软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(1):62−81 [doi: 10.13328/j.cnki.jos.004701]

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

软件定义网络(SDN)研究进展*

张朝昆, 崔 勇, 唐骞祎, 吴建平

(清华大学 计算机科学与技术系,北京 100084)

通讯作者: 崔勇, E-mail: cuiyong@tsinghua.edu.cn

摘 要: 网络抽象促使软件定义网络(software-defined networking,简称 SDN)的产生.SDN 将数据平面与控制平面解耦合,简化了网络管理.首先从 SDN 诞生发展的背景入手,梳理了 SDN 的体系结构,包括数据层、控制层和应用层,并按照 SDN 的层次结构深入阐述其关键技术,特别分析了一致性、可用性和容错性等特性.然后,论述了 SDN 在不同应用场景下的最新研究成果.最后,展望未来研究工作.

关键词: 软件定义网络;接口;交换机;控制器;网络抽象

中图法分类号: TP393

中文引用格式: 张朝昆,崔勇,唐骞祎,吴建平.软件定义网络(SDN)研究进展.软件学报,2015,26(1):62−81. http://www.jos.org.cn/1000-9825/4701.htm

英文引用格式: Zhang CK, Cui Y, Tang HY, Wu JP. State-of-the-Art survey on software-defined networking (SDN). Ruan Jian Xue Bao/Journal of Software, 2015,26(1):62−81 (in Chinese). http://www.jos.org.cn/1000-9825/4701.htm

State-of-the-Art Survey on Software-Defined Networking (SDN)

ZHANG Chao-Kun, CUI Yong, TANG He-Yi, WU Jian-Ping

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

Abstract: Network abstraction brings about the naissance of software-defined networking. SDN decouples data plane and control plane, and simplifies network management. The paper starts with a discussion on the background in the naissance and developments of SDN, combing its architecture that includes data layer, control layer and application layer. Then their key technologies are elaborated according to the hierarchical architecture of SDN. The characteristics of consistency, availability, and tolerance are especially analyzed. Moreover, latest achievements for profiled scenes are introduced. The future works are summarized in the end.

Key words: SDN (software-defined networking); interface; switch; controller; network abstraction

传统网络的层次结构是互联网取得巨大成功的关键.但是随着网络规模的不断扩大,封闭的网络设备内置了过多的复杂协议,增加了运营商定制优化网络的难度,科研人员无法在真实环境中规模部署新协议.同时,互联网流量的快速增长(预计到 2018 年,全球流量将达到 1.6×10^{21} 字节[1]),用户对流量的需求不断扩大,各种新型服务不断出现,增加了网络运维成本.

SDN 起源于 2006 年斯坦福大学的 Clean Slate 研究课题[2].2009 年,Mckeown 教授正式提出了 SDN 概念[3].利用分层的思想,SDN 将数据与控制相分离.在控制层,包括具有逻辑中心化和可编程的控制器,可掌握全局网络信息,方便运营商和科研人员管理配置网络和部署新协议等.在数据层,包括哑的(dumb)交换机(与传统的二层交换机不同,专指用于转发数据的设备).交换机仅提供简单的数据转发功能,可以快速处理匹配的数据包,适应流量日益增长的需求.两层之间采用开放的统一接口(如 OpenFlow[4]等)进行交互.控制器通过标准接口向交

* 基金项目: 国家自然科学基金(61120106008, 61161140454); 国家高技术研究发展计划(863)(2013AA010401); 国家下一代互联网示范工程(CNGI-12-03-003)

收稿时间: 2014-02-28; 修改时间: 2014-04-19; 定稿时间: 2014-07-19; jos 在线出版时间: 2014-08-19

CNKI 网络优先出版: 2014-08-19 14:16, http://www.cnki.net/kcms/doi/10.13328/j.cnki.jos.004701.html

换机下发统一标准规则,交换机仅需按照这些规则执行相应的动作即可.因此,SDN 技术能够有效降低设备负载,协助网络运营商更好地控制基础设施,降低整体运营成本,成为最具前途的网络技术之一.因此,SDN 被 MIT 列为“改变世界的十大创新技术之一”[5].SDN 相关技术研究迅速开展起来,成为近年来的研究热点.2013 年,SIGCOMM 会议收录了多篇相关文章,甚至将 SDN 列为专题来研讨,带动了 SDN 相关研究的蓬勃发展.

本文第 1 节首先论述 SDN 诞生与发展的背景,以此提出 SDN 分层的体系结构,并讨论 SDN 开放式接口.第 2 节和第 3 节分别对 SDN 数据层和控制层的关键技术进行详细阐述.第 4 节论述 SDN 在不同场景下的应用.第 5 节针对未来工作提出见解.最后给出本文结论.

1 SDN 体系结构

SDN 是当前最热门的网络技术之一,它解放了手工操作,减少了配置错误,易于统一快速部署.本节首先论述了 SDN 诞生发展的背景,指出 SDN 的产生及快速发展的必然性定律.接下来探讨了 SDN 主流的体系结构.由于 SDN 标准接口机制确保层次之间既保持相对独立,又能正常通信,因此,标准接口设计的好坏是 SDN 成功设计的关键.

1.1 SDN诞生发展的背景

随着网络的快速发展,传统互联网出现了如传统网络配置复杂度高等诸多问题[6],这些问题说明网络架构需要革新,可编程网络的相关研究为 SDN 的产生提供了可参考的理论依据[7].主动网络[8,9]允许数据包携带用户程序,并能够由网络设备自动执行.用户可以通过编程方式动态地配置网络,达到了方便管理网络的目的.然而由于需求低、协议兼容性差等问题,并未在工业界实际部署.4D 架构[10,11]将可编程的决策平面(即控制层)从数据平面分离,使控制平面逻辑中心化与自动化,其设计思想产生 SDN 控制器的雏形[12].

借鉴计算机系统的抽象结构,未来的网络结构将存在转发抽象、分布状态抽象和配置抽象这 3 类虚拟化概念[13].转发抽象剥离了传统交换机的控制功能,将控制功能交由控制层来完成,并在数据层和控制层之间提供了标准接口,确保交换机完成识别转发数据的任务.控制层需要将设备的分布状态抽象成全网视图,以便众多应用能够通过全网信息进行网络的统一配置.配置抽象进一步简化了网络模型,用户仅需通过控制层提供的应用接口对网络进行简单配置,就可自动完成沿路径转发设备的统一部署.因此,网络抽象思想解耦了路径依赖,成为数据控制分离且接口统一架构(即 SDN)产生的决定因素.

此外,众多标准化组织已经加入到SDN相关标准的制订当中.专门负责订制SDN接口标准的著名组织是开放网络基金会(Open Networking Foundation,简称 ONF)[14],该组织制订的 OpenFlow 协议业已成为 SDN 接口的主流标准,许多运营商和生产厂商根据该标准进行研发.互联网工程任务组(Internet Engineering Task Force,简称 IETF)的 ForCES 工作组[15]、互联网研究专门工作组(Internet Research Task Force,简称 IRTF)的 SDNRG 研究组[16]以及国际电信联盟远程通信标准化组织(ITU Telecommunication Standardization Sector,简称 ITU-T)的多个工作组[17]同样针对 SDN 的新方法和新应用等展开研究.标准化组织的跟进,促使了 SDN 市场的快速发展.据悉,SDN 市场已于 2013 年达到约 2 亿美元的产值,预计到 2016 年将达到 20 亿美元[18],市场需求确保 SDN 有足够的发展空间.由此可见,SDN 具有广阔的发展前景和巨大的研究价值.

1.2 体系结构概述

针对不同的需求,许多组织提出了相应的 SDN 参考架构.SDN 架构[19]最先由 ONF 组织提出,并已经成为学术界和产业界普遍认可的架构.除此之外,欧洲电信标准化组织(European Telecommunications Standards Institute,简称 ETSI)提出的 NFV 架构[20]随之发展起来,该体系结构主要针对运营商网络,并得到了业界的支持.由各大设备厂商和软件公司共同提出了 OpenDaylight[21],目的是为了具体实现 SDN 架构,以便用于实际部署.

ONF 组织最初在白皮书中提到 SDN 体系结构[19],并于 2013 年底发布最新版本[22],其架构如图 1 所示.SDN 由下到上(或称由南向北)分为数据平面、控制平面和应用平面.数据平面与控制平面之间利用 SDN 控制数据平面接口(control-data-plane interface,简称 CDPI)进行通信,CDPI 具有统一的通信标准,目前主要采用 OpenFlow 协

议[4].控制平面与应用平面之间由 SDN 北向接口(northbound interface,简称 NBI)负责通信,NBI 允许用户按实际需求定制开发.

数据平面由交换机等网络元素组成,各网络元素之间由不同规则形成的 SDN 网络数据通路形成连接.控制平面包含逻辑中心的控制器,负责运行控制逻辑策略,维护着全网视图.控制器将全网视图抽象成网络服务,通过访问 CDPI 代理来调用相应的网络数据通路,并为运营商、科研人员及第三方等提供易用的 NBI,方便这些人员订制私有化应用,实现对网络的逻辑管理.应用平面包含着各类基于 SDN 的网络应用,用户无需关心底层设备的技术细节,仅通过简单的编程就能实现新应用的快速部署.CDPI 负责将转发规则从网络操作系统发送到网络设备,它要求能够匹配不同厂商和型号的设备,而并不影响控制层及以上的逻辑.NBI 允许第三方开发个人网络管理软件和应用,为管理人员提供更多的选择.网络抽象特性允许用户可以根据需求选择不同的网络操作系统,而并不影响物理设备的正常运行.

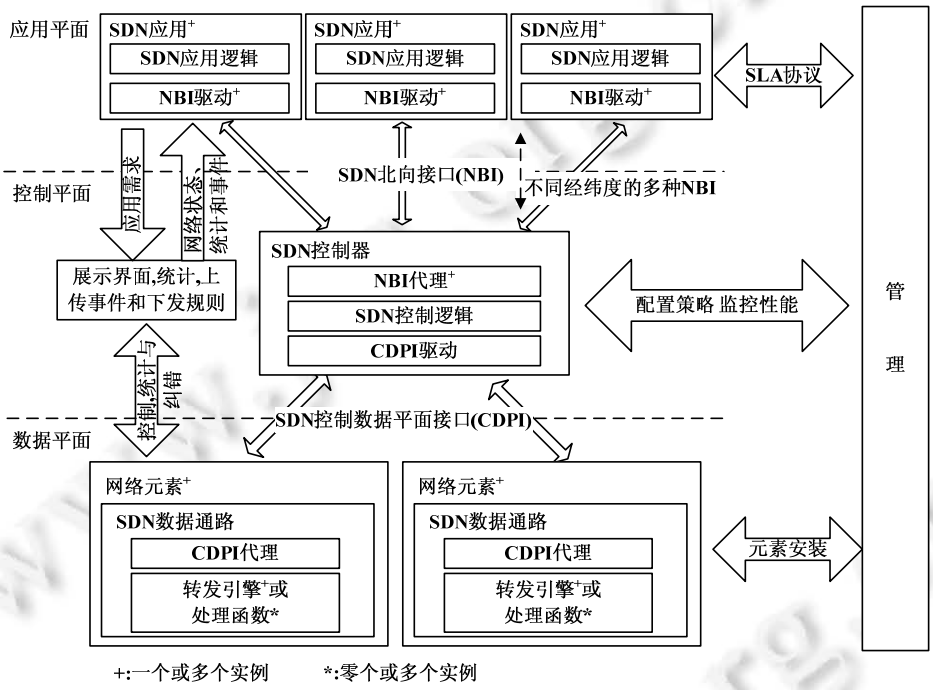

Fig.1 SDN architecture

图 1 SDN 体系结构

NFV[20]是针对运营商网络出现的问题而提出的 SDN 解决方案.网络运营商的网络由专属设备来部署,随着各种新型网络服务的产生,这些专属设备功能变得繁杂,而管理这些繁杂的硬件设备造成运营成本及能耗的增加,从而导致运营商网络的发展遇到瓶颈.针对上述问题,NFV 将传统网络设备的软件与硬件相分离,使网络功能更新独立于硬件设备.为此,NFV 采用了资源虚拟化的方式,在硬件设备中建立一个网络虚拟层,负责将硬件资源虚拟化,形成虚拟计算资源、虚拟存储资源和虚拟网络资源等,运营商通过软件来管理这些虚拟资源.由于采用的是通用硬件设备,NFV 降低了设备成本,减少了能耗,缩短了新网络服务的部署周期,从而适应网络运营商的发展需求.在接口设计方面,NFV 既可以基于非 OpenFlow 协议,又能与 OpenFlow 协同工作,同时还支持 ForCES[16]等多种传统接口标准化协议,以便适应网络运营商对设备的不同需求,并与 ONF 的 SDN 保持相对独立的发展.

OpenDaylight[21]的目标是通过 SDN 的开源开发,推进业界可部署方案具体实施,其架构由设备厂商提出并得到众多 IT 软件厂商的支持.考虑到兼容性问题,OpenDaylight 继承了 SDN 架构形式,同时又结合了 NFV 的特

点.架构共分为 3 个层次,分别是网络应用与业务流程(即应用层)、控制平台(即控制层)和物理与虚拟网络设备(即数据层).OpenDaylight 的控制平台直接由自带的 Java 虚拟机实现.针对不同的网络任务,控制器自身携带了一系列可插入模块,并兼容第三方模块以增强 SDN 的功能.与 ONF 的 SDN 架构最大的不同在于:OpenDaylight 控制器的南向接口除了支持 OpenFlow 协议之外,还支持 NETCONF[23]等配置协议和 BGP[24]等路由协议,并支持生产厂商的专有协议(如思科的 OnePK 协议[25]).为了能够处理不同的标准协议,OpenDaylight 增加了服务抽象层 SAL,它负责将不同的底层协议标准转换成 OpenDaylight 控制层所理解的请求服务,保持了底层协议的透明性,并提高了整体架构的可扩展性.

SDN,NFV 和 OpenDaylight 的对比见表 1.由于 NFV 与 ONF 的 SDN 分别负责不同的网络,两种架构的协同工作能够获得更好的网络体验,将两者结合可以降低设备成本.通过利用通用交换机等设备和软件代替原有设备,使得设备的升级与网络应用的拓展相对独立.OpenDaylight 具有开源性,因此,它可以兼容 SDN,NFV 以及未来与 SDN 并行的体系结构.总之,无论是哪个组织提出的 SDN 体系结构,实现的目标是一致的.SDN 使得数据控制相分离的网络具有开放性和可编程性,科研人员及运营商可以通过 PC 机、手机、Web 网页或未来可能出现的各种途径进行网络部署,而部署工作也仅是应用软件的简单开发或配置.可以预见:针对 SDN 并行架构的研究,是未来研究进展的重要趋势之一.

Table 1 Comparison of SDN, NFV and OpenDaylight

表 1 SDN,NFV 和 OpenDaylight 的对比

体系结构	接口标准	与 SDN 兼容性	相关背景介绍			特点
			应用领域	发起人	组织者	
ONF SDN[22]	OpenFlow	–	校园网,企业网,数据中心	高校科研人员	ONF	强调控制与数据分离等
NFV[20]	多种接口协议,支持可扩展	可协同工作	运营商网络	运营商	ETSI NFV 工作组	强调网络功能虚拟化
OpenDaylight[21]	多种接口协议,支持可扩展	完全支持 OpenFlow	未来的互联网	设备生产厂商和 IT 软件厂商	OpenDaylight (Linux 基金)	强调软件开源及实现

1.3 开放式接口与协议设计

SDN 中的接口具有开放性,以控制器为逻辑中心,南向负责与数据层通信,北向负责与应用层通信.此外,由于单一控制机制容易造成控制节点失效,严重影响性能,可采用多控制器方式[26],此时,多控制器之间将采用东西向通信方式.开放式接口的研究,必然进一步推动 SDN 的深入发展.

在这些开放式接口研究中,控制器南向接口作为数据与控制分离的核心被广泛研究,成为业界关注的焦点.由于控制层与数据层解耦合,使得针对这两层的改进相对独立,在层与层之间仅需提供标准南向接口即可.南向接口是 SDN 分层架构的关键元素,然而逻辑上,它既要保证数据层与控制层之间的正常通信,又要支持两层独立更新;物理上,设备生产厂商需要开发支持这种标准接口的设备,因为传统网络设备是不能在 SDN 网络之中运行的.因此,研发南向标准接口成为 SDN 基础研究中的重要内容之一.

许多组织着手制订南向标准接口.ONF 提出的 CDPI 成为了主流南向接口,它采用 OpenFlow[4]协议.OpenFlow 是 SDN 中第一个广泛使用的数据控制层接口协议,得到学术界普遍关注[27−29],它将单一集成和封闭的网络设备成为灵活可控的通信设备.OpenFlow 协议是基于流的概念来匹配规则的,因此,交换机需要维护一个流表(flow table)来支持 OpenFlow,并按流表进行数据转发,而流表的建立、维护及下发均由控制器来完成.

为了便于设备生产厂商开发支持 OpenFlow 的设备,ONF 开始提供 OpenFlow 协议标准[30].OpenFlow 1.0.0 规定流表头为 12 元组(如源/目的 IP 地址、源/目的 MAC 地址等),在一定程度上满足了用户对 SDN 网络的需求.然而,1.0.0 版本还不完善,如支持的规则和动作过少、仅支持单表、无关联动作的组合容易造成组合爆炸等问题.因此,OpenFlow 1.1.0 增加了部分规则,并开始支持多级流表、群组表和动作集等功能.IPv6 是下一代互联网的核心元素,因此从 1.2 版本开始,OpenFlow 增加了对 IPv6 源/目的地址的支持.网络拥塞一直是传统互联网需要面临的问题之一,在 SDN 网络中也是如此,因此,从 1.3 版本开始支持流控机制.在 1.4.0 版本中,OpenFlow

协议增加了流表删除和复制机制,并考虑了流表一致性问题.总的来说,OpenFlow 支持的功能越来越全,机制也在不断地更新完善.然而,随着 OpenFlow 支持的功能不断增加,流表将容易产生负载过重的问题.如何支持不同粒度、任意组合的功能,是 OpenFlow 下一步发展的关键所在.此外,OpenFlow 允许控制器利用流表指定网络的数据通路,但并未指定如何配置和管理转发设备环境,因此,ONF 提出了 OF-CONFIG 协议[31].作为配置协议,OF-CONFIG 扩展了 NETCONF 标准[23],采用 XML 配置交换机环境,填补了 OpenFlow 在配置方面的缺失.

针对南向接口,除了 ONF 提出的 OpenFlow 协议和 OF-CONFIG 协议外,IETF 提出了 ForCES 协议[16],思科公司提出了 OnePK 协议[25].ForCES 对网络设备内部结构重新定义,将转发元素(forwarding element,简称 FE)和控制元素(control element,简称 CE)分离,形成两个独立的逻辑实体,两个逻辑实体之间依靠 ForCES 协议通信.该协议工作在主从模式下,即,CE 通过 ForCES 协议主动将指令下发给 FE,FE 被动接受这些指令,并通过硬件执行每数据包级的分组转发.OnePK 则是思科公司针对 SDN 产品专门开发的接口协议,该协议可以运行在思科所研发的专属平台上,并支持开发者用 C 或 Java 编写的程序.OnePK 除了支持专有的 OnePK 协议之外,还可支持 OpenFlow 协议等.典型的南向接口协议对比见表 2.

Table 2 Comparison of general south interface protocols

表 2 典型的南向接口协议的对比

南向接口	协议类型	粒度	通信方式	与 OpenFlow 兼容	是否改变传统网络架构	组织	特点
OpenFlow[30]	流表	每流	双向通信	–	是	ONF	根据流表指定的数据通路传输数据
OF-CONFIG[31]	XML	无关	双向通信	是	是	ONF	配置交换机环境
ForCES[16]	传统报文	每分组	主从通信	否	否	IETF	通过传统 IP 网络传输数据
OnePK[25]	未提及	每流	双向通信	是	是	Cisco	专用接口

除了南向接口相关研究之外,控制器北向接口及控制器间东西向接口同样是研究重点.北向接口负责控制层与各种业务应用之间的通信,应用层各项业务通过编程方式调用所需网络抽象资源,掌握全网信息,方便用户对网络配置和应用部署等业务的快速推进.然而,由于应用业务具有多样性,使得北向接口亦呈现多样性,开发难度较大.起初,SDN 允许用户针对不同应用场景定制适合的北向接口标准.统一的北向接口标准将直接影响着各项应用业务的顺利开展.为了统一北向接口,各组织开始制订北向接口标准,如 ONF 的 NBI 接口标准和 OpenDaylight 的 REST 接口标准等.然而,这些标准仅对功能作了描述,而未详细说明实现方式.因此,如何实现统一的北向接口标准,成为业界下一步主要推动的工作.与南北向接口通信的方式不同,东西向接口负责控制器间的通信.由于单一控制器性能有限,无法满足大规模 SDN 网络部署,东西向接口标准的制订使控制器具有可扩展能力,并为负载均衡和性能提升等方面提供了技术保障.

2 数据层关键技术研究

在 SDN 中,数据层与控制层分离,交换机将繁重的控制策略部分交由控制器来负责,而它仅根据控制器下发的规则对数据包进行快速转发.为了避免交换机与控制器频繁交互,双方约定的规则是基于流的,而并非基于每个数据包的.SDN 数据层的功能相对简单,相关技术研究主要集中在交换机和转发规则方面:首先是交换机设计研究,即设计可扩展的快速转发设备,它既可以灵活匹配规则,又能快速转发数据流;其次是转发规则的相关研究,例如规则失效后的一致性更新问题等.下面详细讨论数据层相关的研究成果.

2.1 交换机设计问题

SDN 交换机位于数据层面,用来负责数据流的转发.通常可采用硬件和软件两种方式进行转发.对于硬件来说,具有速度快、成本低和功耗小等优点.一般来说,交换机芯片的处理速度比 CPU 处理速度快两个数量级,比网络处理器(network processor,简称 NP)快一个数量级,并且这种差异将持续很长时间[32].在灵活性方面,硬件则远远低于 CPU 和 NP 等可编程器件.如何设计交换机,做到既保证硬件的转发速率,同时还能确保识别转发规则的

灵活性,成为目前研究的热点问题.

利用硬件处理数据,可以保证转发效率,但亟需解决处理规则不够灵活的问题.为了使硬件能够灵活解决数据层的转发规则匹配严格和动作集元素数量太少等限制性问题,Bosshart 等人[32]针对数据平面转发提出了RMT 模型.该模型实现了一个可重新配置的匹配表,它允许在流水线(pipeline)阶段支持任意宽度和深度的流表.重新配置数据层涉及 4 个方面:① 允许随意替换或增加域定义;② 允许指定流表的数量、拓扑、宽度和深度,仅仅受限于芯片的整体资源(如芯片内存大小等);③ 允许创建新动作;④ 可以随意将数据包放到不同的队列中,并指定发送端口.RMT 模型如图 2 所示.从结构上看,理想的 RMT 模型是由解析器、多个逻辑匹配部件以及可配置输出队列组成.具体的可配置性体现在:通过修改解析器来增加域定义,修改逻辑匹配部件的匹配表来完成新域的匹配,修改逻辑匹配部件的动作集来实现新的动作,修改队列规则来产生新的队列.这样容易模拟网关、路由器和防火墙等设备,也能够使用多标记头或非标准的协议.所有更新操作是通过解析器来实现的,无需修改硬件,只需在芯片设计时留出可配置的接口即可,实现了硬件对数据的灵活处理.

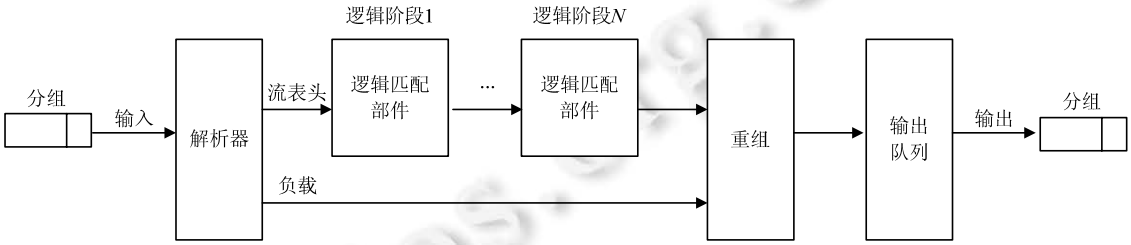

Fig.2 RMT model

图 2 RMT 模型

作为另一种利用硬件灵活处理技术,FlowAdapter[33]采用交换机分层的方式来实现高效、灵活的多表流水线业务.FlowAdapter 交换机分为 3 层:最上层是可以通过更新来支持任何新协议的软件数据平面,底层是相对固定但转发效率高的硬件数据平面,位于中部的 FlowAdapter 层负责软件数据平面和硬件数据平面之间的通信.当控制器下发规则时,软件数据平面将存储这些规则,形成 M 个阶段的流表.由于这些规则相对灵活,不能全部由交换机直接转化成相应转发动作,而硬件数据平面可以实现规则的高速匹配转发.因此可利用中间层 FlowAdapter 将两个数据平面中的规则进行无缝转换,即,将相对灵活的 M 阶段的流表转换成能够被硬件所识别的 N 阶段的流表.为了达到转换目的,FlowAdapter 首先检查软件数据平面的全部规则,然后根据完整的规则将 M 阶段的流表转换成 1 阶段流表,最后再将 1 阶段流表转换成 N 阶段流表发送给硬件数据平面.通过这种无缝转换,理论上解决了传统交换机硬件与控制器之间多表流水线技术不兼容的问题.另外,FlowAdapter 相对控制器完全透明,对 FlowAdapter 交换机的更新不会影响控制器的正常运行.

与利用硬件设计交换机的观点不同,虽然软件处理的速度低于硬件,但是软件方式可以最大限度地提升规则处理的灵活性,同时又能避免由于硬件自身内存较小、流表大小受限[34]、无法有效处理突发流等问题,因而同样受到学术界的关注.利用交换机 CPU[35]处理转发规则,可以避免硬件灵活性差的问题.由于 CPU 处理数据包的能力变得越来越强[36],商用交换机很自然地也会采用这种更强的 CPU.这样,在软件处理转发速度与硬件差别变小的同时,灵活处理转发规则的能力得到提升.进一步地,还可以采用 NP 的处理方式[37].由于 NP 专门用来处理网络的各种任务,如数据包转发、路由查找和协议分析等,因此在网络处理方面,NP 比 CPU 具有更高效的处理能力.无论采用 CPU 还是 NP,应发挥处理方式灵活性的优势,同时尽量避免处理效率低而带来的影响.

此外,在数据平面中,哪些元素可以交给硬件处理,哪些元素可以交给软件处理,也是值得考虑的问题.例如,原本设计到硬件中的计数器功能并不合理[38],而应当放置到 CPU 中处理,这样既可以保证计数器的灵活性,又能节省硬件空间,降低复杂度,同时还能够避免硬件计数的限制.

2.2 转发规则相关研究

与传统网络类似,SDN 中也会出现网络节点失效的问题,导致网络中的转发规则被迫改变,严重影响了网络的可靠性.此外,流量负载转移或网络维护等也会带来转发规则的变化.SDN 允许管理人员自主更新相关规则,但采用较低抽象层次(low-level)的管理方式来更新规则容易出现失误[39],导致出现规则更新不一致的现象.即便没有出现失误,由于存在更新延迟问题,在更新过程中,转发路径上有些交换机已经拥有新规则,而另一些交换机还使用旧规则,仍然会造成规则更新不一致性的问题.

将配置细节进行抽象,使管理人员能够使用较高抽象层次(high-level)的管理方式统一更新,就可防止低层管理引起的不一致性问题.一般采用两阶段提交方式来更新规则[40]:第 1 阶段,当某个规则需要更新时,控制器询问每个交换机是否处理完对应旧规则的流,并对处理完毕的所有交换机进行规则更新;第 2 阶段,当所有交换机都更新完毕时,才完成更新,否则将取消该更新操作.为了能够使用两阶段提交方式更新规则,在预处理阶段,对数据包打上标签以标示新旧策略的版本号.在转发过程中,交换机将检查标签的版本,并按照对应版本的规则执行相应的转发动作.当数据包从出口交换机转发出去时,则去掉标签.然而,这种方式需要等待旧规则的数据包全部处理完毕才能处理新规则的数据包,这样会造成规则空间被占用进而产生较高的成本.增量式一致性更新算法[41]可以解决规则空间成本较高的问题,该算法将规则更新分成多轮进行,每一轮都采用二阶段提交方式更新一个子集,这样可以节省规则空间,达到更新时间与规则空间的折中.McGeer 提出了基于 OpenFlow 的安全更新协议[42]来完善抽象层两阶段提交方式的安全性,该协议将无法识别新旧规则的报文发送至控制器,来保证流转发的正确性.此外,规则更新过程需要考虑性能问题.Ghorbani 等人[43]研究了虚拟机场景下规则更新算法,该算法可以确保在更新过程中拥有足够的带宽,同时不会影响到其他流的正常转发.

由于 OpenFlow 无法主动增加未知协议,为保证新协议能在 SDN 中使用,OpenFlow 只能将该协议更新到接口的规格说明书当中,并未真正做到协议无关.因此,如何做到协议无关转发,成为数据平面可扩展的重要研究方向之一.在数据平面建立与协议无关的流指令集(flow instruction set,简称 FIS),可以做到协议无关的转发[44].协议无关的 FIS 抽象了数据平面,每个协议相关的规则会转化成 FIS 中协议无关的指令,并被数据平面硬件所识别,从而实现快速转发.协议无关的 FIS 使规则与转发设备无关,提高了数据平面的可扩展性,真正实现了控制平面与数据平面的全面分离.

3 控制层关键技术研究

控制器是控制层的核心组件,通过控制器,用户可以逻辑上集中控制交换机,实现数据的快速转发,便捷安全地管理网络,提升网络的整体性能.本节首先详细阐述了以 NOX 控制器[12]为基础的两种技术改进方法:一种是采用多线程的控制模式,另一种是通过增加分布式控制器数量,实现扁平式和层次式控制模式.然后介绍了主流接口语言的研究发展,实现控制语言抽象.最后,深入分析了控制器的一致性、可用性和容错性等特性.

3.1 控制器设计问题

控制器的基本功能是为科研人员提供可用的编程平台.最早且广泛使用的控制器平台是 NOX[12],能够提供一系列基本接口.用户通过 NOX 可以对全局网络信息进行获取、控制与管理,并利用这些接口编写定制的网络应用.随着 SDN 网络规模的扩展,单一结构集中控制的控制器(如 NOX)处理能力受到限制,扩展困难,遇到了性能瓶颈,因此仅能用于小型企业网或科研人员进行仿真等.网络中可采用两种方式扩展单一集中式控制器:一种是通过提高自身控制器处理能力的方式,另一种是采用多控制器的方式来提升整体控制器的处理能力.

控制器拥有全网信息,能够处理全网海量数据,因此需要具有较高的处理能力.NOX-MT[45]提升了 NOX 的性能,它是具有多线程处理功能的 NOX 控制器.NOX-MT 并未改变 NOX 控制器的基本结构,而是利用传统的并行技术提升性能,使 NOX 用户可以快速更新至 NOX-MT,且不会由于控制器平台的更替产生不一致性问题[46].另一种并行控制器为 Maestro[47],它通过良好的并行处理架构,充分发挥了高性能服务器的多核并行处理能力,使其在网络规模较大情况下的性能明显优于 NOX.

对于众多中等规模的网络来说,一般使用一个控制器即可完成相应的控制功能,不会对性能产生明显影响[48].然而对于大规模网络来说,仅依靠多线程处理方式将无法保证性能.一个较大规模的网络可分为若干个域,如图 3 所示.若保持单一控制器集中控制的方式来处理交换机请求,则该控制器将与其他域的交换机之间存在较大延迟,影响网络处理性能,这一影响将随着网络规模的进一步扩大变得无法忍受.此外,单一集中控制存在单点失效问题.通过扩展控制器的数量可以解决上述问题,也就是将控制器物理分布在整个网络中,仅需保持逻辑中心控制特性即可[49].这样可使每个交换机都与较近的控制器进行交互,从而提升网络的整体性能.

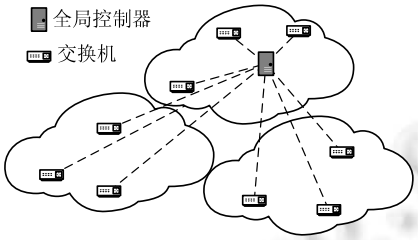

Fig.3 Single controller in SDN

图 3 SDN 中的单一控制器

分布式控制器一般可采用两类方式进行扩展[26],分别是扁平控制方式(如图 4 所示)和层次控制方式(如图 5 所示).对于扁平控制方式,所有控制器被放置在不相交的区域里,分别管理各自的网络.各控制器间的地位相等,并通过东西向接口进行通信.对于层次控制方式,控制器之间具有垂直管理的功能.也就是说,局部控制器负责各自的网络,全局控制器负责局部控制器,控制器之间的交互可通过全局控制器来完成.

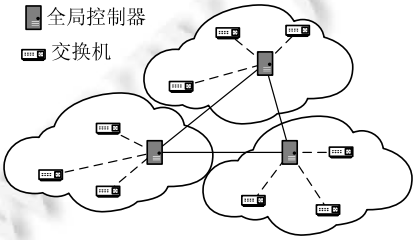

Fig.4 Flat controllers in SDN

图 4 SDN 中的扁平控制器

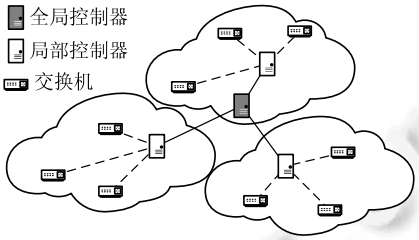

Fig.5 Hierarchical controllers in SDN

图 5 SDN 中的层次控制器

扁平控制方式要求所有控制器都处于同一层次.虽然物理上各个控制器位于不同的区域,但逻辑上所有的控制器均作为全局控制器,掌握全网状态.当网络拓扑变化时,所有控制器将同步进行更新,而交换机仅需调整与控制器的地址映射即可,无需进行复杂的更新操作,因此,扁平分布式扩展对数据层的影响较小.Onix[50]作为首个 SDN 分布式控制器,支持扁平分布式控制器架构.它通过网络信息库(network information base,简称 NIB)进行管理.每个控制器都有对应的 NIB,通过保持 NIB 的一致性,实现控制器之间的同步更新.HyperFlow[51]允许网络运营商部署任意多个控制器,并将这些控制器分布在网络的各个角落.控制器之间保持着物理分离而逻辑集中的特点,因此仍然保持 SDN 集中控制的特点.HyperFlow 通过注册和广播机制进行通信,并在某控制器失效时,通过手动配置的方式将失效控制器管理的交换机重新配置到新控制器上,保证了可用性.在扁平控制方式中,虽然每个控制器掌握全网状态,但只控制局部网络,造成了一定资源的浪费,增加了网络更新时控制器的整体负载.此外,在实际部署中,不同的域可能属于拥有不同经济实体的运营商,无法做到控制器在不同域之间的平等通信.

层次控制方式按照用途将控制器进行了分类.局部控制器相对靠近交换机,它负责本区域内包含的节点,仅掌握本区域的网络状态.例如,与临近交换机进行常规交互和下发高命中规则等.全局控制器负责全网信息的维

护,可以完成需要全网信息的路由等操作.层次控制器交互存在两种方式:一种是局部控制器与全局控制器之间的交互,另一种是全局控制器之间的交互.对于不同运营商所属的域来说,仅需协商好全局控制器之间的信息交互方式即可.Kandoo[52]实现了层次分布式结构.当交换机转发报文时,首先询问较近的局部控制器.若该报文属于局部信息,局部控制器迅速做出回应.若局部控制器无法处理该报文,它将询问全局控制器,并将获取的信息下发给交换机.该方式避免了全局控制器的频繁交互,有效降低了流量负载.由于这种方式取决于局部控制器所处理信息的命中率,因此在局部应用较多的场景中具有较高的执行效率.

SDN 网络操作系统应当具有实时运行所开发应用的能力,即,能够达到开发与执行的平衡.NOX 采用 Python 或 C++,使用 Python 时开发效率较高,执行效率较低;而使用 C++时执行效率很高,开发效率却很低.于是,科研人员致力于开发通用的平台,尽可能地在开发与执行之间达到一个较好的平衡.Beacon[53]是一个基于 Java 的通用平台,它向用户提供了一系列相关的库与接口用于开发,并提供运行时模块化的功能,使其在保证性能的情况下具有了实时运行的能力,实现了开发与执行两者之间的平衡.操作系统将资源形成文件系统,文件具有层次结构,方便操作系统随时读取与调用.控制器所管理的资源也具有类似特征,能够使得用户和应用通过标准的文件 I/O 进行交互.yanc[54]控制器即采用了文件系统的方式处理资源,它把网络应用视为不同的进程,每个进程被分隔成不同的视图或分片(slice).在 yanc 中,无论物理交换机还是数据流参数都将形成文件,不同的应用则根据自身的需求调用不同的文件.此外,SDN 控制器平台已经存在大量实际开发,包括 Floodlight[55],POX[56]和 Ryu[57]等.典型控制器的对比情况见表 3.

Table 3 Comparison of general controller platforms

表 3 典型控制器平台对比

控制器	线程	分布	实现语言	开发团队
NOX[12]	单	否	C++/Python	Nicira
NOX-MT[45]	多	否	C++/Python	Toronto
Maestro[47]	多	否	Java	Rice
Onix[50]	单	是	C++/Python/Java	Nicira
HyperFlow[51]	单	是	C++	Toronto
Kandoo[52]	单	是	C, C++ & Python	Toronto
Beacon[53]	多	是	Java	Stanford
Floodlight[55]	多	是	Java	Big Switch
POX[56]	多	否	Python	Nicira
Ryu[57]	多	是	Python	NTT

3.2 接口语言

控制器提供了北向接口,方便用户配置网络.然而,当今的网络存在各式各样的应用,如流量监控、负载均衡、接入控制和路由等.传统的控制器平台(例如 NOX[12])仅能提供低级配置接口,且使用像 C++这种通用语言,抽象程度较低,造成网络配置成本并未大幅度降低.针对这种情况,科研人员致力于开发一种抽象的、高级配置语言.这些抽象语言能够统一北向接口,改善接口的性能,从而全面降低网络的配置成本.

耶鲁大学团队开发了一系列网络配置语言,旨在搭建具有优化性能的通用北向接口.Nettle[58]是描述性语言,采用了函数响应式编程(functional reactive programming,简称 FRP)方式.FRP 是基于事件响应的编程,符合控制器应对各种应用的实际响应情况.Nettle 的目标是实现网络配置的可编程化,而可编程化的网络配置要求控制器性能足够高,因此,该团队又提出了 McNettle[59],一种多核 Nettle 语言.McNettle 并未改变描述性与 FRP 的特性,而是利用共享内存、多核处理的方式增强了用户开发体验.Procera[60]则进一步对语言抽象作了优化,采用高级的网络策略来应对各种应用.为了使控制器更好地发挥接口语言的性能,该团队提出了 Maple[61],如图 6 所示.Maple 对接口语言作了进一步抽象,允许用户使用自定义抽象策略.为了能够高效地将抽象策略分解成一系列规则,并下发到相应的分布式交换机上,Maple 不但采用了高效的多核调度器,最关键的是,它采用了跟踪运行时优化器(tracing runtime optimizer)来优化性能.该优化器一方面通过记录可重用的策略,将负载尽可能地转移到交换机来处理.另一方面,通过动态跟踪抽象策略与数据内容及环境的依赖性,使流表始终处于最新状态,从

而确保抽象策略转成可用规则的效率.

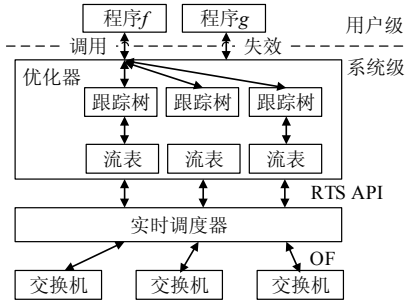

Fig.6 Maple system components

图 6 Maple 系统组件

康奈尔大学与普林斯顿大学联合研究团队首先提出了 Frenetic[62],与 Nettle 等语言类似,它也具有描述性和 FRP 特性.两者主要有 3 点不同:一是抽象层次,由于 Frenetic 是建立在 NOX 之上的语言,因此比 Nettle 抽象层次要高;二是 Frenetic 有查询语言和实时系统,这是 Nettle 所不具备的;三是 Frenetic 基于数据包,而 Nettle 基于事件流.NetCore[63]增强了 Frenetic 的能力,包括增加通配符匹配处理模式、主动生成规则和通用的语法结构等.由于 Frenetic 和 NetCore 都采用模块并行组装方式,这要求每个模块(即服务,如接入控制等)都各自生成所需包的备份,造成一定的资源浪费.Pyretic[64]则采用了模块顺序组装方式,使得每个应用可以顺序处理,也有效地避免了潜在应用冲突发生的可能.为了增强这类接口语言的理论基础,2014 年初,该团队提出利用 Kleene 代数理论作为检测的 NetKAT[65]语言,它改进了 NetCore 语言特性,可以利用数学理论验证配置的正确性,避免了网络配置潜在的问题.上述各类语言的对比情况见表 4.

Table 4 Comparison of interface languages

表 4 接口语言的对比

接口语言	描述性语言	FRP	实现语言	抽象层次	特点
NOX[12]	否	否	C++/Python	低	仅提供 IP 地址操作等低配置接口语言
Nettle[58]	是	是	Haskell	较高	较早采用描述性语言和 FRP 的抽象语言之一,基于事件流
McNettle[59]	是	是	Haskell	较高	多核 Nettle
Procera[60]	是	是	FML[66]	高	与 Frenetic 采用包操作不同,采用事件流
Frenetic[62]	是	是	Python	高	比 Nettle 语言更抽象,有查询语言,实时系统,基于数据包
NetCore[63]	是	否	Haskell	高	改进 Frenetic,采用通配符匹配,主动生成规则等功能
Pyretic[64]	是	否	Python	高	采用顺序组合模式扩展了 Frenetic 和 NetCore
NetKAT[65]	是	否	任意	高	增加 Kleene 代数理论

3.3 控制层特性研究

控制层存在一致性、可用性和容错性等特性,而所提到的这 3 种特性的需求无法同时满足[67],达到三者之间的平衡,是今后的重要工作之一.多数情况下,科研人员是在一部分特性影响较小的前提下重点提升另一部分特性.因此,科研人员在对控制层特性研究中各有侧重.下面对控制层的一致性、可用性和容错性分别进行讨论.

1. 一致性

集中控制是 SDN 区别于其他网络架构的核心优势之一,通过集中控制,用户可以获取全局网络视图,并根据全网信息对网络进行统一设计与部署,理论上保证了网络配置的一致性问题.然而,分布式控制器仍然具有潜在的不一致性问题.由于不同控制器的设计对网络一致性要求有所不同,严格保证分布式状态全局统一的控制器,将无法保证网络性能;反之,如果控制器能够快速响应请求,下发策略,则无法保证全局状态一致性.性能无明显影响的情况下,保证状态一致性成为了 SDN 设计中的关键问题[49].

并发策略同样会导致一致性问题,可由控制层将策略形成规则,并按两阶段提交方式[40]解决.为了避免数据

层过多的参与,控制层可直接通过并发策略组合的方式来解决[68],并可利用细粒度锁(fine-grained locking)确保组合策略无冲突发生.HFT[69]采用了层次策略方案,它将并发策略分解,组织成树的形式,树的每个节点都可独立形成转发规则.HFT 首先对每个节点进行自定义冲突处理操作,这样,整个冲突处理过程就转化成利用自定义冲突处理规则逆向搜索树的过程,从而解决了并发策略一致性问题.

2. 可用性

规则备份可以提升网络的可用性.RuleBricks[70]针对规则备份提出一种高可用性方案.在 RuleBricks 中,不同的规则对应不同颜色的"砖块","砖块"的大小由通配符的地址空间大小决定,其中,最上层的"砖块"对应的规则是目前的活跃规则.如果因为网络节点失效导致某种颜色的"砖块"消失,则下面的备份"砖块"会显现出来成为新的活跃规则.通过对"砖块"的设定、选取和操作的优化,RuleBricks 可以有效限制流的重分配和规则爆炸的问题.

控制器作为 SDN 的核心处理节点,需要处理来自交换机的大量请求,而过重的负载会影响 SDN 的可用性.利用分布式控制器可以平衡负载,提升 SDN 的整体性能.特别地,对于层次控制器(如 Kandoo[52])来说,利用局部控制器承担交换机的多数请求,全局控制器则可以更好地为用户提供服务.然而,分布式控制器架构亦存在可用性问题.由于每个控制器需要处理不同的交换机,网络流量分布不均匀,导致某些控制器可用性降低.针对该问题,ElastiCon[71]采用负载窗口的方式来动态调整各控制器间的流量.ElastiCon 周期性地检查负载窗口,当负载窗口的总负荷发生改变时,将动态扩充或压缩控制器池,以适应当前实际需求.如果负载超过控制器池最大值时,则需要另外增加新的控制器,以保证网络的可用性.

减少交换机的请求次数,可以提升控制层的可用性.DIFANE[72]架构旨在解决数据平面转发规则粒度过细和对集中控制依赖的问题.在 DIFNAE 中,对于 SDN 传统的流处理都交给了数据层,如交换机不再将每流的第 1 个数据包传到控制器等.控制器的任务仅是划分规则,并将规则主动下发到数据层面.因此,DIFANE 可适应大规模的网络拓扑结构和处理更多的转发规则.DevoFlow[73]则按粒度将数据流分成长短流,并在转发器上建立一些特定规则,使数据层能够直接处理短流,仅有少量的长流才交由控制层处理.根据 Zipf 定律[74],长流数量远少于短流数量.因此,DevoFlow 采用的策略可以最大程度地降低控制器负载,提升控制器的可用性.

3. 容错性

与传统的互联网类似,SDN 同样面临着网络节点或链路失效的问题.然而,SDN 控制器可以通过全网信息快速恢复失效节点,具有较强的容错能力.网络节点恢复收敛过程如图 7 所示:① 当某台交换机失效时,其他交换机察觉出变化;② 交换机将变化情况通知控制器;③ 控制器根据所掌握的信息,计算出需要恢复的规则;④ 将更新发送给数据平面中受到影响的网络元素;⑤ 数据平面中受影响的元素分别更新流表信息.

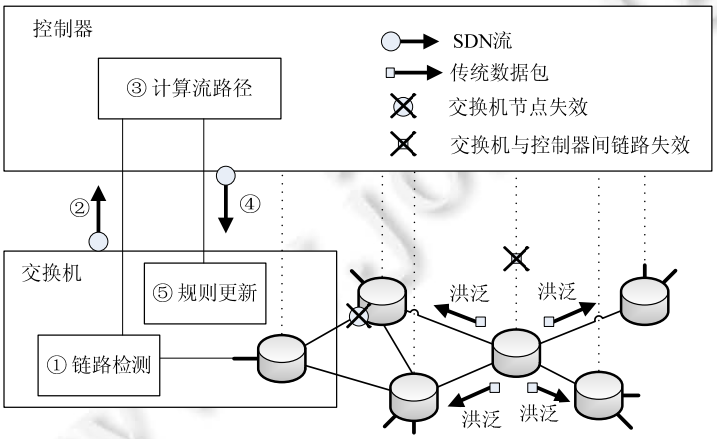

Fig.7 Convergence on a node or link failure

图 7 失效节点或链路的收敛

从链路恢复过程可以看出:在 SDN 架构中,失效信息一般不是通过洪泛方式通知全网,而是直接发送给控制层,并由控制器来做恢复决策,因而不易出现路由振荡的现象.如果是交换机和控制器之间的链接失效,导致无法通信,则收敛过程相对困难.可以采用传统网络的 IGP(如 OSPF 协议)通信,并通过洪泛方式恢复,也可以采用故障转移(failover)[50]方式,同样能够缓解链路失效收敛时间问题.通过在交换机上安装用于验证拓扑连接性的静态转发规则,可以更好地实现网络故障的快速收敛[75].

为了避免由于手动配置导致节点失效,控制层提供了一种高级网络容错语言 FatTire[76].用户可以通过 FatFire 语言指定网络当前的容错度,并根据网络状况自主指定流路径.FatTire 语言编译器具有网内快速恢复机制,可以将用户错误的网络配置迅速恢复回来,提升了控制层的容错性.

4 SDN 应用研究

随着 SDN 的快速发展,SDN 已应用到各个网络场景中,从小型的企业网和校园网扩展到数据中心与广域网,从有线网扩展到无线网.无论应用在任何场景中,大多数应用都采用了 SDN 控制层与数据层分离的方式获取全局视图来管理自己的网络.

4.1 企业网与校园网

在企业网或校园网的部署应用多见于早期的 SDN 研究中[4,12,77,78],为 SDN 研究发展提供了可参考的依据.在之后的实际部署中,由于不同企业或校园对 SDN 的需求存在差异性,无法根据自身的特点进行部署.针对该问题,研究人员完善了 SDN 的功能,支持对企业网和校园网的个性管理.精灵架构[79]允许企业网根据各自需求自主增加新功能,该架构采用外包的形式进行,并且支持企业网增加终端主机、部署中间件、增加交换机和路由器等.Kim 等人[80]进一步研究了利用 SDN 改善网络管理,更好地支持校园网的部署.网络部署一致性问题同样引起了关注,用户通过 SDN 管理网络时仍然会出现网络转发拓扑循环和无效配置等问题.OF.CPP[81]则利用 ACID(数据库事务正确执行的四要素)思想较好地修复了这些问题,有利于企业网络统一部署.

第二代中国教育和科研计算机网(China Education and Research Network II,简称 CERNET2)采用 4over6 技术将百所院校连接在一起,提供了 IPv4 应用和 IPv6 应用接入和互通互访等服务.4over6[82]描述了 IPv4 网络向 IPv6 网络过渡的技术,如图 8 所示,它借鉴了 SDN 网络虚拟化的思想,将 IPv4 网络和 IPv6 网络从数据层分离出来.由于 IPv4 和 IPv6 传输数据的基本原理相同,因此数据层能够对 IPv4 和 IPv6 两者都提供传输服务,实现转发抽象.同时,还可分别为 IPv4 服务提供商和 IPv6 服务提供商提供更方便的管理机制,便于 IPv4 网络向 IPv6 网络的迁移,满足所有 IPv6 网络过渡的需求.

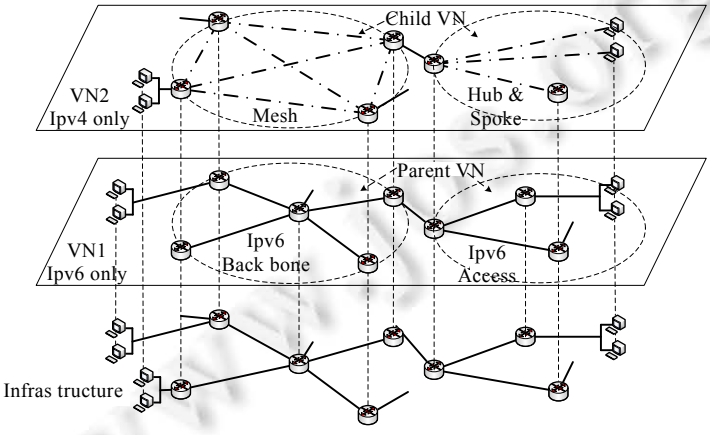

Fig.8 4over6 virtualization architecture

图 8 4over6 虚拟化体系结构