软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(4):993−1008 [doi: 10.13328/j.cnki.jos.005028]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

软件定义网络中北向接口语言综述*

于 洋[1,2,3], 王之梁[1,3], 毕 军[1,3], 施新刚[1,3], 尹 霞[2,3]

[1](清华大学 网络科学与网络空间研究院,北京 100084)

[2](清华大学 计算机科学与技术系,北京 100084)

[3](清华信息科学与技术国家实验室(筹)(清华大学),北京 100084)

通信作者: 王之梁, E-mail: wzl@cernet.edu.cn

摘 要: 软件定义网络(software defined networking,简称 SDN)的产生使得网络中的数据平面与控制平面相分离,网络中的控制逻辑集中于控制器上,运行于控制器上的网络应用使得网络变得更加简单可控和灵活.软件定义网络中的北向接口是指控制器与网络应用之间进行通信的接口.在软件定义网络应用研究与开发的过程中,北向接口占据着一个重要的地位.综述了 SDN 中北向接口的编程语言,首先介绍北向接口编程语言的研究背景,然后根据编程语言的抽象程度、编程模型、实现机制以及是否引入新功能这 4 个方面将编程语言分类,详细介绍每个类别下各种北向接口语言的结构和核心特性,最后结合语言的应用场景对编程语言进行横向比较,进而展望了北向接口编程语言未来的研究方向.

关键词: 软件定义网络;北向接口;编程语言

中图法分类号: TP393

中文引用格式: 于洋,王之梁,毕军,施新刚,尹霞.软件定义网络中北向接口语言综述.软件学报,2016,27(4):993−1008. http://www.jos.org.cn/1000-9825/5028.htm

英文引用格式: Yu Y, Wang ZL, Bi J, Shi XG, Yin, X. Survey on the languages in the northbound interface of software defined networking. Ruan Jian Xue Bao/Journal of Software, 2016,27(4):993−1008 (in Chinese). http://www.jos.org.cn/1000-9825/5028.htm

Survey on the Languages in the Northbound Interface of Software Defined Networking

YU Yang[1,2,3], WANG Zhi-Liang[1,3], BI Jun[1,3], SHI Xin-Gang[1,3], YIN Xia[2,3]

[1](Institute for the Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China)

[2](Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

[3](Tsinghua National Laboratory for Information Science and Technology (Tsinghua University), Beijing 100084, China)

Abstract: Software defined networking (SDN) is a research trend as it decouples the control plane from the data plane. SDN applications are essential since they can be used to make the network simple to manage, flexible, more secure and more powerful. Northbound Interface is the communication interface between the controller and applications, it plays an important role in the process of the research and development of SDN. The state of the art of the programming languages in the SDN Northbound Interface is surveyed in this paper. It first summarizes the research background of the programming languages in the Northbound Interface. By classifying these languages into different categories according to their abstraction, programming model, implementation mechanisms and whether introducing new features, the study analyzes the key characteristic and language structure in each category. Incorporating with application scenarios in SDN, this paper compares the advantages and disadvantages of each language, and at the end discusses the future research trend.

* 基金项目: 国家高技术研究发展计划(863)(2015AA016105, 2013AA013505); 国家自然科学基金(61202357, 61472213)
Foundation item: National High Technology Research and Development Program of China (863) (2015AA016105, 2013AA013505); National Natural Science Foundation of China (61202357, 61472213)
收稿时间: 2014-11-07; 修改时间: 2015-06-12; 采用时间: 2016-01-03; jos 在线出版时间: 2016-01-16
CNKI 网络优先出版: 2016-01-18 13:51:00, http://www.cnki.net/kcms/detail/11.2560.TP.20160118.1351.005.html

Key words: software defined network; northbound interface; programming language

随着互联网的发展,网络结构越来越复杂,网络上部署的应用也越来越多,在传统的网络体系结构下很难部署新型的网络功能.在这种情况下,软件定义网络(software defined networking,简称 SDN)[1]应运而生,SDN 通过将网络中的数据平面和控制平面分离开来,实现对网络设备的灵活控制.这种数据和控制平面分离技术给网络带来了更大的灵活性和可操作性.在 SDN 中,网络由中心控制器集中控制下层的一系列分布式的网络设备,包括交换机、路由器以及中间盒(middlebox)等,控制器可以获取网络中各个资源的详细信息,下层的设备(例如交换机)仅仅实现简单的报文转发功能.图 1 是 SDN 标准化组织开放网络基金会(Open Networking Foundation,简称 ONF)提出的 SDN 体系结构,下层的网络设备是 SDN 中的数据平面,构成了 SDN 的基础设施层(infrastructure layer);SDN 的中心控制器软件是 SDN 中的控制平面,构成了 SDN 的控制器层(controller layer);运行于控制器之上的便是 SDN 中的业务应用,构成了 SDN 的应用层(application layer).SDN 中的控制器通过南向接口与基础设施层的网络设备进行通信,实现对数据平面的控制,通过北向接口的 API(application programming interface,应用编程接口)为上层的应用服务.

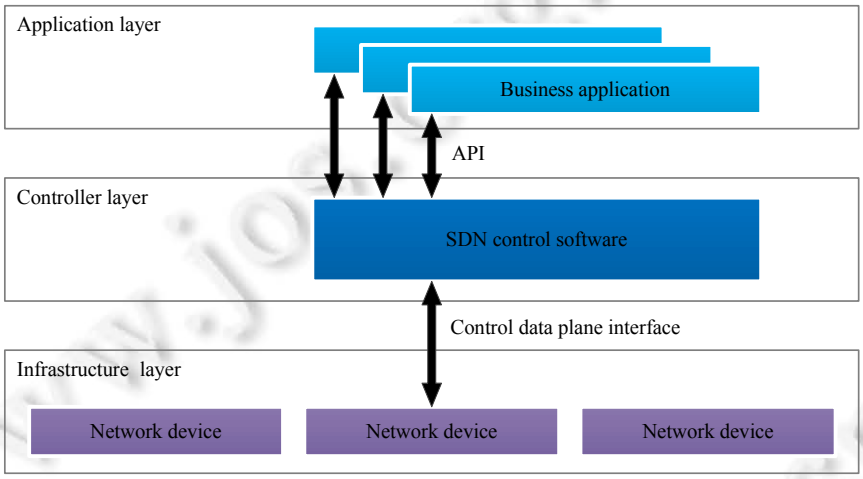

Fig.1 SDN architecture[1]

图 1 SDN 体系结构图[1]

SDN 让网络的可控性变强.传统网络的灵活性和可控性较差,增加部署协议或规则是很繁琐的一项工作;但在 SDN 中,由于网络设备的所有控制逻辑已被集中在 SDN 的中心控制器中,使得网络的灵活性和可控性得到显著增强.因为网络设备的所有控制逻辑已被集中在 SDN 的中心控制器中.控制器可以提供网络的全局视图以及下层设备的信息,所以,编程者可以在控制器上编写策略,例如负载均衡、防火墙、网络地址转换(NAT)、虚拟专用网络(VPN)等功能,进而控制下层的设备.可以说,SDN 中的应用可以体现 SDN 可编程的这一核心特性.

北向接口(north bound interface,简称 NBI)的出现繁荣了 SDN 中的应用.北向接口主要是指 SDN 中的控制器与网络应用之间进行通信的接口,一般表现为控制器为应用提供的 API 编程接口.北向接口可以将控制器内的信息暴露给 SDN 中的应用以及管理系统,从而可以利用这些接口去进行,如请求网络中设备的状态、请求网络视图、操纵下层的网络设备等等操作.利用北向接口提供的网络资源,编程者可以定制自己的网络策略并与网络进行交互,充分利用 SDN 带来的网络可编程的优点.SDN 中的北向接口不仅可以让编程者充分利用网络资源,还可以细粒度地在流的级别上控制网络资源.为了让一个应用或者服务正确地运行于网络之上,网络的管理员不再需要对现有网络设备进行升级以及复杂的配置,只需要向控制器请求相应需要的资源.

鉴于北向接口有这样的优势,ONF 成立了相关的北向接口工作组 North Bound Interface Working Group(简称 NBI-WG),它致力于建立 SDN 中北向接口的原型及其信息模型,为应用开发者制定标准化的北向接口.近几年也纷纷涌现了众多针对北向接口的编程语言,它们的出现让编写 SDN 网络应用的开发部署等工作变得更加

简捷.关于北向接口语言的情况,在最近的几篇 SDN 综述[2−5]中均有提及.但是,目前仍然还没有一篇关于北向接口语言的详细综述,所以本文针对北向接口的编程语言,分类介绍它们的详细特征,同时重点针对编程模式讨论各个语言的不同,进而为后续研究北向接口的工作指明方向.

本文第 1 节首先介绍 SDN 中北向接口的编程语言出现的背景.第 2 节对各种语言进行纵向的分类.第 3 节针对每一类语言给出各个语言的详细特征,包括语言产生的动机、核心思想、应用领域等.第 4 节横向比较总结各个分类下的编程语言.第 5 节对北向接口的研究领域做出展望.第 6 节给出总结.

1 SDN 中北向接口语言的研究背景

SDN中的北向接口加强了网络的可编程的特性.SDN应用的逐渐繁荣,北向接口充当着重要的角色.因为编程者可以利用北向接口提供的网络资源部署自己的网络策略,即网络应用,运行于 SDN 控制器之上,进而控制网络的行为.

在现有的主流控制器,如 NOX[6],POX[7],FloodLight[8],Beacon[9],OpenDaylight[10],OpenContrail[11]中,都存在这样的北向接口,编程者如果需要部署一项新的应用,只需利用它们提供的相应 API,便可以实现对下层设备流级别的控制,但是,这些控制器中提供的都是一个低级的流表规则的管理系统,存在着如下缺点:

(1) 为了让一个网络策略或是管理系统能够正确运行,编程者需要不停地适应 SDN 中的内部结构,需要考虑更多的便是系统的内部细节,而不是算法或是策略本身,这可能会导致算法出现错误或是软件层面上的错误.同时,利用 OpenFlow[12]协议的细节编写的北向接口中低层次的指令系统很难去阅读和维护,也很难调试.那么用低级语言去编写应用的过程便成为了一个既耗时又容易出错的过程.

(2) 低层次的编程语言让应用的并行运行变得困难,因为不同的应用有不同的转发规则,而且它们的转发规则很容易产生冲突.例如,在同一个网络拓扑中,应用 *A* 需要过滤来自源地址为 10.0.0.1/8 的流量,然而另一个应用 *B* 需要转发源地址为 10.0.0.1/8 的流量,这两个应用如果并行运行就可能会产生冲突,在低级语言中,为了解决这样的冲突问题,需要编程者手动设置每一条流表的优先级,这个过程又提高了编程的复杂度.

(3) 在众多 SDN 的应用中,很多应用需要将下层的物理网络切片为虚拟网络.应用需要获取一个虚拟的网络拓扑,然后在虚拟的网络上进行模块化的编程;或者是把同一个物理交换机抽象为一系列虚拟交换机并出现在不同的虚拟网络中.这两个过程即便采用低级语言也都很难实现,所以,简化网络切片的过程需要一个高度抽象的北向接口.

基于这样的问题,创建易于维护的高层次的北向接口编程语言是很有必要的,北向接口需要一个高层次的抽象以方便应用的编写.北向接口中高级的编程语言可以屏蔽底层设备的信息,使得应用的编写过程不再依赖于底层的细节.另外,高级语言比低级语言更加抽象、简洁,易于学习与维护,也有利于代码的重利用和编程的模块化.

2 SDN 北向接口编程语言的分类

近年来,编程语言经历着飞速的发展,从汇编等低级语言,逐渐发展到 Java,Python 这样的高级语言,这个过程给计算机领域的发展带来了极大的飞跃.与此类似,SDN 中的编程语言也在经历着类似的进步,从类机器语言的低级语言,逐渐发展到高层次的编程语言.高级的编程语言都有一个共同的特征,就是能够更加简洁地表达网络应用,给编程者、网络参与者都带来方便,使得网络应用的设计实现和部署过程变得简单.本节从抽象程度、编程模型、实现机制以及是否引入新功能这 4 个方面对北向接口的高级编程语言做出了分类.然后针对每一类编程语言,给出了典型代表.

2.1 北向接口语言的分类

北向接口语言从不同的角度可以存在不同的分类标准,主要标准为以下几个方面,如图 2 所示,有抽象程度、编程模型、实现机制以及是否引入新功能这 4 个方面.在不同的分类标准下,还有更加详细的分类,下面将

对北向接口语言的 4 个分类标准进行详细介绍.

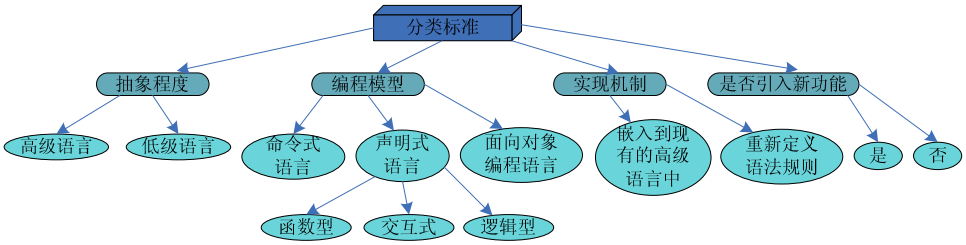

Fig.2 Classification standard of languages in the northbound interface

图 2 北向接口语言的分类标准

2.2 以抽象程度为标准分类

从北向接口中的语言对底层设备的抽象程度来区分,分为低级语言和高级语言.类比于计算机编程语言的分类,与编程语言中按照是否封装计算机硬件指令作为区分高级语言和低级语言的标准一样,按照抽象程度分类是希望根据 SDN 北向接口语言是否封装了控制器与硬件的配置细节以及它们之间的南向接口指令来作为区分.北向接口中的高级语言是指封装了 SDN 控制器中不同的性质和功能的[1]、具有更高可读性的、更加方便于编写应用的语言,抽象程度较高;而低级语言是指与硬件设备配置信息相关的、需要处理硬件层面流表细节的语言,抽象程度较低.它们之间存在语言表达能力上的区别,高级语言的表达能力更强.

在低级语言中,典型代表有基于 OpenFlow、P4[13]、POF(protocol-oblivious forwarding)[14]设计的控制器中提供的北向接口,例如 NOX 提供的 C++以及 Python 的 APIs,还有 Beacon、FloodLight、OpenDaylight 中提供的 Java 的 APIs.另外,tinyNBI[15]也属于低级语言中的一种.

其中,tinyNBI 与上面所述的接口不同的是,它重构了现有的 5 个版本的 OpenFlow 协议提供的 API,把这 5 个版本整合为一个简单的兼容所有标准 OpenFlow 的 API,tinyNBI 的核心功能在于通过对 SDN 数据平面原语的准确抽象,为高级的编程语言提供更加清晰的底层接口.

另外,对于控制器中提供的 RESTful API(representational state transfer API),这类北向接口也与其他有所不同,它可以被任何语言所编写的应用所用,体现为 Web 请求的方式,例如在 Juniper 的 Contrail 和 FloodLight 中的图形化用户界面(graphical user interface,简称 GUI)里面都利用了 RESTful API 来编写.

北向接口中的低级语言直接利用交换机的硬件接口去编写转发行为的流表,编程者需要充分考虑硬件的细节,还需要手动去维护策略生成的流表的优先级信息.同时,由于流表下发到交换机中的异步的行为,在这个过程中,数据平面内可能会存在未被处理的报文,所以编程者还需要手动维护数据平面的一致性,以确保规则可以正确指导下层设备中报文的转发行为.这些低层次的语言给开发者带来了麻烦,同时也不利于应用的模块化以及代码的重构.

与低级语言相对的便是北向接口中的高级语言,占据了北向接口语言中的绝大部分,典型代表有 pyretic[16]等,高级语言对底层设备提供的接口进行了封装,可以让编程者关注于网络协议和策略的相关逻辑,而不用关心数据平面的细节,简化了编程者的工作.同时,利用高级语言可以进行模块化的编程,甚至可以支持网络策略的并行编程,代码易读且便于维护,加快了网络协议开发与部署的过程.

2.3 以编程模式为标准分类

从各个编程语言的编程模式上来区分,还可以将编程语言分为命令式语言(imperative language)、声明式语言(declarative language)、面向对象语言(object-oriented language).类比于计算机编程语言中的编程模式,北向接口语言也可以通过语言的基础风格以及语言中组织元素和数据结构的方式作为分类标准.所以,低级语言的编程模式是在使用时需要考虑不同设备的配置细节、控制器内部的实现细节,甚至还有流表是否会存在冲突、如何维持数据平面的一致性等等.低级语言的编程模式大部分都是命令式的,而高级语言的编程模式则可以分为

命令式语言、声明式语言(包括逻辑型、交互式、函数型语言等)及面向对象语言[17]等.这种分类方式借鉴了文献[3]中的关于编程语言的分类,在本文中对各类北向接口编程语言进行了更细致的分类.在计算机编程语言中,命令式语言[17],如 Python、C、Ruby 等,它们的编程模式是命令系统如何去做事情,无论编程者想要的是什么,这种语言都会按照程序的指令完成任务.与此相对,声明式语言[18],如 SQL 等,更强调于编程者想要实现的是什么,而语言自身可以对问题进行描述.面向对象语言,如 C++、Java 等,则是将对象作为程序的基本单元,将程序和数据封装其中,可以提高软件的重用性、灵活性和扩展性.例如,声明式语言 SQL 封装了检索时数据库是如何获取数据的信息,而让使用者关注于自己希望获取什么样的数据或是自己需要做何种操作;而在命令式语言中,检索数据的过程则需要编写一系列的指令去指明执行的流程.由于在一种语言中,可能存在多种编程模型,所以一种语言可能不仅仅只属于一种类型.

在现有的北向接口的语言中,命令式语言、声明式语言、面向对象语言的概念都与计算机编程语言中的概念相应地保持一致.其中,声明式编程语言占据大部分.其中,又可以分为逻辑型语言、函数型语言以及交互式语言.这种分类是基于语言的编程模式的,鉴于有些语言,如 Nettle[19],同时存在交互式编程模型和函数型编程模型,所以它便可能出现在多个分类中.交互式语言的代表有 FML[20],Nettle,Procera[21];函数型语言如 FatTire[22],FlowLog[23],Frenetic[24,25],HFT[26],Maple[27],Nettle,NetCore[28,29],Procera 等;逻辑型语言的例子有 Flog[30],HFT,Merlin[31],Assertion Language[32];面向对象类语言如 pyretic,Splendid Isolation[33],NCL[34];命令式语言的代表有 pyretic.下一节我们将针对每一个分类介绍 SDN 中北向接口编程语言的详细信息.

2.4 按照是否提供新功能分类

北向接口出现的主要目的是简化编程者的工作,让网络协议的开发和部署过程变得更加简洁.北向接口可以提供的基本功能是提供控制器与应用或者更高层控制程序之间的通信,如从控制器获取信息,应用策略的下发等;但在众多北向接口语言中,还有一类语言能够提供新的网络功能,如网络验证、网络切片等功能.例如,FatTire、FlowLog、NetKAT[35]、Flog、Merlin,还有 Splendid Isolation.它们的出现除了简化网络编程,还让网络中的其余功能,如模型检查、动态验证、网络虚拟化等过程变得更加简单.

FatTire 利用正则表达式描述网络路径,保证每一条流在其原始路径出现故障时还有可选路径.FlowLog 和 Flog 可以处理模型检查、动态验证等过程,NetKAT 可以提供一个完备的数学理论基础去指导网络原语,同时也可以提供便捷的网络验证功能.Assertion Language 可以提供嵌入应用的网络验证功能.而 Splendid Isolation 则让网络虚拟化、网络切片、流量隔离等过程更加简单.

2.5 按照语言的实现机制分类

各类语言在实现机制上存在一个共同点,即基本上所有的高级语言都由两部分组成:上层的高级语言以及下层的编译器.高级语言可以方便编程者利用语言提供的抽象接口完成自己所需的网络策略,而下层的编译器负责将编程者编写的策略编译为流表规则下发到数据平面的设备中.在这样的实现机制中,也存在两种语言,一类是嵌入到现有的高级语言中,一类是自己定义一套语法和语义规则.前者主要包括 Procera,HFT,PANE[36],Maple,FlowLog,FatTire,NetKAT,Flog,Merlin;后者主要包括 FML,Netttle,Frenetic,pyretic,NetCore,NCL,Splendid Isolation,Assertion Language.

在嵌入到现有的高级语言的北向接口语言中,FML 基于非递归的 DATALOG[37]语言,Nettle 嵌入到 Haskell 语言中,Frenetic-OCaml 基于 OCaml 语言,pyretic 基于 Python,NCL 基于 Java,Splendid Isolation 基于 Python.

3 SDN 北向接口的编程语言

上一节介绍了北向接口中编程语言的分类,下面将详细介绍北向接口中的各类编程语言的详细信息,如产生背景、核心功能、优缺点等等.由于在北向接口的编程语言中,大部分是编程模型为声明式语言的高级语言,同时,编程模型这个分类标准下的子类也最多,所以本节将按照编程模型这个分类标准来详细介绍各种语言.

3.1 北向接口中的交互式语言

交互式语言利用交互式编程模型,交互式编程模型主要面向数据流或者动态变化的变量编程,既能很方便地描述静态的数据流,也能轻松地表达动态变化的函数.交互式编程语言能够利用事件作为触发点驱动网络中的交互行为,强调利用网络编程而代替对于网络参数的配置.举一个简单的例子,在 $a=b$ 这条命令式的语句中,是把变量 b 的值赋给了变量 a,然而如果改变 b 的值,在命令式编程模型中并不会改变 a 的值.但是,在交互式语言中,b 的变化可以反映给 a,实现了变化的传播过程.与此类似,在北向接口中,交互式语言也可以将网络中的事件作为驱动,如果网络中存在一个事件的变化,这个变化也会反映给相关的网络事件,驱动它们做出改变,从而实现变化的传播.在北向接口的交互式语言中,典型代表有 FML、Nettle 和 Procera,其中,由于 Nettle 和 Procera 也存在函数型编程模型,所以它们也属于函数型语言.利用这 3 种语言可以处理如权限控制(ACLs)、虚拟网络(VLANs)等应用.在这种语言的基础上编写的应用都会转化为一些类如 allow,deny 等策略,这些策略会被编译为流表下发到下层的设备中去,进而指导网络的行为.但是,这 3 类语言也有不同点,见表 1.

Table 1 Main difference in reactive languages

表 1 交互式语言之间的不同点

编程语言	设备要求	核心特征
FML	仅 OpenFlow 设备	定义网络关键字,简化网络配置过程
Nettle	仅 OpenFlow 设备	把来自设备的信号转化为事件流,供编程者处理
Procera	所有设备	高度抽象的接口可以处理动态的网络协议以及产生与网络交互的实时性行为

3.1.1 FML[20]

FML(flow-based management language)[20]是可以方便地管理企业网络配置的声明式语言,它是专门针对 OpenFlow 设备提出的一种交互式语言.它的实现基于 NOX 控制器,采用 NOX 中用回调函数实现事件驱动的机制,把 NOX 里面的细节语句封装为语言的关键字,实现网络中的交互过程.

FML 被设计为可以替代现有的网络配置的方案,编程者可以利用 FML 所作的抽象,方便地表达网络中的各种常见配置.FML 是针对网络中流的管理而设计出的高级语言,针对每一个应用,编程者都可以利用 FML 定义一系列其需要的关键字(如 deny,allow 等等),同时由网络中的超级用户(一般指管理员)对网络中的用户、主机分配权限.FML 中可以利用关键字的优先级去处理流表的优先级,解决了多个应用带来的流表规则冲突的问题.

经过测量发现,FML 生成的流表规则确实能够在线性的时间内匹配大部分的流量,运行效率较高.同时,对于每一个应用,也可以引入新的关键字来处理复杂场景下的问题.

总体来说,FML 主要是为了简化网络策略的配置而产生的对流控制的抽象,它灵活、准确,可以同时被多个网络编程者所用;并且方便用户学习;也可以用作 SDN 北向接口的抽象语言,这是它的主要优点.但是它也有自身的局限性:比如只限于产生对网络中流级别的控制,不能用于转发策略随时间变化的动态协议.

3.1.2 Nettle[19]

Nettle[19]是仅支持 OpenFlow 设备的交互式编程语言,意在简化 SDN 中控制平面编程模型的复杂性、低级性以及容易出错等缺点.Nettle 借鉴了领域专用语言(domain specific language,简称 DSL)[38]以及功能交互语言的设计思想,继承了交互式编程模型的表达性强、拥有完备的数学基础等优点.

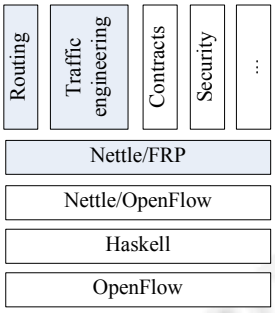

Fig.3 Nettle layered system architecture[19]

图 3 Nettle 层级的系统结构图[19]

在 Nettle 的层级结构中,底层为 OpenFlow 设备,上一层的 Haskell[39]是 Nettle 的宿主语言,供 Nettle 内嵌于其中.基于 Haskell 上层的 HOpenFlow 把 OpenFlow 协议进行了抽象,再上一层便是功能交互的语言模型,模型的上层便是对领域专用语言 DSL[38]的扩展.每一种扩展都是一个应用的模型,例如可以有一个 DSL 用来做访问控制,另一个用来做流量工程,或者其他的网络应用.如图 3 所示.

Nettle 聚焦于 OpenFlow 交换机之间的控制信息流,把来自交换机的信息作为一个个信号,都分别转换为事件流,细粒度地控制交换机行为.传统的语言事件驱动采用回调或者循环的方式进行,但是,Nettle 通过不连续的周期性随时间变化的信号组成的事件流来带动事件驱动的过程.它不仅向编程者屏蔽了底层设备细节,更可以根据编程者的输入随时调整需要统计的网络信息,对编程者更加友好.

3.1.3 Procera[21]

Procera[21]也是一种交互式语言,然而,它却不仅仅能支持 OpenFlow 设备.Procera 的架构更像是基于一般 SDN 控制器的一个上层控制器,这个上层控制器会给编程者一些特定的接口,让编程者轻松编写事件驱动的程序.经由 Procera 控制器的输出可以直接被底层的控制器运用.

如果网络中的应用有实时性处理网络状态改变的需求,仅仅利用 FML 是无法实现的,所以在这些应用中,Procera 的事件驱动的编程模型便体现出其重要性.

Procera 适用于交互性的网络策略(reactive policy),利用它定义出的策略可以转化为随时间变化的事件流.例如在网络用户设备流量监测这个应用中,管理员会给这个网络的所有设备分配一个最大使用的带宽和流量,如果在一段时间内监测到某些设备或用户的使用量超过了阈值,就会禁掉这个用户或设备的访问权限.由于每一个用户的月使用流量情况会通过一个接口被控制器所收集,监测到用户的使用量超过阈值这个过程并不困难,甚至是用传统的 FML 便可以方便测量,但是,如果一个被禁掉的用户下个月的使用量回到正常值,此时再利用 FML 就不方便描述和处理这个过程了,相比之下,Procera 可以通过如图 4 所示的代码片段表达这个事件驱动的过程,第 2 行中的 useTableSF 函数定期地将用户的使用情况累计到 useTable 中,第 6 行是查询用户的使用情况,第 7 行是查询监控情况,如果用户流量的使用量还未超过监控阈值,则允许继续使用,否则,禁掉用户使用流量的权限.由于系统每秒都会从设备中收集用户使用网络情况的监控日志,这个过程便是一个事件驱动的交互式过程.图 4 所示的代码片段也代表了交互式语言一般的实现形式.

```
proc world→do
  useTable←useTableSF-≺world
  capTable←capTableSF-≺world
  let policy req=
      let src=srcEthAddr reg
          use=lookupUse src useTable
          cap=lookupCap src capTable
      in if use≺cap then allow else deny
  returnA-≺policy
```

Fig.4 Example: Device usage caps[21]

图 4 例子:设备流量使用监控[21]

3.2 北向接口中的函数型语言

函数型语言[17]占据了北向接口高级语言的大部分,它们同时也都属于声明式语言.函数型语言会像定义数学上的函数那样定义程序以及子程序,函数型编程模型主要把运算过程尽量写成一系列嵌套的函数调用.例如:在数学表达式(2−1)×3+4 中,如果用传统面向过程的编程语言(C 语言)可能写成这样:

int *a*=2−1;

int *b*=3×a;

int *c*=*b*+4;

但在函数型编程中,就可以写成 int result=add(multiply(subtract(2,1),3),4),主要思想是把运算过程定义为不同的函数.但是,现在的大部分函数型语言都是“不纯”的,因为它们除了具备函数型编程模型外,还具备交互式,甚至命令式编程模型.不过,与其他语言相同的是,北向接口中的函数型语言也强调对网络编程过程的抽象与简化.函数型语言的代表有 FatTire,FlogLog,Frenetic,HFT,PANE,Maple,Nettle,NetCore,Procera,NetKAT 等.在这些函数型语言中,Frenetic,HFT,PANE,Maple,NetCore 侧重于简化编写报文转发策略,处理生成的流表规则出现冲突的情况;Nettle 和 Procera 在上一小节已经提到过,它们也同时属于交互式语言;其余的函数型语言也都相应地引

入了不同的特性.例如 FateTire 重在依赖正则表达式去描述网络策略,FlowLog 可以编写模型检查、动态验证、有状态的中间盒等应用,NetKAT 重点在于提供了一个完备的数学理论基础来指导网络原语,不仅简化了编程模型,还可以提供网络验证等功能.

下面将会在侧重于简化编写报文转发策略的语言中选取 Frenetic、在侧重于引入新的网络特性的语言中选取 FlowLog 来介绍它们的详细特征.同时,由于 NetCore 的服务能力强于 Frenetic,所以也会详细介绍这种语言,并与 Frenetic 做出比较.其余语言由于在语法结构或实现形式上有相似的地方,所以集中在一个小节内加以介绍.

3.2.1 Frenetic[24,25]

Frenetic 属于专门针对 SDN 北向接口而设计的领域专用语言(domain specific language,简称 DSL),它可以为上层应用提供相关函数的封装,所以属于函数型编程语言.主要解决 SDN 北向接口存在的以下问题:

(1) 控制器的可见性是有限的;

(2) 在控制器上的编程没有支持模块化;

(3) 交换机与控制器之间的通信采用的是异步的模式,这样,在规则下发之前的报文容易匹配不正确的流表规则,从而被不正确地转发或丢弃.

针对上面的情况,Frenetic 包含一个能产生高层次抽象的编程语言和一个实时的编译系统,前者支持模块化和并行化,有利于编程者简捷地编写网络应用,后者可以处理底层设备复杂的细节.Frenetic 采用类 SQL 语法定义查询性语言,重点在于描述报文的转发规则,处理报文级别的数据.利用 Frenetic 可以简化的过程分为以下 3 类.

(1) 查询网络状态.例如在 MAC(media access control address)地址学习的过程中,如果当 mac 地址 srcmac 出现在新的端口 inport 时,程序希望收到一个报文,此时的查询语句可以描述为

```
Select(packets)*GroupBy([srcmac])*SplitWhen([inport])*Limit 1;
```

(2) 寻找符合特定规则的报文信息.例如流量监控的协议可以表述为

```
def web_monitor():
    q=(Select(bytes)*Where (inport=2 & srcport=80)*Every(30))
    q>>Print()
```

(3) 生成报文转发规则并由实时的编译系统下发到交换机上去.下面的语句表述了利用 Frenetic 表达网络协议安装流表的过程:

```
def repeater():
    rules=[Rule(inport:1,[fwd(2)]),Rule(inport:2,[fwd(1)])]
    register(rules)
```

Frenetic 在这 3 类事件中可以提供高级的查询语言或协议语言,或者一个高层次的抽象,编程者不用去考虑协议生成流表的细节,因为这个过程完全交给实时编译系统去处理了.编程者只需考虑自己需要什么.

Frenetic 抽象出的接口可以控制网络中的报文.利用 Frenetic,网络中的每一个报文对于编程者都是可见的.同时,为了进一步简化编程,对于交换机发送给控制器的报文,编程者还可以选择只关注第 1 个 packetin 消息,屏蔽其余的报文.这样既提高了编程者的效率,也防止不重要的报文(例如在流表同步的过程中网络中的包)影响网络策略的整体逻辑,降低了控制器的负载.

Frenetic 将查询网络状态与更改网络状态这两个过程隔离开,同时支持并行化的编程,允许多个应用同时编写并运行于 SDN 控制器之上.另外,对于缓存的报文可以自动生成转发规则防止网络拥堵的状态出现.值得关注的是,Frenetic 通过给不同的流表规定一个整数的优先级来解决多应用之间存在的冲突问题.

此外,Frenetic 还可以支持在源码的级别进行网络验证,验证网络中的重要性质,比如网络验证中常见的 3 个问题:(1) 主机 A 和主机 B 之间是否可达;(2) 到主机 C 的路径中是否存在回路;(3) 是否所有的 SSH 流量都被隔离了.Frenetic 都可以通过编写高层次代码而做出回答.

3.2.2 NetCore[28]

与 Frenetic 类似,NetCore 也由两部分组成,表达报文转发策略的编程模型以及一个编译器可以将利用 NetCore 编写的策略编译为流表安装到下层交换机上.NetCore 是基于 Frenetic 的改进版本,它有 Frenetic 的 3 个基本功能,同时还有区别于 Frenetic 的如下特性.

(1) NetCore 包含了一个简化的"查询语言",可以分析流量历史记录.

(2) NetCore 里面的编译器采用了一种新的算法,这种算法可以生成高效的交换机类.首先,通配的流表替代了 Frenetic 中简单的精确匹配的流表去处理更多的报文;其次,安装流表的过程采用主动的方式,即不是通过报文触发控制器再生成流表,而是在报文到达控制器之前先把流表下发到交换机中,这样既减轻了控制器的负载,同时也减少了网络中缓存的报文.

NetCore 的核心提升在于一个可以支持应用并行运行的编译器,利用 NetCore 编写的策略如下所示:

SrcAddr:10.0.0.0/8\(SrcAddr:10.0.0.1∪DstPort:80)→{Switch 1}

经过 NetCore 的编译器之后,生成的流表规则如下所示:

SrcAddr:10.0.0.1:{}

DstPort:80 :{}

SrcAddr:10.0.0.0/8:{switch 1}

经过对 NetCore 的性能评价,NetCore 在编译器上确实作了优化,流表的命中率相比于 Frenetic 也有显著的提高.

3.2.3 FlowLog[23]

FlowLog 属于声明型语言中的函数型语言,与前几个重在抽象处理报文过程的语言不同,FlowLog 引入了新的特性,侧重点在于网络验证.主要解决网络验证领域的如下问题:

(1) 网络中会存在大量的缓存报文;

(2) 交换机里会安装大量的流表;

(3) 网络拓扑可能会相当复杂.

现有的对于 SDN 的验证大部分需要基于网络配置在数据平面进行,如报文头空间的分析[40](header space analysis).基于这样的问题,如果要在网络应用的层面进行网络验证,必须要将底层的细节做出抽象,屏蔽流表信息、报文信息以及复杂的拓扑信息,从而方便编程者编写验证逻辑.编程者只需验证一个或几个配置,便可以检测网络的可达性信息是否存在循环、回路等等.

FlowLog 的表达性强,而且分析能力也强,每一个 FlowLog 的程序都会定义为一个在控制器状态层面上的有限状态的传感器.FlowLog 可以掌握控制器状态,也可以改变控制器状态,并且对报文可以做出变量的修改,不局限于流级别的控制.同时,FlowLog 的编译器也可以将编程者编写的策略转化为交换机上的流表,这个过程同样在简化编程者的工作.

3.2.4 其他函数型语言

在函数型语言中,HFT[26],PANE[36],Maple[27]等与 Frenetic 类似,侧重于简化编写报文转发策略,处理生成的流表规则出现冲突的情况.它们又有各自的特点,HFT 提出了有优先级的流表的概念,用于定义和实现 SDN 中具备优先级属性的策略框架.而 PANE 则属于在 HFT 的基础之上,让网络上编写策略的过程变成一个人人可参与的过程.PANE 的核心是在描述什么样的网络参与者拥有什么样的权限去做出什么样的网络策略.Maple 的提出则是基于交换机的流表不支持否定匹配的情况,Maple 中的编译器可以自动生成具有优先级的流表规则,简化编程者的工作.

从实现层面上看,HFT,PANE 和 Maple 也有相似的地方,即都是用树状的数据结构去组织流表或策略.HFT 将 SDN 上的策略组织成一棵树状结构,树的每一部分可以独立地决定一个报文的转发行为.PANE 则采用共享树(share tree)这样的结构让管理员给网络参与者分配读或写的权限,中心化的控制器可以通过更改网络配置完成网络参与者(principal)的高层次的意图.Maple 提出了追踪树(trace tree,简称 TT)这样的模型,其重点也是在对

TT 模型进行改进和优化.

另外,在函数型语言中,还有一类语言类似于 FlowLog[23]一样引入了新特性,例如,FatTire[22]的新特性在于可以用正则表达式来描述网络的路径,可以发现网络链路中错误的配置.FatTire 利用网络容错的正则表达式,使得网络中各种容错的场景更容易理解.NetKAT[35]以 Kleene algebra 为理论基础去描述全网的结构,利用 Boolean algebra 为理论基础去描述下层交换机的行为,任何违背 KAT 公理的行为都会被 NetKAT 拒绝,这样也给网络验证带来了便利.

3.3 北向接口中的逻辑型语言

逻辑型语言[17]也属于声明式语言的一种,逻辑型语言主要基于形式逻辑,通过设定规则而不是设定步骤来解决问题,它由众多具有逻辑形式的句子组成,表达了某个领域问题的事实和规则.SDN 的编程语言中,Flog, HFT 以及 Merlin 都属于逻辑型编程语言,它们的共同点是利用逻辑型的编程模型去编写网络协议,从而简化协议中的逻辑模型.例如,Flog 将模型检查的功能简化,HFT 可以将生成网络中流表优先级的过程简化,Merlin 可以将权限分发给子网的过程简化等等.下面的表 2 从对设备的要求、控制对象级别、核心特征这 3 个方面描述了它们的不同点.HFT 作为函数型语言的一种,已经在上一小节提到.下面将详细介绍 Flog、Merlin 以及 Assertion Language.

Table 2 Main difference in logical languages

表 2 逻辑型语言的不同点

编程语言	设备要求	核心特征
Flog	正常交换机	以网络中的事件作为驱动,把 SDN 中的报文转发规则抽象为断言+行为+优先级模式
HFT	正常交换机	编译生成有优先级的流表
Merlin	网络中的所有中间盒设备	兼容已有的系统,支持静态协议以及动态改变的网络策略
Assertion language	正常交换机	在 SDN 上层的应用层面对网络进行验证

3.3.1 Flog[30]

```
# Network Events
flow(dstip=IP), inport=2→seen(IP).

# Information Processing
seen(IP)+→allow(IP).
allow(IP)+→allow(IP).

# Policy Generation
inport(2)|>fwd(1), level(0).
allow(IP)→srcip(IP), inport(1)|>fwd(2), level(0).
```

Fig.5 Stateful firewall[30]

图 5 有状态的防火墙[30]

Flog 属于逻辑型编程语言,也是声明式语言的一种,它结合了上文提到的 FML 和 Frenetic,把 SDN 的报文转发规则抽象为断言+行为+优先级的模式,属于事件驱动,也是在流级别上控制报文.与前两者不同的是,Flog 引入了新的编程特性,如模型检查、动态验证以及编写有状态的中间盒,这一点与 FlowLog 类似.以有状态的防火墙这个应用为例,利用 Flog 只需以下 5 行代码便可以实现这样的功能.由代码可以看到,Flog 是基于形式逻辑的语言,与函数型语言不同的是,并不是靠复杂的函数调用来解决问题.如图 5 所示.

Flog 结合了 FML 以及 Frenetic 的优点,并且克服了 FML 编程模型不灵活的缺点,但是,Flog 由于只限于流级别的控制,因而也存在一定的局限性.

3.3.2 Merlin[31]

Merlin 是一种基于正则表达式的高级逻辑型语言.第 3.2.3 节中介绍的函数型语言 FatTire 也利用了正则表达式,不过,在 FatTire 中,可以用正则表达式表达网络路径,保证每一条流在它的原始路径出现故障时还有可选路径,采用函数型编程模型;而在 Merlin 中,支持将抽象出来的网络函数应用于中间盒、终端以及硬件中,可以为网络里的中间盒的控制提供一个统一的框架,兼容已有的网络系统,采用逻辑型编程模型.同样,Merlin 也是以策略的定义作为输入,输出底层的指令,并同时允许管理员将权限分发给网络用户.

Merlin 同样由上层的高级语言和编译器组成,它允许网络管理员鉴别流量的类型、控制转发策略以及定义特殊的流量限制.同时,Merlin 的运行时系统(runtime system)还为动态改变网络协议提供了一个灵活的框架.

3.3.3 Assertion Language[32]

Assertion Language 是为了方便网络验证而提出的高级逻辑型语言.SDN 中虽然可以有全网的视图以及全网设备的行为记录,但是现有的 SDN 中的网络验证大部分都是基于静态的网络配置或是在数据平面进行分析验证,所以无法根据上层应用逻辑的动态变化而做出实时性的验证.在这种情况下,Assertion Language 便是能够对数据平面不断变化的网络做出验证的一种语言,它不仅能够做出可达性、是否存在循环以及切片是否分离这种一般性的验证,还可以利用本文提出的语言做出“有状态的防火墙”,“在 MAC 学习的过程中减少控制器的开销”这样的验证.

Assertion Language 可以针对变化的网络做出实时性的验证,采用数据平面的事件作为驱动的方式,利用正则表达式描述不同种类的报文的路径特征,采用逻辑型编程模型.但与其他语言不同的是,本文提出的语言不是为了抽象现有北向接口而提出来的,而是专门针对网络验证的高级接口.属于为了实现 SDN 中的某个新功能而做出的北向控制平面的抽象.

3.4 北向接口中的命令式语言

命令式语言由于更强调如何去实现一个过程,而不是强调实现什么过程,所以更加注重怎么做网络策略.前几种 SDN 北向接口的语言都重在抽象出一个接口,供编程者用它去获取自己需要的策略资源.而命令式语言更加关注的是让编程者去想如何实现这个算法策略.命令式语言的典型代表便是 Pyretic.

3.4.1 Pyretic[16]

Pyretic 也是对 SDN 北向接口语言的抽象,是 Frenetic 编程语言的 python 子类.Pyretic 可以让网络编程者和管理者写出模块化的网络应用,它基于 python,实时的编译系统可以将利用 pyretic 写出的程序转化为流表规则下发到交换机中,Pyretic 也是仅支持 OpenFlow 设备.

Pyretic 处理报文,对于编程者来说,屏蔽了底层 OpenFlow 规则被使用的细节,利用高级语言编写转换函数,把 packetin 报文转化为 packetout 报文,同时编译出相应的无冲突的流表规则.Pyretic 支持模块化编程,支持报文各个字段层次的或与非的操作,支持网络虚拟化,还可以利用事件的监听回调机制处理动态的策略等等.

3.5 北向接口中的面向对象编程语言

在 SDN 的北向接口的高级编程语言中,有一类语言专门为了实现网络虚拟化的功能,它们的编程模式更像面向对象的编程模型,即每一个对象都应该能够接受数据、处理数据并将数据传达给其他对象.这种编程模式的好处是可以将网络中的元素抽象为一个个对象,灵活且利于维护.SDN 中这类语言的典型代表有上文提到的 Pyretic,还有文献[33].

3.5.1 Splendid Isolation[33]

特定种类的流量彼此分离是很多网络操作得以正确进行的前提,例如高等院校要保护学生的记录,情报局的认证系统的流量要与常规流量分离,数据中心的租户之间的流量要分离等等.但是,在当今网络虚拟化的机制中,有很多机制都是临时的,比如 VLAN(虚拟局域网)虽然可以隔离处理网络中不同类型的报文,但加重了原本复杂的网络配置;防火墙虽然可以阻止特定类型的报文进入特定的网络切片中,但需要在控制层加入 hypervisor(例如 Flowvisor[41])并且需要可信.总之,没有一种方案能够提供一个完全满意的隔离机制,而且这些机制都不能提供一个验证网络是否被有效隔离的功能.

基于以上问题,文献[33]提出了一个可以允许多个网络程序并行运行的网络切片机制,同一个物理拓扑可以被多个应用程序所用,它可以实现流量隔离、物理隔离以及控制隔离.通过这种语言编写的程序易读,并且对用户友好,采用网络隔离的抽象接口,可以编写简单、灵活的程序,进而把物理网络分为多个虚拟切片,再在这个切片上编写应用程序.

3.5.2 Network Control Language(简称 NCL)[34]

NCL 是由华为公司提出的北向接口的高级语言,它基于 Java,编程者同样不用关心设备的实现细节,只关注应用的需求.由于已将 NCL 嵌入到 Java 中,所以用户可以直接用 Java 编写网络应用,NCL 中的编译器可以将策

略编译为数据平面的低级语言.利用 NCL 面向对象的语法模式可以完成如下功能:(1) 方便地定义自己的虚拟网络;(2) 时间监听与回调机制可以处理网络中的协议和通知;(3) 可以应对上层的拓扑查询等应用.

NCL 的系统结构如图 6 所示.

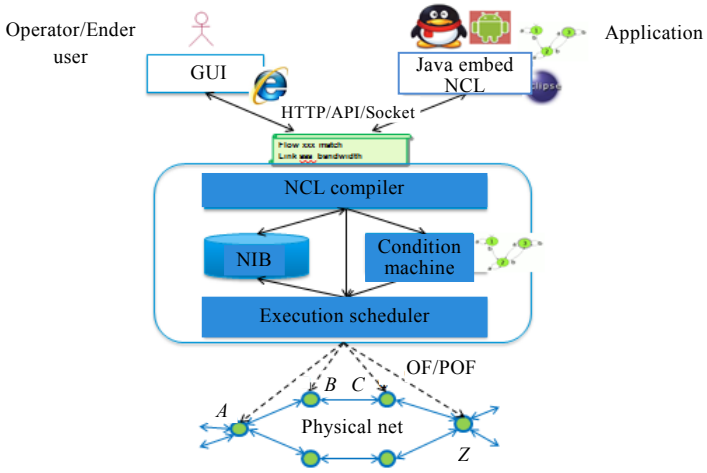

Fig.6 NCL architecture[34]

图 6 NCL 系统结构[34]

3.6 小 结

以上便是北向接口中的高级语言,总结北向接口中的各种编程语言,它们的共同点都是为了给网络编程带来简捷,通过抽象底层硬件设备细节,抽象流表生成及下发细节,抽象网络配置的细节,简化编程者的工作,使得网络管理员甚至是一般的网络参与者都能配置自己需要的网络策略,进而指导网络的行为.同时,某些北向接口的语言还可以提供网络验证、兼容各种网络中间盒设备、网络虚拟化等特性,使得 SDN 的可编程特性得到效益的最大化.下一节将对北向接口中的高级语言进行横向比较.

4 北向接口语言的横向比较

经过了上一节对于北向接口高级语言的分类和详细介绍,下面将对各个分类的语言进行一个横向的比较.

对于高级语言和低级语言而言,高级语言有如下优势:

(1) 屏蔽底层设备接口细节,方便编程者只关注网络策略的算法逻辑;

(2) 支持并行的应用同时运行,不用编程者手动解决流量冲突问题;

(3) 模块化的编程方便代码的复用;

(4) 物理网络可以被切分为隔离好的虚拟网络,方便路由等策略的部署.

而针对高级语言中的声明式语言、命令式语言以及面向对象语言,在不同的应用场景下分别有不同的优势.函数型语言中的 Frenetic,HFT,PANE,Maple,NetCore 更适合处理对于转发、防火墙等需要处理报文的应用;对于网络验证、编写有状态的防火墙的应用来说,Flog,Frenetic,FlowLog,Assertion Language 有明显的优势;对于网络虚拟化以及需要对底层流量做隔离的应用来说,面向对象的编程语言 Splendid Isolation,pyretic 有明显的优势.表 3 给出了在各个分类下的编程语言的详细比较.

由表 3 可以看出,从是否引入新的网络功能这个分类标准来看,对于需要进行网络验证、模型检查的网络,Flog、FlowLog、Merlin、Assertion Language 等都存在优势,而对于需要作网络切片、流量隔离的网络来说,Splendid Isolation 和 NCL 是比较不错的选择.同时,如果需要在网络上部署多个协议,可以采用 HFT、PANE、Maple、pyretic 等能够有效解决流表冲突的语言.

从语言的实现机制上,对于嵌入现有语言的北向接口语言来说,可以利用现有语言丰富的库以及复杂的功

能,对于编程者来说也能相对容易些.与此相对应,对于自定义语法规则的语言来说,更容易填写新的功能以及针对网络需求优化编译策略.

Table 3 Programming languages in each category

表 3 各个分类下的编程语言

编程语言	语言形式	编程模型	是否引入新功能	实现机制	主要面向的应用
NOX,POX 提供的北向 API	低级语言	命令式	直接利用设备提供的接口编程	内嵌于 Java 或 Python	符合相应 OpenFlow 版本的网络应用
tinyNBI	低级语言	命令式	兼容所有版本的 OpenFlow 协议	自定义语法规则	兼容现有 5 个版本 OpenFlow 的应用
FML	高级语言	交互式	简化协议部署过程	基于非递归的 DATALOG 语言	企业网络配置管理,包括 ACL、NAT、VLAN 等
Nettle	高级语言	交互式、函数型	网络事件的驱动转化为流驱动,可以处理动态协议	嵌入到 Haskell 语言中	支持 OpenFlow 交换机的网络.面向的应用未作明确限定,可支持动态路由、流量工程、安全策略、负载均衡等应用
Procera	高级语言	交互式、函数型	更好地处理网络中的交互行为	自定义语法规则	家庭网络和企业网络等的网络控制策略
Frenetic	高级语言	函数型	更加简化的报文转发模型,同时支持网络拓扑的改变以及网络验证等功能	扩展 OCaml 语法规则	未明确限定应用场景,可支持网络监测、路由、负载均衡等应用.可支持源码层面的网络验证等功能
HFT	高级语言	函数型、逻辑型	把策略转化为有优先级的流表,有效避免流表冲突	自定义语法规则	面向具有层次化策略的网络应用,能够解决策略冲突
PANE	高级语言	函数型、逻辑型	对 HFT 的扩展,可以给网络参与者分配权限,让他们参与网络配置	自定义语法规则	面向具有层次化策略的网络应用,能够解决策略冲突
Maple	高级语言	函数型	优化策略生成流表的过程并提供一个高效、多核的调度器	自定义语法规则	对具体的应用场景未作限定
NetCore	高级语言	函数型	改进 Frenetic 的编译算法,更高效地编译报文转发策略	基于 Frenetic	未明确限定应用场景,可支持网络监测、路由、负载均衡等应用
FlowLog	高级语言	函数型	更加方便地进行网络验证	自定义语法规则	面向模式检查、网络验证等应用
FatTire	高级语言	函数型	利用正则表达式提高网络的容错性	自定义语法规则	链路故障的发现以及容错处理等应用
NetKAT	高级语言	函数型	拥有完备的数学理论,可以进行网络验证	自定义语法规则	对具体的应用场景未作限定
Flog	高级语言	逻辑型	可进行模型检查、动态验证、编写有状态的中间盒	自定义语法规则	面向模型检查、动态验证等
Merlin	高级语言	逻辑型	支持将网络权限向下分发	自定义语法规则	对具体的应用场景未作限定,可鉴别流量类型
Assertion language	高级语言	逻辑型	在应用的级别作网络验证	基于 python	面向数据平面的网络验证
pyretic	高级语言	函数型、命令式	支持协议策略的并行化编写	基于 python	对具体的应用场景未作限定
Splendid isolation	高级语言	面向对象	提供网络切片功能	基于 python	主要面向网络虚拟化、网络隔离、网络切片等应用
NCL	高级语言	面向对象	支持网络切片、事件驱动以及网络资源请求功能	基于 Java	主要面向网络虚拟化、网络隔离、网络切片等应用

在语言的表达能力上,高级语言由于对细节的封装程度高,所以表达性要强于低级语言,更加易读,并且方便学习,所以对于用户也更加友好.在实现形式上,基本上所有的高级语言都由两部分组成:上层的高级语言以及下层的编译器,这一点基本类似.在运行效率上,编译器的效率决定了整体的运行效率,在高级语言中,NetCore 和 Maple 等在编译器上作了很多有效的优化,所以运行效率相对较高.

5 SDN 北向接口语言研究方向的展望

SDN 作为一种新型的网络体系结构,用可编程的交换机代替了传统网络中的交换机、路由器、防火墙等设备,通过集中的控制器去管理网络.SDN 的上层应用可以利用北向接口获取网络信息并做出网络策略指导下层设备的行为.这样的架构给网络的可编程性带来可能,但是基于 SDN 的编程却并不简单.针对现在 OpenFlow 给出的接口过于低级带来的各种问题,SDN 北向接口中的高级语言不断涌现,它们都在不同的应用层面上给网络编程者带来方便,克服了低级语言存在的种种问题.

但是目前针对 SDN 北向接口的研究工作还处于起步阶段,尚未发展成熟,仍有很多值得研究的问题,未来可进一步研究的相关重要方向包括:

(1) 北向接口还没有形成一个统一的标准.针对现在控制器多样化的情况,需要制定一套适用于定义北向接口的通用语义,这样不仅编程者有一个标准的部署应用的流程,开发控制器的人也可以利用这些已标准化的语义去开发各自的北向接口.

(2) 各个北向接口的编程语言都是针对特定场景而提出的,由于语言特性的限制,也只能固定简化某一类网络应用.如果能有一种较为通用的语言可以解决绝大多数场景下的问题,不拘束于语言的特性,则可以极大地提高编程效率.

(3) 对于一些常用的网络应用,北向接口语言可以提供一套常用功能函数库的支持,以提高 SDN 网络中的编程效率.

(4) 目前北向接口高级语言框架中编译器的性能还有待优化.例如,在提高编译生成流表的效率、尽量减少编译生成流表的优先级等方面,还有很大的提升空间.

(5) 现有的很多语言还仅仅支持单一控制器,如果能让这类语言自动部署运行在分布式控制器中,则可以有效提高控制器的性能以及整个网络的鲁棒性.

(6) 现有的北向接口语言往往只能支持有限的新功能,如何将多种新功能集成到同一个北向接口语言中,有待进一步加以研究.

(7) 对于可以处理网络链路错误配置的北向接口语言(如 FatTire 等)来说,如何能够发现交换机层级的错误以及如何让链路进行自动恢复也是未来的研究工作.

(8) 对于支持网络隔离和虚拟化的北向接口语言(如 Splendid Isolation),现在只限于在拓扑上进行网络切片,未来可以考虑针对网络带宽等更多的网络资源进行切片.

6 总 结

本文是一篇针对软件定义网络中北向接口语言的综述,为了简化软件定义网络中的编程模型,给网络策略的开发和部署提供便捷,近年来,北向接口的语言纷纷涌现出来,在这些语言中,根据不同的分类标准,又可以划分为不同的层次,本文详细介绍了学术界和工业界现有的各类北向接口的编程语言,同时对各种语言进行了横向比较,并对未来北向接口的研究工作进行了展望.

References:

[1] ONF Market Education Committee. Software-Defined Networking: The New Norm for Networks. ONF White Paper. Palo Alto: Open Networking Foundation, 2012.

[2] Nunes BAA, Mendonca M, Nguyen XN, Obraczka K, Turletti T. A survey of software-defined networking: Past, present, and future of programmable networks. IEEE Communication Surveys & Tutorials, 2014,16(3):1617−1634. [doi: 10.1109/SURV.2014.012214.00180]

[3] Kreutz D, Ramos FMV, Verissimo PE, Rothenberg CE, Azodolmolky S, Uhlig S. Software-Defined networking: A comprehensive survey. Proc. of the IEEE, 2015,103(1):14−76. [doi: 10.1109/JPROC.2014.2371999]

[4] Hu F, Hao Q, Bao K. A survey on software defined networking (SDN) and OpenFlow: From concept to implementation communications surveys & tutorials. IEEE Communication Surveys & Tutorials, 2014,16(4):2181−2606. [doi: 10.1109/COMST.2014.2326417]

[5] Zuo QY, Chen M, Zhao GS, Xing CY, Zhang GM, Jiang PC. Research on OpenFlow-based SDN technologies. Ruan Jian Xue Bao/Journal of Software, 2013,24(5):1078−1097 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4390.htm [doi: 10.3724/SP.J.1001.2013.04390]

[6] Gude N, Koponen T, Pettit J, Pfaff B, Casado M, McKeown N, Shenker S. NOX: Towards an operating system for networks. Computer Communication Review, 2008,38(3):105−110. [doi: 10.1145/1384609.1384625]

[7] McCauley M. POX, 2012.

[8] Floodlight Is A Java-Based OpenFlow Controller, 2012.

[9] https://openflow.stanford.edu/display/Beacon/Home

[10] http://www.opendaylight.org/

[11] http://www.juniper.net/us/en/dm/sdn/

[12] Mckeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L, Rexford J, Shenker S, Turner J. OpenFlow: Enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 2008,38(2):69−74. [doi: 10.1145/1355734.1355746]

[13] Bosshart P, Dan D, Gibb G, Martin I, McKeown N, Rexford J, Schlesinger C, Talayco D, Vahdat A, Varghese G, Walker D. P4: Programming protocol-independent packet processors. Computer Communication Review, 2014,44(3):87−95. [doi: 10.1145/2656877.2656890]

[14] Song H. Protocol-Oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane. In: Proc. of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, ser. HotSDN 2013. Hong Kong: ACM, 2013. 127−132. [doi: 10.1145/2491185.2491190]

[15] Casey CJ, Sutton A, Sprintson A. tinyNBI: Distilling an API from essential OpenFlow abstractions. In: Proc. of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, ser. HotSDN 2014. Chicago: ACM, 2014. 37−42. [doi: 10.1145/2620728.2620757]

[16] Monsanto C, Reich J, Foster N, Rexford J, Walker D. Composing software-defined networks. In: Proc. of the 10th USENIX Conf. on Networked Systems Design and Implementation, ser. NSDI 2013. Berkeley: USENIX Association, 2013. 1−14.

[17] Laird A. The four major programming paradigms topic paper #17. Computer Science, 2009. http://www.alexlaird.com/content/uploads/2009/05/topicpaper17-thefourmajorprogrammingparadigms.pdf

[18] Coenen F. Characteristics of declarative programming languages. 1999. http://cgi.csc.liv.ac.uk/~frans/OldLectures/2CS24/declarative.html

[19] Voellmy A, Hudak P. Nettle: Taking the sting out of programming network routers. In: Proc. of the PADL 2011. Berlin, Heidelberg: Springer-Verlag, 2011. 235−249. [doi: 10.1007/978-3-642-18378-2_19]

[20] Hinrichs TL, Gude NS, Casado M, Mitchell JC, Shenker S. Practical declarative network management. In: Proc. of the WREN 2009. Barcelonan: ACM, 2009. 1−10. [doi: 10.1145/1592681.1592683]

[21] Voellmy A, Kim H, Feamster N. Procera: A language for high-level reactive network control. In: Proc. of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, ser. HotSDN 2012. Helsinki: ACM, 2012. 43−48. [doi: 10.1145/2342441.2342451]

[22] Reitblatt M, Canini M, Guha A, Foster N. Fattire: Declarative fault tolerance for software defined networks. In: Proc. of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, ser. HotSDN 2013. Hong Kong: ACM, 2013. 109−114. [doi: 10.1145/2491185.2491187]

[23] Nelson T, Guha A, Dougherty DJ, Fisler K, Krishnamurthi S. A balance of power: Expressive, analyzable controller programming. In: Proc. of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, ser. HotSDN 2013. Hong Kong: ACM, 2013. 79−84. [doi: 10.1145/2491185.2491201]

[24] Foster N, Harrison R, Freedman MJ, Monsanto C, Rexford J, Story A, Walker D. Frenetic: A network programming language. In: Proc. of the ICPF 2011. Tokyo: ACM, 2011. 279−291. [doi: 10.1145/2034773.2034812]

[25] Foster N, Guha A, Reitblatt M, Story A, Freedman MJ, Katta N.P, Monsanto C, Reich J, Rexford J, Schlesinger C, Walker D, Harrison MR. Languages for software-defined networks. IEEE Communications Magazine, 2013,51(2):128−134. [doi: 10.1109/MCOM.2013.6461197]

[26] Ferguson AD, Guha A, Liang C, Fonseca R, Krishnamurthi S. Hierarchical policies for software defined networks. In: Proc. of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, ser. HotSDN 2012. Helsinki: ACM, 2012. 37−42. [doi: 10.1145/2342441.2342450]

[27] Voellmy A, Wang J, Yang YR, Ford B, Hudak P. Maple: Simplifying SDN programming using algorithmic policies. In: Proc. of the SIGCOMM 2013. Hong Kong: ACM, 2013. 87−98. [doi: 10.1145/2486001.2486030]

[28] Monsanto C, Foster N, Harrison R, Walker D. A compiler and run-time system for network programming languages. In: Proc. of the POPL 2012. Philadelphia: ACM, 2012,47(1):217−230. [doi: 10.1145/2103656.2103685]

[29] Guha A, Reitblatt M, Foster N. Machine-Verified network controllers. In: Proc. of the PLDI 2013. Seattle: ACM, 2013. 483−494. [doi: 10.1145/2491956.2462178]

[30] Katta NP, Rexford J, Walker D. Logic programming for software-defined networks. In: Proc. of the ACM SIGPLAN Workshop on Cross-Model Language Design and Implementation, ser. XLDI. 2012. https://www.cs.princeton.edu/~dpw/papers/xldi-2012.pdf

[31] Soule R, Basu S, Kleinberg R, Sirer EG, Foster N. Managing the network with Merlin. In: Proc. of the 12th ACM Workshop on Hot Topics in Networks (HotNets-XII). College Park, 2013. 1−7. [doi: 10.1145/2535771.2535792]

[32] Beckett R, Zou XK, Zhang SY, Malik S, Rexford J, Walker D. An assertion language for debugging SDN applications. In: Proc. of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, ser. HotSDN 2014. Chicago: ACM, 2014. 91−96. [doi: 10.1145/2620728.2620743]

[33] Gutz S, Story A, Schlesinger C, Foster N. Splendid isolation: A slice abstraction for software-defined networks. In: Proc. of the 1st Workshop on Hot Topics in Software Defined Networks, ser. HotSDN 2012. Helsinki: ACM, 2012. 79−84. [doi: 10.1145/2342441.2342458]

[34] Network Control Language(NCL) Software Defined Networking Research Group. 2014. https://datatracker.ietf.org/meeting/90/agenda/sdnrg/

[35] Anderson CJ, Foster N, Guha A, Jeannin JB, Kozen D, Schlesinger C, Walker D. NetKAT: Semantic foundations for networks. In: Proc. of the POPL 2014. San Diego: ACM, 2014. 113−126. [doi: 10.1145/2535838.2535862]

[36] Ferguson AD, Guha A, Liang C, Fonseca R, Krishnamurthi S. Participatory networking: An API for application control of SDNs. In: Proc. of the SIGCOMM 2013. Hong Kong: ACM, 2013. 327−338. [doi: 10.1145/2486001.2486003]

[37] Pfenning F. Logic programming lecture 26 datalog. 2006. http://www.cs.cmu.edu/~fp/courses/lp/

[38] Oliveira N, Pereira MJV, Henriques PR, Cruz D. Domain specific languages: A theoretical survey. In: Proc. of the 3rd Compilers, Programming Languages, Related Technologies and Applications (CoRTA 2009). 2009. http://alfa.di.uminho.pt/~danieladacruz/CoRTA09DSLsurveyvf.pdf

[39] Bird R. Introduction to Functional Programming Using Haskell. New York: Prentice Hall, 1998.

[40] Kazemian P, Varghese G, McKeown N. Header space analysis: Static checking for networks. In: Proc. of the 9th USENIX Conf. on Networked Systems Design and Implementation, ser. NSDI 2012. Lombard: USENIX Association, 2012. 9. https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/kazemian

[41] Sherwood R, Gibb G, Yap KK, Appenzeller G, Casado M, McKeown N, Parulkar G. FlowVisor: A network virtualization layer. Technical Report, OPENFLOW-TR-2009-01, OpenFlow Consortium, 2009. 1−14.

附中文参考文献:

[5] 左青云,陈鸣,赵广松,邢长友,张国敏,蒋培成.基于 OpenFlow 的 SDN 技术.软件学报,2013,24(5):1078−1097. http://www.jos.org.cn/1000-9825/4390.htm [doi: 10.3724/SP.J.1001.2013.04390]

于洋(1989－),女,吉林松原人,硕士生,主要研究领域为软件定义网络.

施新刚(1980－),男,博士,高级工程师,主要研究领域为网络测量,互联网体系结构与协议,互联网路由.

王之梁(1978－),男,博士,副研究员,主要研究领域为网络协议测试与形式化方法,互联网体系结构与协议,软件定义网络.

尹霞(1972－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为互联网体系结构与协议,网络协议测试与形式化方法,互联网路由.

毕军(1972－),男,博士,研究员,博士生导师,CCF 杰出会员,主要研究领域为互联网体系结构与协议,未来互联网,软件定义网络,互联网路由.