODUCTION

By Software-Defined Network (SDN) technology, network engineers and administrators can control and manage network services through abstraction of lower level functionality. This end can be achieved by splitting the system that makes decisions where to send the packets (control plane) from the underlying systems known as data plane that just are in charge of forwarding packets to the selected destinations. In order to be practical, SDN needs some mechanism for the control plane to interact with the data plane. OpenFlow [1] is such a protocol that has been used in network research community in recent years. There are couples of OpenFlow controller frameworks such as POX[1], NOX[2], Beacon[3], Floodlight[4], Trema[5], NodeFlow[6] and Ryu[7]. The emerge of OpenFlow simplified network management by providing high level abstractions to control a set of switches remotely. An OpenFlow framework requires network engineers or administrators to write code programs to control and manage data traffic in their network and control network devices. However, one potential problem that network engineers face is that the programming languages that support OpenFlow are complex and administrators are required to know much irrelevant information to develop and deploy a control policy. The problem will be more prohibitive for a beginner network engineer who does not have a good background in the programming language of the controller. There are other challenges for programmers including [2]: interaction between concurrent modules, low-level interface to switch hardware, and multi-tiered programming model. A simple and unified language in a higher abstraction level that does not depend on a specific language can fill this gap. We are thinking about a text-based scheme like XML by which a human-friendly semantic of operations is developed for policy description. XML has been used widely in network management and configuration protocols like NETCONF. This work can be annexed to the current Openflow controller as a top layer service.

[1] http://www.noxrepo.org/pox/about-pox/
[2] http://www.noxrepo.org/
[3] https://openflow.stanford.edu/display/Beacon/Home
[4] http://www.projectfloodlight.org/
[5] http://trema.github.io/trema/
[6] http://garyberger.net/?p=537
[7] http://osrg.github.io/ryu/

II. RELATED WORKS

XML documents are widely used to describe systems, to configure or control them. One attractive usage is code generation. There are several works that try to use XML notation as a representation of source codes. JavaML [3] is such a work that represents java source code in XML notation. The source code representation in JavaML is in a way that constructs like superclasses, methods, message sends, and literal numbers are all directly represented in the elements and attributes of the document content. XML notation is also used in another work named srcML [4] by which structural information is added to unstructured source code files. srcML aims to enhance source code representation by adding syntactic information obtained from parse tree.

In [5], Liang et al. proposed a proof of concept to use XML as a description for OpenFlow Networking experiments. They defined networking experiments in a hierarchical model in which the experiment is in the highest level, and each experiment contains information, topology, deployment, control, and output components. However, they just provided the format description, but not mentioned by which way they generated their XML parser. Furthermore, they have not provided a full evaluation of their system in the real environments. Our work differs from their work in the way we design a platform by which network engineers can describe their network control rules by XML notations regardless of the underlying topology or network elements. In other words, Liang's work focuses on description of network environment not the control of network traffic and behavior.

There are several works and projects aim to propose a higher level abstraction above the OpenFlow APIs in their development frameworks [2, 6-8]. Frenetic [2] which has been implemented in python emerged with some simple rules including predicate-action pairs, in which actions support filtering, forwarding, duplicating, and modifying packets. Later it included other more complicated operators like packet processing functionalities [6]. Pyretic [8] as a modern SDN programming language based on Frenetic and beyond the current parallel composition operator, presents two more complex abstractions: sequential composition operator and applying control policies over abstract topologies. By these abstractions the development of modular control programs becomes simpler.

III. Software Define Networking

Software-Define Networking defines two separate layers as the new architecture of network environments. A data plane that is supposed to do operations like buffering packets, forwarding, dropping, tagging and collecting packet statistics. Control plane on the other hand, may have the algorithms to track the dynamic topology of the network and has route processing capability. Control plane usually consists of a separate powerful machine called controller. The control plane uses its computed data and the data plane's statistics to manage and govern a set of dependent switches by installing or removing packet forwarding rules over them.

OpenFlow as a realization of SDN follows this two-layer architecture. In a typical OpenFlow network, if a switch can find a rule match to the received packet in its flow table, then it proceeds with that rule. Otherwise, the packet is sent to the controller for more processing. Controller examines packet header and establishes a rule based on that packet. The next similar packets arrived to the switches then are not required to go to the controller since the switches have appropriate rule to process them. Although forwarding packets to the controller increases their latency but it occurs not very much.

IV. The Proposed System

Our XML translator consists of a lexical analyzer (lexer) and a syntax analyzer (parser). Lexer tokenizes the input file and matches the tags, attributes, identifiers, constant values and so on based on regular grammars. Syntax analyzer on the other hand performs syntactic analysis of the input file and if it does not found any problem in this step, generates appropriate Java source code. The overall architecture of the system is depicted in the following figure.

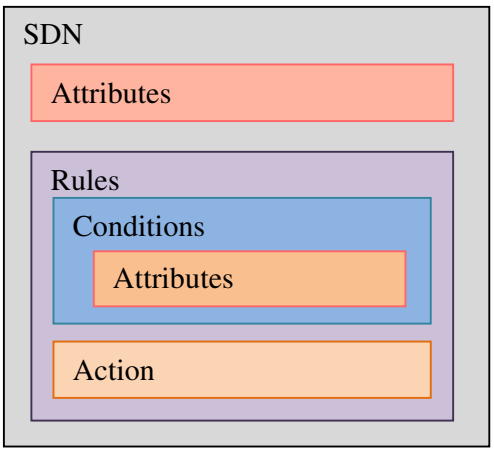

Figure 2. Format of a policy description file

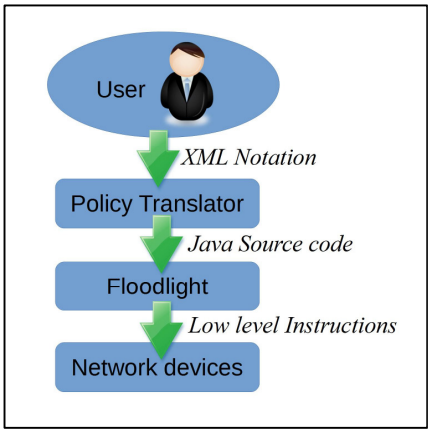

Figure 1. The overall architecture of the system

A. Language Specification

The overall design of an XML file to be used as control program should follow the format of figure 2. Rule description is defined in a hierarchical structure in which at the top level, we define the class name in the SDN element. Then a list of rules contacting zero or more rule elements should be declared. Inside each rule, one or more conditions are expressed. To have compositional conditions, condition elements support logical operators which are stated by attribute "connector". For example, if a condition element has an "or" connector, it will be joined to the previous condition by logical "or" operator in java source code. The conditions themselves comply with a simple pattern "variable op value". Variables can be chosen from src_ip (source IP), dest_ip (destination IP), src_prt (source port) and dest_prt (destination port). The current supported operator is equal sign. Value can be a port number or IP address. An example of XML file is shown in figure 3 and its Java source code is presented in Appendix A.

The XML sample contains two rules. The first one says that all the packets who are going to IP address 10.0.0.2 or who they are coming from IP address 192.168.0.1 should be forwarded to port 1 of the switches (which assigned to IP address 10.0.0.1). The second rule indicates that each packet originated from Telnet service (port 23) should be dropped (specified by port 0). This example shows that it is possible to implement all network policy management schemes and services like firewalls or load balancing with this XML notation.

B. Lexical Analizer

We designed our lexer with LEX format by which we defined the matching patterns for XML elements, attributes and literals that are needed in a typical network controller. The

```
<SDN class="Demo">
  <rules>
     <rule>
       <condition>destIP=10.0.0.2
       </condition>
       <condition connector="or">
           srcIP=192.168.0.1
       </condition>
       <action>outPort=1
       </action>
     </rule>

     <rule>
       <condition>srcPort=23
       </condition>
       <action>outPort=0
       </action>
     </rule>
  </rules>
</SDN>
```

Figure 3. A sample XML policy description

source of lexer file has to be fed to Flex to generate a c source code. Tag names are considered as keywords in the script and are required to comply exactly with the XML specification.

C. *Syntax Analizer*

The grammar section of translator is defined in syntax analyzer. We used YACC tool to generate the translator. In its input file, we define all the tokens that are introduced in the lexer, grammar production rules that examine the syntax of the XML file and the output associated with each production rule leading to code generation for XML file. Syntax analyzer description follows the BNF notation for Context Free Grammars.

Each string in the input xml file that is not matched to the designed tokens and rules will cause an error to the program and termination of the program with an error message.

V. CONCLUSION

This paper described a new approach of software-defined networks which presents a higher level of abstraction compared to current software-defined network programming languages. We defined a script language based on XML notation by which network administrators can define control policies without concerning about the complexities of underlying controller framework. Indeed, this will make software-defined networking easier and more attractive for network administrators.

This work opens up the opportunity of using service oriented architecture and web services as a model of collaboration between SDN controllers (e.g. a controller offers load balancing or firewalling).

Extending this work to support more API's and more complicated scenarios is intended to be done in the future works. Technically, examining and manipulating other OpenFlow headers is possible by this approach. Supporting more controller frameworks with different languages is also planned for next phases.

REFERENCES

[1] 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, pp. 69-74, mar, 2008.

[2] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story and D. Walker, "Frenetic: A Network Programming Language," *SIGPLAN Not.,* vol. 46, pp. 279-291, sep, 2011.

[3] G. J. Badros, "JavaML: a markup language for Java source code," *Computer Networks,* vol. 33, pp. 159-177, 2000.

[4] J. I. Maletic, M. L. Collard and A. Marcus, "Source code files as structured documents," in *Program Comprehension, 2002. Proceedings. 10th International Workshop On,* 2002, pp. 289-292.

[5] J. Liang, Z. Lin and Y. Ma, "OF-NEDL: An openflow networking experiment description language based on XML," in *Proceedings of the 2012 International Conference on Web Information Systems and Mining,* Chengdu, China, 2012, pp. 686-697.

[6] C. Monsanto, N. Foster, R. Harrison and D. Walker, "A Compiler and Run-time System for Network Programming Languages," *SIGPLAN Not.,* vol. 47, pp. 217-230, jan, 2012.

[7] C. J. Anderson, N. Foster, A. Guha, J. Jeannin, D. Kozen, C. Schlesinger and D. Walker, "NetKAT: Semantic Foundations for Networks," *SIGPLAN Not.,* vol. 49, pp. 113-126, jan, 2014.

[8] C. Monsanto, J. Reich, N. Foster, J. Rexford and D. Walker, "Composing software-defined networks," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation,* Lombard, IL, 2013, pp. 1-14.