软件定义网络系统中面向流的调度算法

宛 考 罗雪峰 江 勇 徐 恪
(清华大学计算机科学与技术系 北京 100084)

摘 要 软件定义网络(Software Defined Networking,SDN)由于其控制和转发分离的特性,在学术研究和工程上广泛应用于数据中心等领域. SDN 体系结构中没有规定特定转发机制,而是通过更改控制模块或者开发具体的控制器 App 来实现. NOX 是斯坦福大学在 2008 年提出的第 1 种 OpenFlow 控制器,它基于 OpenFlow 协议提出了类似传统路由器 OSPF 协议的最短路径转发算法,即 NOX 路由模块(NOX routing module). 该算法通过 SDN 系统控制链路收集全局交换机静态参数,并没有利用 SDN 系统的优势来获得交换机运行时的状态信息来提高转发性能. 该文根据 SDN 集中控制以及控制器计算能力可扩展等特性,设计 SDN 结点、路径和流的评价体系,将各体系分支的评价指标作为交换机运行时的状态信息参数,在此基础上提出均衡算法. 该 Online 算法具有自适应性和启发性,有效地提高了 SDN 在转发过程中(特别是针对大数据量)的运行效率.

关键词 软件定义网络;自适应启发式算法;集中控制;流分析;下一代互联网
中图法分类号 TP393 **DOI 号** 10.11897/SP.J.1016.2016.01208

The Flow-oriented Scheduling Algorithms In SDN System

WAN Kao LUO Xue-Feng JIANG Yong XU Ke
(*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)

Abstract Software Defined Networking (SDN) has been widely applied to many areas by research and operational communities, such as Data Center Network, with the nature of decoupling control and data planes. There are no predefined routing algorithms strictly limited to SDN's design strategies, which are recommended to be realized by reconstructing or programming at the controller layer. NOX, which is the first controller designed by researchers in Stanford University, has presented a simple shortest path routing algorithm based on OpenFlow, i.e. NOX routing module. This module has realized the shortest path routing based on traditional routing mechanism, while it hasn't fully taken the advantages of SDN, such as sensing and detecting status of the network. The NOX routing module just try to gather some static information from all of the switchers through secure channel. To make full use of central control of SDN and its scalability of controller, this paper have developed an evaluation system to estimate the priority value of the nodes, links and flows in SDN, which is considered as the runtime state information that used as the parameters in balance algorithm, which is online and adaptive heuristic, with a module to collect real-time status information and another module to assess this status. The experiments have proofed that these ameliorations have improved the efficiency of flow forwarding, especially in the scenario with high burden of routing large scale of data.

收稿日期:2014-12-15;在线出版日期:2015-07-23. 本课题得到国家自然科学基金(60503053,61140454,61170292)、国家科技重大专项基金(2012ZX03005001,2014ZX03002004)、国家"九七三"重点基础研究发展规划项目基金(2012CB315803)、国家"八六三"高技术研究发展计划项目基金(2013AA013302)及深圳市软件定义网络技术重点实验室资助.**宛 考**,男,1982 年生,博士研究生,主要研究方向为计算机网络体系结构、下一代互联网、云计算、SDN. E-mail: wank12@mails.tsinghua.edu.cn. **罗雪峰**,男,1989 年生,硕士研究生,主要研究方向为下一代互联网、SDN. **江 勇**,男,1975 年生,博士,教授,中国计算机学会(CCF)会员,主要研究领域为计算机网络体系结构、下一代互联网、移动互联网、互联网应用等. **徐 恪**,男,1974 年生,博士,教授,中国计算机学会(CCF)会员,主要研究领域为下一代互联网体系结构、高性能交换机和路由器体系结构、P2P 网络、Overlay 网络、物联网等.

Keywords　software defined networking; adaptive heuristic algorithm; central control; flow analysis; next-generation Internet

1 引　言

软件定义网络(Software Defined Networking, SDN)系统作为信息网络研究的一种实现方式,其控制平面和数据平面分离能够提供信息处理的高度可控、易于实现和部署等特性.目前 SDN 系统基本以 OpenFlow[1]为主要协议进行开发和实现,在企业网、数据中心、接入网、网络虚拟智能等应用中有颠覆性的表现.开放网络基金会(Open Networking Foundation,ONF)①组织成员中的一些大型网络技术公司都推出了自己的 SDN 体系实现,如 IBM-SDN[2]、Nicira-SDN[3]、G-Scale Network[4].在学术研究中,基于集中控制的网络流量负载、流传输质量是研究的热点.

SDN 最著名的应用场景是提高数据中心之间的链路利用率.Google 的 B4 网络中[4],使用 SDN 调度专用交换机能够使得链路利用率达到 95%以上,Microsoft 的 SWAN 控制器[5]除了具有高链路利用率外,还能够自动解决拥堵更新(congestion-free update)问题,并能使用非专用商业交换机.以上两种 SDN 构架和方案是基于广义网和数据中心之间的,调度的数据流量需求预先已知,不能很好应对非预知数据,同时方案在小规模数据中心或者数据中心内部实现代价过大,本文提出的基于 SDN 系统面向流的启发式均衡算法能够在一定程度上解决以上两个问题.

在集中式控制机制下调度资源并使系统状态达到最优化已有相关研究,如文献[6]中提出在数据中心中根据当前网络负载情况调整交换机数量的方法实现负载均衡;文献[7]中对 SDN 控制器进行了分析,通过建立规则集来实现流分布,减小控制器过载和延迟,以及控制规则表规模.这两种算法都是通过更改传输机制的方式,因此都需要修改 OpenFlow 协议.文献[8]中提出增加一个实体 Rack Manager 来处理源路由匹配问题,减少控制器匹配负担,但其收集信息都是通过静态调用实现不能很好应对网络流量变化情况.文献[9]中提出一个调度系统,设计定制化的路由策略,能够动态调度计算和带宽资源;文献[10]中把负载均衡算法作为插件嵌入 NOX.这 3 篇文章都提出了增加额外事件驱动模块辅助控制器,但这些模块只是针对当前状态,不能对未来网络运行状态进行预测.文献[11]在 SDN 交换机和普通交换机混合节点网络中提出了基于 OSPF 的 SOTE 算法,通过动态分配权值减轻了混合局部拥堵路径,提高整个混合网络的工作效率.但是该算法抓取信息是即时信息,仅对当前状态有效.

为了让调度算法获得网络各实体的状态信息并能达到最优化部署,本文将网络实体运行状态信息收集和权值评价体系与调度算法脱离,以线程独立的模块运行在控制器系统中.其中权值评价体系主要针对运行过程中节点、路径和流进行实时估值评价.各模块工作方式为:控制器额外增加信息收集模块收集实体运行信息,优化计算模块,使其根据一些规则对这些收集信息进行评价后得出相应估值,流分配模块再根据这些估值选择最佳匹配方案,调度模块根据最佳方案进行部署.通过以上 4 个独立模块协同工作后可以使系统运行更加高效.

在权值评价体系中,本文使用启发式自适应算法来预测未来网络运行状态.由于 SDN 系统特性是全局控制和基于流分析的,因此可以收集的信息既有系统历史信息,也有系统全局信息;既有交换机节点信息,也有流信息.控制器端通过收集有用信息来估计未来系统状态(自适应),并通过预测的未来状态求出可能性最优解(启发式),因此可制定更加合理优化的转发策略.

本文提到的自适应启发式算法应用在以下几个方面:(1)流的动态性预测.根据测量流的出现次数来估算流的发生频率,预测未来流分布情况.引入流量参数能够更加准确地描述流的运行状态;(2)估算交换机转发上限阈值.节点评价模块通过当前承载转发的流对交换机有一定预判,在发生堵塞之前能够及时预警,避免节点堵塞不能工作;(3)评估节点转发性能.预测节点剩余带宽或者剩余流量,链路算法以此为权值进行最短路径选择;(4)路径评估.对两节点所有路径进行筛选,创建可选路径集合,并在可选路径中预测未来路径上的流状态,为路径选择算法提供估计参数;(5)对流进行评估.当节点发生超载时,为了节点恢复正常需要转移部分承载流,流评价模块对该节点上运行的流进行评价估值.

本文第 2 节讨论 SVN 系统中面向流的调度算

① https://www.opennetworking.org/

法的背景、可行性和动机;第 3 节具体研究算法细节;第 4 节给出算法实现框架;第 5 节通过实验对算法进行评价;第 6 节讨论未来工作和阐述本文结论.

2 算法背景

本节通过两个实例讨论在最短路径算法中,因不能获得网络中实时状态信息和预测未来运行情况而带来的系统资源利用问题和可能引起拥堵情况,并讨论了 SDN 上实现面对流的调度算法的可行性.

2.1 传统算法的问题

2.1.1 最短路径选择

NOX 和 FOX 默认是以类 OSPF[12] 选择最短路径方式来寻路的,权值为节点带宽. 如图 1 示例,左图显示一些节点已经有些负载. 中图有新流 A→D 申请产生,在节点 AD 间有 3 条可选路径:AEFGD,AGD,ABCD. 通过传统最短路径计算得到路径 AGD. 但此时 A 点 G 点都存在负载,因此可能引起节点超载. 右图通过收集 3 条路径实时运行状态,得出最优路径 ABCD. 这些运行状态是实时的,因此实现算法必须是 Online 的.

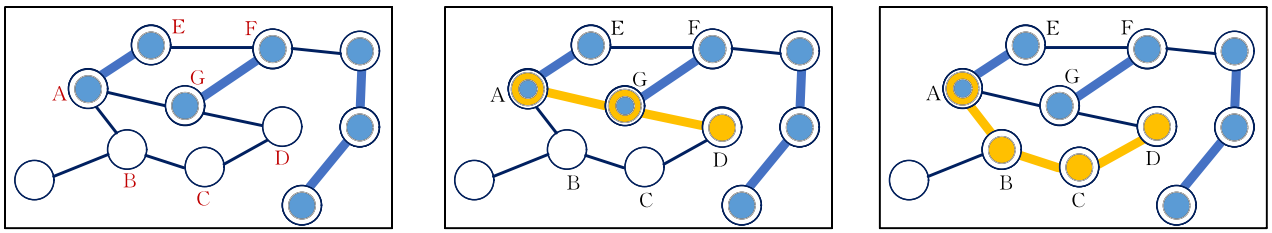

图 1 流选择不同节点(左图显示插入流前网络运行状态;中图显示选择最短路径流的运行状态,经过的节点有两个接近满负荷;右图显示选择最优运行时节点权值,尽管路径不是最短)

2.1.2 系统资源利用情况

静态算法调度时,评价各个节点运行状态是相互独立不相关的,各节点不知道其他节点的运行状态. 每个节点都会优先服务优先值高的流,由于没有全局意识,优先值低的流就会很有可能被节点拒绝服务,但如果全局情况下,评价低的流是可以运行的. 如图 2 示例,左图中显示在某初始环境下,网络中有 3 个流:A→E、A→F、A→D,其分别承载的节点路径为:AE、AGF、ABCD. 各个节点创建本节点阈值来评价继续增加负载的能力,各节点优先选择负载比较小的流,这样能够使得承载该服务后不超过阈值. 系统某时刻需要创建新的流 A→D,假设在评价体系下,流 A→D 负载很大. 在独立节点情况下,各个节点不考虑全局信息,只根据本节点运行状态创建本节点阈值,如左图所示. A 相邻结点 E、G、B 因节点本身独立阈值设置,均不能负载该流,系统拒绝承载该服务. 中图因控制器知悉全局情况,控制器可调整 E 节点分流,流 A→F 改变路径为 AEF,这样 G 节点因空出流 A→F 的承载能力,可以承载流 A→D 的服务,创建路径 AGD. 这种调度方式在路由器没有全局实时智能下不会出现,通常做法是拒绝服务造成丢包. 右图是另外一种分配方式,将 ABCD 当前运行的流段移到 AGD,新建的流以 ABCD 路径进行传输. 控制器可根据一些权值判定那种方式更加合理,再具体进行部署.

在大量复杂拓扑情况下,当流集合总流到达系统能力阈值时,此种情况经常出现,造成系统资源不能充分利用的情况.

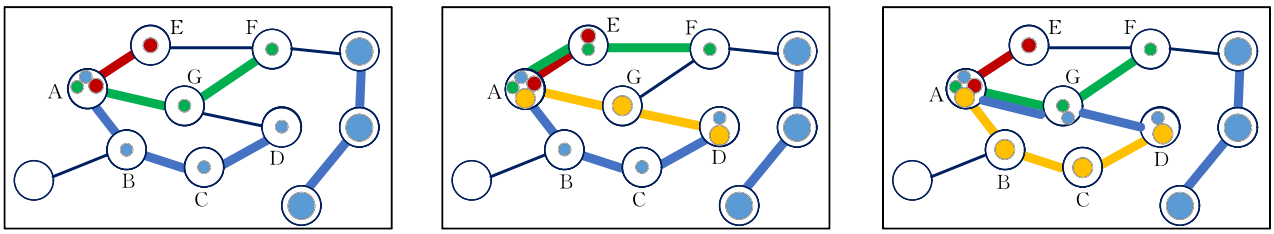

图 2 全局信息下流的可分配(左图有 3 个流:AE、AGF、ABCD. 当创建新的流 A→D,左图 A 相邻结点 E、G、B 因节点本身独立阈值设置,均不能负载该流,系统拒绝承载该服务. 中图因知悉全局情况,可调整分流,流 AGF 转向为 AEF,这样 G 节点因空出流 AGF 的承载能力,系统可以承载流 A→D 的服务,创建路径 AGD. 右图是另外一种可能的路径分配方式)

2.1.3 Offline 算法问题

文献[6-8]中提到的算法是静态 Offline 算法. 其核心实现方式是:收集部分或者全局的信息,通过最优化算法算出最优解,再根据最优解进行资源重新

部署. Offline 算法的问题是收集的信息是基于瞬时的，其最优解只对收集时刻的整个系统状态有效. 如果系统不是基于时不变假设(Non-Time Variant Assumption)，当系统运行一段时间后，系统状态发生变化，之前最优解便有可能失效. 同理，当发生节点超载时，由于处理错误模块的算法计算和部署延迟性，当系统部署好处理策略后其超载节点已经不是超载状态，因此处理方法失效.

2.2　运行时状态算法在 SDN 上可行性

为了提高系统运行性能和减少失效配置，需要有运行时状态算法对全局进行均衡调度. 该算法需要满足：(1)基于网络节点和链路的实时信息；(2)信息是全局性的；(3)算法能够在一段时间内具有有效性；(4)算法能够快速部署，并且部署代价比较小.

SDN 系统以下几个特性能够比较好的满足以上几点，使得运行时状态算法在 SDN 系统上具有可行性.

2.2.1　集中控制

本文所讨论到的集中控制，主要关注在集中收集全局信息上. SDN 并不只限于集中式控制，一些商业控制器都有分布式控制模型，如 Google、Nicira、NEC 共同设计的 Onix 模型[13]，并在 B4 网络中应用. 一些超大型网络收集全局信息可能需要多个控制器并对信息碎片进行拼接. 但本文的算法不是瞬时全局信息相关的，其启发式可以允许有一定的信息“时差”.

传统路由器网络中，OSPF 通过 LSA 得到全局网络拓扑以及各节点间路径传输带宽，路由器分布式转发全局性信息，并独立维护全局信息副本. 文献[14-15]中对 OSPF 和 IS-IS 协议重新设置权值，目的是为了尽可能描述节点和链路运行时状态，但还是以静态方式实行，因此需要多次触发权值收集和计算模块，这种间接性获得全局信息机制有一定局限性. 如果需要更多的全局性信息，如运行时节点状态，运行时链路状态及运行时传输流状态，路由器需要更复杂的扩展模块，这种高性能智能路由器功能升级困难，命令行接口及配置复杂，部署成本昂贵，难以实际推广应用.

相对于传统网络，SDN 额外提供控制器实现集中控制功能，控制器通过有专门的控制链路连接各交换机来收集全局信息，有专门的操作系统以及 App 实现复杂的调度计算功能. 在传输过程中，集中控制的主要实现有收集全局信息、执行控制功能、全局部署策略.

2.2.2　实时性控制信息丰富

SDN 是基于流分析的，如 PCEP 协议和 BGP-LS/SNMP 的路径记录，ForCes 协议的 XML 文本描述. 本文将以 OpenFlow 作为示例具体分析，并在实验中实现.

对于流的相关分析，其分析粒度可在 OpenFlow1.0 规范定义十元组(输入端口、MAC 源地址、MAC 目的地址、以太网类型、VLANID、IP 源地址、IP 目的地址、IP 端口、TCP 源端口、TCP 目的端口)进行选择，没有被选择的元组通过通配符号代替，如在防火墙级别的流分析上，元组可选择传统五元组. 通过流分析，能够得到更加丰富的运行时状态信息. OpenFlow 流定义中，就有多种信息供控制器读取. 在控制器端设置特殊的流分析模块，可获得更多信息如流应用特性、带宽大小、长短流、流的出现频率等等. 流的描述特征越丰富，其对流的实时预测越准确. 文献[16]通过设计链路权值使局部网络传输稳定，文献[17]通过 MPLS 标签处理来保持网络流量稳定，文献[18]通过在 QoS 支持的网络中对网络事件进行学习来预测未来情况，这些都是额外增加路由器模块来获取网络复杂信息.

SDN 作为全网状态敏感的系统，为基于全网感知的算法提供实现基础. 控制器收集到的全局信息不再局限于网络静态信息，还可以收集系统运行时的状态信息，如节点工作信息、流传输信息. 这些信息为 Online 算法提供了很好的参考数据，如 LABERIO[19]. 通过这些信息可使用自适应启发式的算法来预测未来特性，因此会更智能更高效. 算法的计算代价全部由控制器及额外模块承担，在交换机层并不感知，因此具有很强的可执行性.

SDN 协议对流的操作控制使得可以进行多粒度处理，算法本身提供的可操作项也更多. 如当节点运行出现热点即超过阈值时，控制器可自适应选择最大可能转出该节点的部分工作流将其排出，并安放这些流到其他节点，而不仅标记该点不可用.

2.2.3　可编程和可扩展构架

在传统路由器结构中，一些具有多个管理功能结构的调度算法往往需要高性能路由器来实现，这些路由器往往价格昂贵，且配置复杂和功能升级代价过大. 在 SDN 系统中，具体的控制器系统提供了 API，我们可通过编写具体控制器系统的 App 对网络资源进行调度和操作. 本文通过在控制器端增加额外的模块来收集信息、计算权值和优化数据集，这些模块是线程独立的，调度模块会根据这些结果计

算最优结果并部署到系统中.

3 算法设计

3.1 流的动态特性和流参数的规约简化

设 SDN 系统中所有流 $X_1, X_2, \cdots, X_n$ 组成集合 $\mathbb{X}$,对于某个流 X_i,运行带宽为 b_{X_i},出现频率为 Λ_{X_i}. 流频率本身具有时间的统计特性,可以通过建立流的历史信息库,对其启发式预测未来运行情况. 信息库样本量越大,预测会越精确. 对流频率的信息收集方式可以通过以下两种方式实现.

(1)传输协议规定流在传输过程中携带相关描述信息,运行过程中触发一些事件时,可让控制器事件处理模块调度流分析程序,对流携带的信息进行储存和统计. 在 OpenFlow 的 1.3 版本(目前 OpenFlow 最新发布版本为 1.5,本文仅讨论稳定版本 1.3)中,与流运行时状态相关的描述信息字段如表 1 所述.

表 1 OpenFlow 流消息描述字段

字段意义	字段名
流表项自创建起时间	duration_secduration_nsec
流过期前时间	idle_timeouthard_timeout
流中包计数	packet_count
流中字节计数	byte_count

从表 1 中可知这些信息字段是基于交换机流表的,这些字段不太关注流本身运行时工作状态. 因此通过流携带描述相关信息这种方式需要重新扩展协议标准,更改和增加控制器系统底层模块对新加字段的解析功能.

(2)在控制器 APP 层增加对流频率测量模块和储存模块. 这种方式比较灵活,使用控制器 OS 提供的功能和 API 接口,可根据考察标准的不同定制不同监测方式和收集储存方式. 本文采取这种方式.

实际操作中由于无法预知准确流特性,Λ_{X_i} 值可通过流检测模块对其历史信息进行计算得到的统计值 λ_{X_i} 近似得出. 假设信息收集模块更新频率 Fr_c,已经收集信息 c 次,其中流 X_i 出现 x_i 次,即 $\lambda_{X_i}=x_i/c$. 对于节点集合 $\mathbb{S}$ 某个节点 S,记上面运行的流集合为 $X(S)$,流 X_i 在其出现概率为 $\lambda_{X_i}(S)$,设该流路径上节点集合为 P_{X_i},则有如下关系:

$$\forall A,B\in P_{X_i},\ \lambda_{X_i}(A)=\lambda_{X_i}(B)=x_i/c;$$

$$\forall C\notin P_{X_i},\ \lambda_{X_i}(C)=0.$$

λ_{X_i} 具有一定时刻相关性,也和流 X_i 承载内容性质相关,如一些流在白天出现频率很高,夜间出现频率很低. 频率可用 $\lambda_{X_i}(T,App)$ 表示,T 代表时刻、App 代表流种类,这取决于对流分析粒度.

流的分析粒度大小是通过流观测值参数数量来调整的. 2.2.2 节中提到 OpenFlow1.0 规范的十元组可作为流分析的基础参数. 但这些参数不是必须的,参数数量同信息收集模块复杂度和运行消耗代价有很大联系. 在数据中心网络中流数量和种类都非常多,为了减轻收集和计算模块负担,我们需要减少流的参数数量,因此对流参数的简化和分类规约是必需的. 文献[20]中通过 FlowVisor 从网络功能和流类型划分的粒度对网络流进行规约,这种方法控制了分析粒度,但不能获得运行时统计特性. 文献[21]中分析数据流之间的通配特征,减小了对控制器的访问,这是从细粒度对流进行的分析. 本文仅选取了流的起始地址和目的地址二元组,以交换机节点为基本单位,对网络性能进行均衡分析.

3.2 节点转发阈值上限

本文认为节点被动拥堵造成的损失远大于节点主动拒绝服务丢包造成的损失,因为被动拥堵是不可控,无法通过有效算法解决的. 因此需要设定阈值,避免节点超载后运行到完全瘫痪. 通过交换机节点负载流的带宽总和的带宽占用率 $P_{wl}(S)$ 作为某个节点 S 运行时承载状态. 设 $b(S)$ 为节点带宽,工作流负载占用带宽为 $wl(S)=\sum_{i=1}^{n}b_{X_i}$,则有 $P_{wl}(S)=wl(S)/b(S)$.

一般情况下管理员根据经验或者由交换机缺省自定义阈值,通常在 50%~70%之间. 这种设定的静态阈值不能根据节点实际运行情况进行自动浮动,当不同流经过某交换机节点时,控制器因固定阈值采取的处理事件应对不能很好适应实际运行的流状态,从而对整体性能造成影响. 固定阈值特别不适应承载动态和不能预知的负载,如交换机节点承载带宽小的流情况. 假设短流 X_{short} 产生前,系统中已有流集合 $X_1,\cdots,X_n$ 运行,对系统资源配置已存在在优化状态 $S_{\text{op}}(X_1,\cdots,X_n)$. 当流 X_{short} 在交换机节点 S 生成时使得该节点负载超过固定阈值 $TH(S)$,引发系统处理节点超过阈值事件. 控制器对系统中各交换机节点进行流调控重新分配后,得到优化状态 $S_{\text{op}}(X_1,\cdots,X_n,X_{\text{short}})$. 短流 X_{short} 很快结束并在很长时间内不再产生,此时系统中流 $X_1,\cdots,X_n$ 运行在有短流 X_{short} 存在的优化配置 $X_1,\cdots,X_n,X_{\text{short}}$ 下. 因为系统工作在流集合 $X_1,\cdots,X_n$ 下,有 $S_{\text{op}}(X_1,\cdots,X_n)\leq S_{\text{op}}(X_1,\cdots,X_n,X_{\text{short}})$,即系统状态 $S_{\text{op}}(X_1,\cdots,X_n,$

X_{short})在X_1，…，X_n下不是最优状态. 没有预测流X_{short}的运行时状态就会造成此类情况发生，此时如果将阈值调高些就不会触发系统重分配事件，从而避免引起的次优状态后果.

流进行规约后在相同路径在一段时间内流的性质是相同和具有延续性的，如长短流带宽、占总流带宽比重、发生频率等等. 因此可以通过自适应启发式算法，根据在某个节点上承载流的历史特性预测未来特性. 根据流未来特性，可以设置节点的上限阈值.

3.2.1　通过节点带宽估计阈值上限

本文引用绝对中位差 MAD(Median Absolute Deviation)[22]统计方法处理节点历史信息. MAD 作为稳健统计方法它为传统统计方法提供了一种替代方案. 其目的是生成一种不会受模型假设的细微偏差过分影响的模拟量. MAD 是统计离差的一种稳定估计量，对于没有均值或者方差的分布来说，它具有更好的性能，如 Cauchy 分布，因此比样本方差或者标准差具有更广泛的应用. 对于数据集中的异常值，MAD 比标准差具有更好的容忍性. 由于标准差对样本值与均值的差取平方，所以从平均来看，大偏差值会造成更严重的偏离，这样异常值的影响将更大. 而对于 MAD，少量异常值与均值之差的数量级是可以忽略不计的. MAD 定义为

$$MAD(X)=median_i(|X_i-median_j(X_j)|),$$

由 3.2.1 节定义，经过当前节点的 MAD 值为

$$MAD=median_i(|b_{X_i}-median_j(b_{X_j})|),$$

定义节点带宽上限阈值为

$$TH^u=1-\alpha\cdot MAD,$$

其中，$\alpha\in R^+$，为自定义参数，系统管理员通过调整 α 值来微调节点转换流的程度. MAD 值越小说明该节点上的流带宽很平稳，节点上线阈值可以调大些，节点可以承载更多的流. MAD 值越大，说明该节点各流带宽差别很大，需要调小上限阈值避免带宽大的流突发造成节点过载拥塞.

3.2.2　通过节点流量估计阈值上限

上节 MAD 参考的是流的带宽统计值，流生成频率λ_{X_i}没参与计算. 在简单系统中，运行时带宽作为参数调节交换机节点阈值已经够用，这是因为简单系统中流种类较单一且频率变化不大. 但在复杂系统中，流种类较多频率变化频繁，因此调节节点阈值需要考虑流频率参数. $\lambda_{X_i}b_{X_i}$作为流量，能够更精确描述流运行状态，能很好地解决频率高的短流优先级过低和频率低长流优先级过高的问题. 通过流量估计 MAD_F值定义如下：

$$MAD_F=median_i(|\lambda_{X_i}b_{X_i}-median_j(\lambda_{X_j}b_{X_j})|),$$

定义节点的流量上限阈值为

$$TH_F^u=1-\alpha\cdot MAD_F.$$

以上两种算法复杂度为 $O(n^2)$. 需要注意的是，当采取 4.1 节中计算模块，每次计算都有上一次数据记录，我们不需要每次都统计所有样本，因此算法复杂度可降为 $O(n)$，每次计算因有上一次均值参与，故具有自适应性.

3.3　节点评估

当交换机节点负载超过阈值后，控制器需要对该节点上的流进行重新调度，减轻该节点负载. 对于选出来的流，需要对其进行重新寻找路径算法操作. 触发该算法事件有：当一个流需要重新放置时或者产生新的流. 传统 OSPF 协议运行最短路径算法，对节点评估权值为$\lceil Metric/b(S)\rceil$，其中 $Metric$ 通常取静态值为 10^8 bit，通过 LSA 传输信息. 该算法有两个问题：

一是节点样本量过大. 最短路径算法基本通过 Dijkstra 算法实现，此算法的时间复杂度为 $O(n^2)$，n 为节点个数. 当没有采取启发性时，网络节点探测方式是圆形波阵面展开的，当结点数量很多时，算法消耗过大. 有一些基于 Dijkstra 优化的算法，方式是对样本节点进行预优化，这样可以使得选择样本减少，目前认为最好能够到 $O(l+n\log n)$[23]，l 为链接个数.

二是节点带宽不能反映当前节点运行时状态. 传统算法中没有根据实际节点实际运行时情况，$b(S)$越大的节点，被分配的流越多，因此实际上越拥堵. 因此需要对节点进行运行状态评估.

节点评估算法对当前节点可承载状态进行评估，路径选择算法淘汰大量不在阈值范围的节点以减少样本量. 通过这两个步骤能够有效解决传统算法的两个问题. 采用该算法代价是在控制器端增加全局监控模块，独立动态计算模块以及矩阵维护模块. 这些额外的模块在数据转发过程中的消耗可忽略不计，因为本身算法是根据流情况在少量样本中取值，这些样本和取值都是独立模块已经维护好的统计结果.

3.3.1　节点剩余带宽评估

节点剩余带宽评估(Node Left Bandwidth Evaluation，NLBE)方式为：设节点 S 可被调用带宽为 $Avail(S)$，上限阈值为 $TH^u(S)$，一般的有

$$P_{wl}(S)\leq TH^u(S),$$

则剩余带宽为

$$l(S)=b(S)-wl(S).$$

可被调用带宽为

$$Avail(S)=TH^{u}(S)\cdot b(S)-wl(S).$$

拓扑中节点集合$\mathbb{S}$中各节点状态$b(S)$,b_{X_i}为实际观测值,通过信息收集模块收集,并被写到节点状态向量.$wl(S)$,$TH^{u}(S)$,$Avail(S)$通过计算模块读取节点状态向量矩阵得到,并被写到节点状态向量对应区域.路径算法读取$Avail(S)$作为权值,淘汰大量不在阈值$TH_{Avail(S)}$范围内节点(本节算法为小于阈值情况),计算得出预测最优路径.其中

$$TH_{Avail(S)}=\sum_{X_i\in\boldsymbol{X}_{\text{trigger}}}b_{X_i},$$

$\boldsymbol{X}_{\text{trigger}}$为触发最短路径算法发生时刻节点$S$同时新生成流集合,一般情况下为了降低算法复杂度和增加即时反应速度,$TH_{Avail(S)}$取静态经验值.其算法复杂度为$O(n)$.

3.3.2 节点剩余流量评估

NLBE算法是基于节点本身的当前承载能力的,但是流频率λ_{X_i}没有被考虑,$\lambda_{X_i}b_{X_i}$流量传输特性对未来某节点运行状态有很大影响.假设某个时刻系统检测到一个带宽大的流正在运行,算得$Avail(S)$值比较小.但其发生频率很小,之后便很少出现.这样会使得这段时间内实际$Avail^{ac}(S)$值远大于计算$Avail(S)$,造成资源浪费.同样情况,如果流带宽小但频率大,实际$Avail^{ac}(S)$值远小于计算$Avail(S)$,造成系统资源最优分配失败率增加.以上两种情况都会造成系统次优状态运行概率增加,整个系统处理能力降低.

节点剩余流量评估(Node Left Flux Evaluation,NLFE)方式为:记节点S可用流量为$Flux_{Avail}(S)$,工作负载可记为

$$wl_{Fl}(S)=\sum_{i=1}^{n}b_{X_i}\cdot\lambda_{X_i}.$$

流可能消失而造成剩下的流量为

$$\sum_{i=1}^{n}(1-\lambda_{X_i})\cdot b_{X_i}.$$

可被调用流量为

$$\begin{aligned}Flux_{Avail}(S)&=TH_{F}^{u}(S)\cdot b(S)+\sum_{i=1}^{n}(1-\lambda_{X_i})\cdot b_{X_i}-wl_{Fl}(S)\\&=TH_{F}^{u}(S)\cdot b(S)+\sum_{i=1}^{n}b_{X_i}(1-2\lambda_{X_i}).\end{aligned}$$

由$Flux_{Avail}(S)$定义可得到一些现象:当$\lambda_{X_i}=1$时,该算法退化到NLBE算法.该流发生频率$\lambda_{X_i}=1$时,可取固定预估值.当流带宽比价平稳,所有的流频率$\lambda_{X_i}\leq 1/2$时,可知一定有足够的容纳带宽不超过阈值的流,这类节点被当做优质节点.最短路径算法选择节点时优先考虑优质节点,这样能够尽量缩短寻找节点时间.其算法复杂度为$O(n)$,因为阈值和流量都是通过前次数据迭代计算的,因此具有自适应性.

3.4 可选路径评估

可选路径评估运用在新建流或者选出一些流后需要重新放置路径,该算法需要额外模块.在某时刻T_i通过最短路径算法选出的节点状态是时刻T_i的状态,NLFE中通过流估计值λ_{X_i}来预估节点未来状态.鉴于SDN控制器控制层面可扩展特性,可以建立两节点间的路径数据库,实时算法可通过可选路径上节点状态来评估路径情况.当网络拓扑改变时,触发寻找两点路径集合算法,更新各网络路径数据库,该数据库仅记录两节点连通情况,非运行状态的复杂信息.本算法默认网络节点硬件上开关频率很低,如在信息收集频率内不会开关多次.这样收集到的节点信息被认为是有效的.运行状态信息收集模块收集各节点运行时信息,当发现某条路径上的一些节点超载时,排除这些路径,在剩下可选路径中,通过下面的评价算法获取信息,判定可选最优路径.

3.4.1 路径带宽评估

路径带宽评估(Link Bandwidth Evaluation,LBE)方式为:设路径集合开始节点S_{sour}和结束节点S_{des}表示为,$Path(S_{\text{sour}},S_{\text{des}})$为两节点可选路径集合,$P_i$为可选路径集合某一条路径,$S_{i,j}$为$P_i$路径上的某一传输节点.则$P_i$路径可用带宽为每条路径上节点带宽最小值决定,即有

$$b_{P_i}=\{\min b_{S_{i,j}},\ S_{i,j}\in P_i\}.$$

路径选择算法选择带宽最大的路径,即

$$P_m=\{\max b_{P_i},\ P_i\in Path(S_{\text{sour}},S_{\text{des}})\}.$$

其算法复杂度为$O(n)$.

3.4.2 路径流量评估

路径流量评估(Link Flux Evaluation,LFE):流X在路径P_i上各节点$S_{i,j}$的带宽被认为是一致的,但流的实际频率Λ_{X_i}在各节点测得频率不等.由于信息收集模块收集信息频率Fr_c并不和优化调整模块频率Fr_o同步,为了使优化调整信息有效,有$Fr_o-Fr_c=\Delta F$.其中,ΔF为两模块频率差,ΔF过大会使得收集信息时效性降低,ΔF过小频繁触发优化事件会产生过优化状态,即整个系统已经接近优化状态,再次触发优化取得优化结果差别并不大.

Fr_o除了受到优化调整模块主动触发影响外，还受到流生成和交换机节点超载受到影响，因此 Fr_o是个动态值，为反馈函数：

$$Fr_o = func(\lambda_{X_1},\lambda_{X_2},\cdots,\lambda_{X_n},Fr_o,Fr_c,TH^u_{S_1},TH^u_{S_2},\cdots,TH^u_{S_n}).$$

为了便于取值，一般使 ΔF 主动取值 $\Delta F\in[0,Fr_c/2]$，从而得到估计值 Fr_o.

在可选路径 P_i 上每个节点 $S_{i,j}$ 上测到的频率 $\lambda_X(S_{i,j})$并不是相等的. 其相对误差为 $|\lambda_X(S_{i,a})-\lambda_X(S_{i,b})|\leq\Delta F, S_{i,a},S_{i,b}\in P_i$.

流 X 如存在多条可选路径，则各路径必然存在至少两个以上(源和目的两个节点)重合节点. 重合节点上的流 X 频率，接近等于该节点上所有可选路径频率之和. 如图 1 中，流 A→D，有 3 条可选路径，节点 A 上测试到的流频率为 3 条路径流频率之和. 节点 G 上测试到的流频率为两条路径上流频率之和. 因此对于流 X 在 P_i 上的频率，可取非重合节点最大值：

$$\lambda_X(P_i)=\{\max\lambda_X(S_{i,j})\mid S_{i,j}\in P_i, S_{i,j}\notin Path(S_{sour},S_{des})-P_i\}.$$

通过统计流在各可选路径上的概率，可估计该流在可选路径上产生的未来流量为

$$Flux_X(P_i)=\lambda_X(P_i)\cdot b_{P_i}.$$

对于路径的选择，有两种选择方法：

(1)最大流量(LFE-MFE)

最大流量指选择该流所有经过的路径中有最大流量路径，即选择路径：

$$P_m=\{\max Flux_X(P_i),P_i\in Path(S_{sour},S_{des})\}.$$

因为该流在这条路径上运行流量最大，被认为该条路径容量最适合该流，该条路径同其他可选路径比起来，不会因为进行调整而最先选择该流. 其算法复杂度为 $O(n)$.

(2)最佳匹配(LFE-BFE)

流在运行中，流量有大有小. 最佳匹配指选择流量最合适的路径，选择合适路径后，相差流量趋近为 0，使得该剩余流量不浪费，流 X 的流量期望为

$$\overline{Flux_X}=\sum_{i=1}^{n}Flux_X(P_i)\cdot\lambda_X(P_i).$$

取路径：

$$P_m=\{\min|Flux_X(P_i)-\overline{Flux_X}|, P_i\in Path(S_{sour},S_{des})\}.$$

其算法复杂度为 $O(n)$，以上流量取值都基于流频率参数，因此具有自适应性.

3.5　流评估

传统传输方式中节点对流的处理方式是做尽力转发服务，如果节点不能承担服务则丢弃转发包，一些协议如 TCP 会根据丢包情况被动调节流量. 这种以节点为单位的分布式控制转发方式在全局上对流进行分析控制比较弱，而 SDN 的集中控制方式能够将监控、收集信息、计算路径、转发等过程进行解耦而不损害转发性能和效率，通过独立的进程可以专门全局对流进行控制，可以收集更加丰富的信息. 通过这些信息集可以对流进行评价计算，独立的转发进程可根据这些评价值进行更好的流调度.

同样受到流分析粒度大小影响，如果粒度过细，造成的系统负载和延时过高. 为了使得评价模块有效运行，本文算法只对规约到二元组的流进行评估.

当一个节点在运行中超载后，控制器需要对该节点进行修正，以便该节点回到正常负载状态.

传统情况采取鸵鸟策略(No aware Politics)：最原始的方法就是交换机节点保持工作状态不变，继续对承载的流进行转发服务，该节点继续超载直到堵塞(即是负载超过 100%). 交换机堵塞后会触发堵塞事件，控制器会标记该节点堵塞，再把上面承载的流调度到其他节点. 此种策略下交换机必然会出现传输堵塞事件，控制器会进行堵塞事件处理以致系统传输性能降低.

在 SDN 系统中可采取方法很多，鉴于 SDN 集中控制和控制能力可扩充特性，当一个节点上的负载超过阈值时，可以对当前流进行选择，将一些流转移到其他路径. 这种方式需要对流进行评估，以便相关选择算法根据其评估值进行选择. 在选择流中，算法并不复杂，一般选择权值最高最值得排除的流.

同节点评估和路径评估不同的是，流评估不仅仅注重运行时状态，更加重视其“迁移”的困难程度.

3.5.1　流运行时状态评估

控制器流评价模块评价流运行时状态，可参考参数有流带宽、流频率、流流量、流内容等. 其目的是把在当前节点状态最不好的流选出，并能使其正常工作.

(1)带宽评估

最小带宽方法(Minimum Bandwidth，Min-B)是从带宽最小的流往上选择. 将当前节点流集合 $X(S)$ 占用带宽 b_{X_i} 进行排序，得到流带宽有序序列：$\{X_1,X_2,\cdots,X_n\}$. 从最小到最大开始累加，其和第 1 次达到需要移动的带宽量 b_{out} 时，开始排除这些短流. 设向上差值为

$$\Delta b_{\uparrow}=\sum_{i=1}^{m} b_{X_i}-b_{\text{out}},$$

即求

$$m=\{\min \Delta b_{\uparrow} \mid \Delta b_{\uparrow}\geq 0, X_i\in X(S)\}.$$

最大带宽方法(Maximum Bandwidth,Max-B)是将当前节点流带宽排序后,直接选择最大带宽的流移出.设向下差值为

$$\Delta b_{\downarrow}=\sum_{i=n}^{m} b_{X_i}-b_{\text{out}},$$

即求

$$m=\{\min \Delta b_{\downarrow} \mid \Delta b_{\downarrow}\geq 0,\ X_i\in X(S)\}.$$

使用最小带宽方法好处是能够恰到好处地选出空间,使得空出浪费最小.因为带宽大的流移出来能够合适的路径不太多,其他节点可提供带宽都不够.缺点是这些带宽小的流需要 m 次部署工作.最大带宽方法能够让空余的带宽可以填补更多的流,该节点超载的几率降低,移出流的重新部署的次数少了,缺点是被移出的带宽大的流可能没其他交换机节点有足够的剩余带宽能够承担.其算法复杂度为 $O(n)$.

(2)流量评估

通过带宽评价流运行状态不需要额外模块,但不能更加准确评估流运行状态.当增加流量 $b_{X_i}\lambda_{X_i}$ 时,也可用类似带宽评估的方法进行流量评估.将带宽 b_{X_i} 参数改成流量 $b_{X_i}\lambda_{X_i}$,可使用最小流量方法(Minimum Flux,Min-F),最大流量方法(Maximum Flux,Max-F).其算法复杂度为 $O(n)$.

3.5.2 流可管理状态评估

流管理评价主要在控制器调度流方面,当对流进行操作时会有相应代价.一般情况在流传输中是路径分配好就不再移动的,除非路径节点发生超载触发重新寻找路径.但在数据中心中当需要高效承载更多服务时,需要优化流的路径选择,对流的操作评价就需要考虑流移动的代价.流的"管理性质"分两种场景:一是指 SDN 系统对流可控程度,如流非正常工作管理,流转移的困难程度;二是指在移动流场景中移动便捷性和移动后提高系统整体效率程度.

3.5.2.1 违反服务等级协议评估

在流转发中,默认流的频率 λ_{X_i} 是基于稳定概率的,其占用带宽 b_{X_i} 也是在一定上下限范围之内的.但是,也有可能在流传输中出现奇点情况,如某一时刻带宽激增.用户本身可能并不知道,如 DDOS 攻击等等,流量不受交换机带宽限制突然增大.本文算法为了避免出现这种情况,使得统计数据对原本数据造成污染,附带了违反服务等级协议(Service-Level Agreement Violation,SLAV)处理模块.

目前,SDN 并没有官方的类似于云基础设施即服务(Infrastructure As A Service,IAAS)层的完整契约模块,如限制流量(本身是最大服务的),契约服务,用户管理等等.本文在实验中增加部分简单功能,避免奇点干扰减少统计误差,该模块在算法中是可选的.在 SDN 扩展模块中,也可以通过流用户同 SDN 进行契约协商,即服务等级协议(Service-Level Agreement,SLA).SDN 系统承诺满足服务,如果 SDN 系统违反发生 SLA 情况,系统会有一些惩罚补偿.但用户发生 SLAV,则系统可对其服务进行选择,拒绝服务或者评估为优先级比较低的服务权值.Radware① 公司在 OpenDaylight② 控制器系统框架内提出了 Defense4All 产品,主要目的是防范 DDOS 攻击,为用户提供安全管理和服务.本文增加类似模块,该模块在算法中是可选的,仅辅助信息采集模块实现简单功能:通过检测流量避免奇点干扰减少统计误差.

需要注意的是流量激增本身不能准确标明该流是受到劫持或是处于不正常工作状态,因此需要与用户契约模块协同工作,与其中声明的流量大小进行比对.OpenDaylight 项目中 Virtual Tenant Network 提出了租户概念,该子项目还在发展中,但没有特定标准规定租户应该声明何种消息格式和内容.

其算法思想如下:设一个流 Y 第 i 次统计该流的带宽记为 $b_Y(i)$,经过第 n 次统计,该流 Y 统计的平均流量为

$$\overline{b_Y}=\frac{1}{n}\sum_{i=1}^{n} b_Y(i).$$

在第 $n+1$ 次,如果流量激增,即超过本身某个限度时 $b_Y(n+1)-\overline{b_Y}\geq \mu\cdot \overline{b_Y}+\nu$,该流被认为是超过系统能够承担能力,其中参数 $\mu\in R^+$、$\nu\in R$ 可以作为系统与用户的服务契约.

流 X_i 发生违约时,被 SLAV 管理模块设置权值 $Sl_{\text{ten}}(X_i)$,并被记录到该流的状态库中.$Sl_{\text{ten}}(X_i)$ 可适当取比较大的值,以便让流选择程序将该流移出繁忙节点,避免引起整个系统性能降低.

SDN 因自身调度也发生违反 SLA 情况时,控制器也可增加模块记录流 X_i 被 SDN 系统违反 SLA 情况.在调度流时,可通过 SLAV 管理模块设置权值 $Sl_{\text{sys}}(X_i)$,并通过设置系统惩罚值 $Penalty(Sl_{\text{sys}}(X_i))$

① http://www.radware.com/
② http://www.opendaylight.org/

增加调度模块的权值来提高该流服务质量. 这种方式对某一具体流具有优化性，不提高整个系统性能. 在本文中，该部分在实验部分被忽略.

3.5.2.2　流可移动性评估(Flow Movable Evaluation，FME)

流管理中有两种情况：(1) 新的流 X_{new} 生成时，设 X_{new} 开始节点和目的节点分别为 S_{sour} 和 S_{des}，因节点负载情况，系统无法通过最短路径算法创建一条最短路径 P_i 来负载该流；(2) 对已运行的流 X_t，当前所用路径为 P_i，可选路径集合 $Path(S_{\text{sour}}, S_{\text{des}})$，$S_{i,j}$ 为 P_i 路径上的某一传输节点. 假设运行过程中 $S_{i,j}$ 拥堵或者超过阈值，系统会选择另外一条可选路径 P_a 来传输流 X_t.

传统方法中如果没有可选路径，即：(1) 没找到 P_i 或者 (2) 没找到 P_a，系统就认为无法服务. 不过在集中控制情况下，系统可尝试预判是否可以将部分权值低的流转移到其他路径，在路径 P_i 增加额外能力来承载流 X_i 服务. 流的可移动性评估可通过更改流表的节点数目来参考.

(1) 情况中，可选择 $Path(S_{\text{sour}}, S_{\text{des}})$ 中路径上承载流量最小的路径 P_i，尝试增加流 X_{new}，来得到过载节点集 $S_{ov}(P_i)$，从而规约到 (2) 情况.

对每个过载节点 $S_{i,j} \in S_{ov}(P_i)$ 上承载的所有流集合 $X(S_{i,j})$ 中每条流 X_i，通过路径选择算法得到新可选路径 P_a 后，有相同路径节点集合为 $P_i \cap P_a$. 这对于原路径 P_i，需要在流表中减少 X_i 流表项的节点集合为 $S_{\text{del}} = P_i - P_i \cap P_a$，需要在流表中增加 X_i 流表项的节点集合为 $S_{\text{add}} = P_a - P_i \cap P_a$.

流转移代价 $Mov(X_i)$ 定义为 $Mov(X_i) = Cost(S_{\text{add}}) + Cost(S_{\text{del}})$，在 SDN 系统中，各节点调整流转移代价有着细微差别，与各节点流表以及整个系统的运行状态有关，本算法认为每个更改节点消耗相等. 流评价权值定义为 $M_{X_i} = Mov(X_i) - \eta \cdot Sl_{\text{ten}}(X_i)$，其中 $Sl_{\text{ten}}(X_i)$ 为可选值.

根据以上分析，则可以在当前节点 $S_{i,j}$ 上承载的所有流集合 $X(S_{i,j})$ 中，对所有流进行筛选和权值排序. 选出权值较小的流进行移出. 设有 r 个，将这些流移出后，路径空出流量为

$$M(X) = \sum_{i=1}^{r} b_{X_i} \lambda_{X_i},\ X = (X_{\text{new}}, X_t).$$

其满足条件为 $M(X) \geq b_X \lambda_X$.

当移动流过多或者修改节点过多，系统可能因为修改流表消耗过多时间，修改后系统也不易恢复稳定状态. 通过定义移动阈值 M^{up} 来控制移动情况，当移动节点数超过 M^{up} 可判定不移动该流. 即要求

$$\sum_{i=1}^{r} [S_{\text{del}}(X_i) + S_{\text{add}}(X_i)] \leq M^{\text{up}}.$$

当没有可选路径时，即每条流都找不到替代路径，或者找到路径但不满足 M^{up} 阈值，则认为系统不能经过调整满足服务. 其算法复杂度为 $O(n^2)$，同样由于采取 4.1 节中记录前次优化数据方式，我们不需要每次都全局计算样本值，只需用当前值同上次优化后的值进行计算，其算法复杂度为可降为 $O(n)$. 由于 M^{up} 与具体系统拓扑和任务负债有关，只能通过经验取值，因此流可以移动性评估具有启发性.

4　系统框架及运行流程

SDN 可以通过不同协议来实现，使用 OpenFlow 优点在于能够控制流量通过交换机方式，并通过网络传递到网络拥有者、单个用户或者单个应用程序. 能够让用户制定一些路径政策，以便找到可用带宽、较少延迟或阻塞. OpenFlow 也有扩展性和安全性等弱点. 安全性弱点是因为集中控制的原因，外部攻击可以从一个点攻击，不像分散式的会相对安全. ONF 现在也有专门的项目组解决安全性问题，但目前还没有解决扩展问题的项目组. OpenFlow 协议 1.3 版本提高了扩展性，但还达不到商业程度需求. 同时，目前实现 SDN 北向接口(North Bound Interface，NBI) 的方法在数量已经超过 20 多个，虽然每一种方法都很有价值，但是数量过多也造成编程接口的混乱，网络服务提供商、系统架构和应用开发人员都有采用不同的接口来实现自己的 SDN 应用实例的方案. 因此本文提出的各个模块，可能部分能够通过使用一些控制器提供的 API 实现，但不能保证所有控制器 API 能够完全实现所有模块. 在不能用 API 实现的情况下，需要编写独立模块协同控制器系统工作.

4.1　系统框架

节点、流和路径等网络状态信息运算模块通过控制器 App 实现，各状态信息通过数据库储存其历史信息，其原始数据、计算数据和优化数据存于对应数据表中，供与状态信息管理模块提取. 某时刻的优化数据是通过当前网络状态数据和该时刻前一时刻优化后的历史数据计算而来，因此不需要对大量历史原始数据计算便可得到当前时刻最优值. Framework 中各模块进程独立，各模块负责结点管理，路径管理，流管理，SLAV 管理功能. 调度模块负载选取优

化后数据进行部署. 以 Path 模块为例,模块内部构成如图 3.

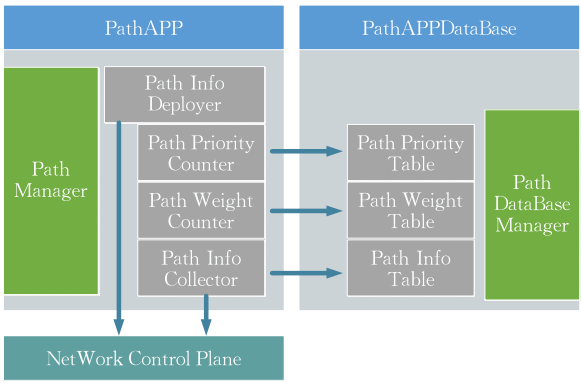

图 3 模块构成(以路径模块为例)

各模块内部功能如下:

(1) 信息收集模块

收集相关信息,包括节点运行状态、节点位置、链路情况、流运行状态信息等等. 收集信息过于频繁,系统全局信息会比较精确,但带来的消耗也过大. 在实际运行过程中,可由系统管理员设定一个初值 F_0,然后根据实际运行情况及反馈信息,调整收集信息频率,避免因为收集信息带来的消耗,具体值在 5.1.3 节中设置.

(2) 权值计算模块

通过文章中提到的自适应启发式算法,通过收集到的信息数据库,计算相关权值.

(3) 优先值计算模块

该模块将权值根据评价算法进行计算,得出最终优选结果或者优选结果集合,由调度模块选取进行部署.

其中,为了尽量缩短自适应算法样本的收集时间,需要一定频率将各状态信息保存到数据库. 模块重新启动能够读取这些数据,这样能够获得之前已经统计的样本信息,提高预估值精确度. 估价数据库包括流经过节点,流带宽等流运行状态信息. 当新流增加时可以改写新流增加事件,控制器登记同时可以直接建立流初始化信息. 当一个节点没经过预估值突然堵塞时,同样可改写堵塞事件,分析堵塞原因(堵塞的流情况),哪个流引起故障,然后记录相关信息到流状态数据库,这个时候可以根据对引起堵塞的流评价 $Sl_{\text{ten}}(X_i)$ 值.

4.2 工作流程

所有模块都是通过事件驱动,其中信息收集和状态计算都是线程独立模块通过一定频率触发. 收集模块收集到信息后,写在原始数据库,再通过状态计算模块进行优化计算,得出一些权值和筛选后的数据集. 调度模块再根据事件处理函数选择这些数据集进行合理计算,得出结果后重新部署.

在实现算法评价时,本文关注的事件如下:

(1) 节点堵塞

节点堵塞由 SDN 系统本身触发事件,可改写节点堵塞事件处理函数,标记引发堵塞的流,并通过流评价模块得到流权值,根据这些权值选择一个或者多个流进行分流.

(2) 节点超载

即节点超过阈值,与节点堵塞不同的是节点超载时节点还可以继续工作. 该事件由检控模块发起预警事件,并由调度模块处理,处理方式同节点堵塞.

(3) 新建流

当节点有新建流时,系统触发新建流事件. 可改写该事件:最短路径算法读取的权值改为状态计算模块算出的值,当最短路径算法找不到路径时添加分流算法.

5 实验和算法评估

状态优化模块通过不同评价算法得出相应数据值,调度算法根据这些值部署到系统中,根据系统运行情况来评估这些评价算法性能,同时与 SDN 系统缺省算法进行比较.

5.1 实 验

5.1.1 平台介绍

由于条件限制,本文选择最方便获得的 Mininet[①],目前 Mininet 是大部分科研机构和工程实验室接受的交换机模拟运行环境. 但 Mininet 设计构架在控制器扩展情况,存在以下讨论议题:(1) 单一交换机不能同时指定连接多个控制器;(2) 同一实验场景中,独立交换机不能指定不同的控制器 IP;(3) 多控制器之间交互和发现. 因此,算法中各独立进程模块只能被部署运行在单一控制器上,实验结果不能很好展现算法框架在分布式控制器环境下性能优势. 尽管如此,算法框架能够对未来运行状态有一定预判性,能够有效避免拥塞,在单机控制器环境中能够体现效率优势,5.2 节中实验结果能够说明情况.

5.1.2 硬件设置

实验选择 POX[②] 作为控制器系统,算法基于 POX 中 Python 脚本实现,可以推广到其他控制器平台. 控制器所在软件环境为 Ubuntu14.04 桌面

① http://www.mininet.org
② http://www.noxrepo.org

64 位版本，硬件环境为 DELL R710，其中 CPU 为双 Xeon E5620，2.4 GHz；内存 16 GB，硬盘 600 GB.

Mininet 中交换机选择胖树为拓扑结构模拟数据中心场景. 在严格胖树拓扑结构情况下[24]，k 个 POD(组)里有 k 个叶节点交换机，每个 POD 下层 $k/2$ 交换机连接 $k/2$ 个 Host，k 个 POD 共可连接 $k^3/4$ 给个 Host，其中有叶节点交换机 k^2 个，中心交换机 $(k/2)^2$ 个，核心路由器 1 个. 一般数据中心一个 cluster 的 Host 数量在 1000 以上，可以取 $k=16$ 来模拟 1024 个 Host 场景，此时叶节点交换机 256 个，核心交换机 64 个.

但这在实际实验中模拟 320 个交换机已达到承载服务器的极限性能，无法再进行模拟流的传输工作. 而在实际数据中心网络拓扑搭建中，为了减少交换机成本，并不遵循严格胖树拓扑结构. 本文实验在 Mininet 环境中各节点组成仿胖树结构 8 个 POD 中各中心交换机与核心路由器相连. 每个 POD 有 8 个交换机，每个叶节点交换机连接 Host 个数服从[11,20]的随机分布，这种结构可处理 700 多个 Host 情况. 整个系统结构有 64 个叶节点交换机，8 个中心交换机，叶节点交换机带宽为 100 MB，中心交换机带宽 1 GB. 模拟环境所在软件环境为 Ubuntu 14.04 服务器 64 位版本，Mininet 版本为 2.1.0. 硬件环境为 DELL R710，其中 CPU 为双 Xeon E5620，2.4 GHz；内存 32 GB，双硬盘 600 GB.

为了在单一承载硬件服务器上尽可能模拟多交换机节点来接近实际数据中心拓扑情况而不降低性能，实验没有设计在交换机传输带宽为 10 GB 和 100 GB 场景. 通过同比例设计传输流量大小，并不影响对算法在实际运行中的效率和性能评估. 实验中 Mininet 没有生成与交换机相连的 Host，这是因为：(1)算法只关心交换机运行性能，将叶节点交换机中各 Host 之间传输的流通过流规约为叶节点交换机之间的流；(2)算法中不涉及 Host，为了节省模拟运行环境承载服务器的资源和提高性能，流直接在节点交换机端产生.

5.1.3　数据设置

在 Mininet 中可以通过 ping，iperf，wget，netperf，netcat 或者 python 环境下的 scapy 等命令在 Mininet 控制台指定某特定节点产生流. 但这些流生成方式是静态的，不能动态模拟实际数据中心流工作情况. 为了更好模拟实际网络运行情况，试验中设计 POX 程序通过独立进程来模拟动态流产生过程，通过 Wireshark 进行接受包统计. 根据 5.1.2 小节中特别指出的 Mininet 模拟数据中心环境的承载服务器实际情况，流带宽与交换机带宽设计成一定比例. 实验中通过设计长流和短流来模拟实际流带宽分布，流传输节点配对在叶节点交换机集合中随机产生，在实验中设置为每个叶节点交换机每秒产生规约后的流(3.1 节)个数为服从泊松分布，其中 λ 值取值范围为[50,2000]，当流数目总数越大时，其并发流配对越多，即 λ 取值越大. 同时规定当发生拥塞时，保存该流下次重发直到传输成功. 在节点交换机/中心交换机带宽 100 MB/1 GB 情况下，大流带宽大小服从[1 Mbps，10 Mbps]为区间的随机分布；小流带宽大小服从[1 Kbps，10 Kbps]为区间的随机分布. 各种相同带宽大小的流发生频率分别服从指数分布，每次实验流的数量总和相等，长短流按照设定配给比例产生，缺省情况下比例各为 50%.

5.1.4　算法设置

实验过程中，系统运行过程如下：

(1)节点正常工作，按照控制器分配好的流表正常进行转发包工作.

(2)通过 MAD 或者 MAD_F 判断节点是否超过阈值.

(3)节点状态超过阈值，触发该节点重新选择路径事件.

(4)路径选择事件使用最短路径(NLBE 和 NLFE)，最优可选路径(LBE 和 LFE)算法，两个路径算法因参数不同分别有带宽和流量两种实现方式.

(5)如果分流事件增加流评价功能，则对流使用可移动性评估(FME)算法.

MAD-NLBE 标签表示实验中采取算法为节点阈值使用带宽估计算法，当需要选择路径时使用节点剩余带宽评估算法. 其他算法标签可从表 2 中得出相关内容. 我们设置以带宽为评价基础的 MAD-NLBE，MAD-LBE 和以流量为评价基础的 MAD_F-NLBE，MAD_F-LBE，MAD_F-NLBE-FME，MAD_F-LBE-FME 算法进行实验，并进行相互比较. 同时我们也引入 NOX 和 Floodlight 中缺省的基于 OSPF 协议的最短路径算法(ShortP)，与以上算法进行比较.

表 2　实验算法简称

英文缩写	中文意义
MAD	节点带宽估计阈值上限
MAD_F	节点流量估计阈值上限
NLBE	节点剩余带宽评估
NLFE	节点剩余流量评估
LBE	路径带宽评估
LFE	路径流量评估
ShortP	最短路径
FME	流可移动性评估

5.1.5 算法时间和初始值设置

由于我们利用额外模块统计流频率,并将最近一次优化后的数据进行记录作为下一次优化操作的输入值,这样减少了大样本量的统计和取值计算. 4.1 节中提到,F_0 值设置同收集信息模块相关,与实验中交换机数目,链路数目和任务数目相关,也同实验机器性能相关. 由于没有准确值,我们只能通过统计数据得出合理值. 我们测到,在任务随机分布情况和 5.1.2 节中硬件环境下,控制器收集所有数据并计算结果进行部署需要 2 s～3 s 延时,我们将 F_0 值设为 3 s. 当 F_0 值超过 10 s 时,当突发任务增多时,优化设置逐渐失效.

5.2 实验评价

系统可通过多种参数评价性能. 本文选系统服务时间宏观上评价各策略在系统上运行效率,选取各个策略运行中发生的分流次数来评价对流服务造成的影响,节点超载预警次数评价策略对节点智能调度能力.

在各项实验中,能够观测到系统存在一个处理能力阈值,超过该阈值系统开始频繁出现部分流得不到服务情况. 该阈值与网络拓扑、流配对分布、流量大小等网络运行时状态有关. 该阈值是动态值,只能通过观测评测数据确定其范围,不能得到准确值.

5.2.1 服务时间

相同流数量和大小情况下,各个算法处理时间不同,我们在相同负载和资源情况下比较各个方案处理时间. 实验结果如图 4 所示.

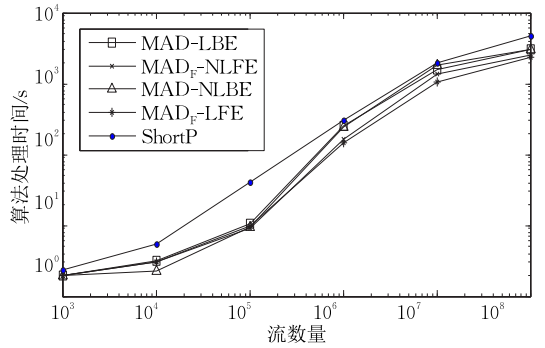

图 4 流处理时间的各算法比较

通过图例可以看到,最短路径算法(ShortP)开始和其他算法差别处于线性关系,但是到了 10^7 附近,性能急剧变差. 这是因为到了系统处理阈值,开始有流不能得到服务. 带宽和流量的权值都能够有效降低寻路节点样本量,因此各优化算法都能够更快响应. 在系统处理阈值之后,也能保持较快的反应时间. 流量能够更加精确的反应节点状态,因此在选择节点算法中选择的路径能够更好地完成流传输服务,体现在整体服务时间更少.

5.2.2 发生的分流事件次数

当新流产生时,如果通过分配路径算法不能找到可承载的路径,就可能触发分流事件. 该情况表明可选路径上已经饱和,我们对算法 MAD$_F$-NLFE 和 MAD$_F$-LFE 分别增加了触发主动分流事件处理:MAD$_F$-NLFE-FME 和 MAD$_F$-LFE-FME,与原算法比较处理结果如图 5 所示.

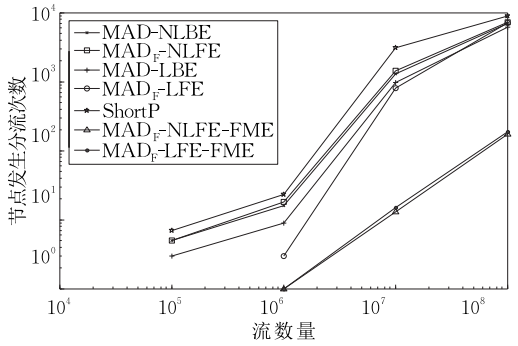

图 5 发生分流事件数目

从实验结果可以看到,在一般情况下,采用 MAD$_F$ 比 MAD 情况要好,采用可选路径比最短路径要好. 不过超过系统处理能力阈值后,可选路径比最短路径要差,这是因为,最短路径获取的是即时信息,可选路径是经过一定频率计算的统计值,超过系统处理能力阈值后,可选路径反应能力没有最短路径快,从而不能得到服务的流更多. 在使用分流处理功能后,系统并不能完全消除该事件,从实验结果可以看出,产生流与该事件大部分满足 log 取值的线性关系.

5.2.3 节点过载次数

本文通过观测节点过载次数来判定算法的稳定性. 如果算法发生节点过载次数过大,则该算法分配流承载路径并不能考虑流运行情况,会造成路径重新分配. 节点过载情况如图 6.

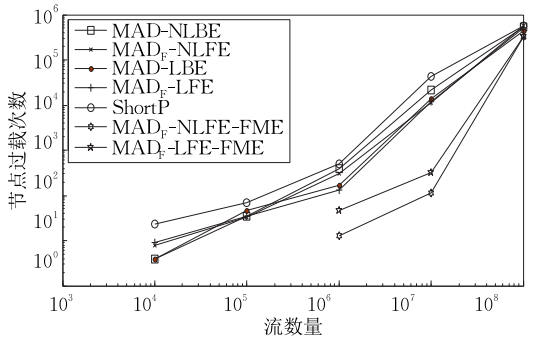

图 6 节点过载情况

从实验结果可以看出，原始的最短路径算法引起的节点过载次数很高，具有运行时情况的算法能够减少过载情况. 但是超过系统处理阈值时各算法节点过载数目趋近相同，这是因为大量流情况已经有很多节点趋于饱和状态，没有过多的空余节点承担额外流. 分流处理进程在系统性能阈值之内能很好的避免节点过载，但超过阈值引起的改变有限.

5.3　实验结论

通过以上 3 个实验场景，对各种评价体系进行实验评价. 发现增加评价体系后，系统在处理能力阈值之内能够有效增加系统性能. 增加额外处理模块后，能够减少系统节点过载等消耗. 但是超过系统能力阈值后，算法性能降低，当需要适时信息时，一些原先运行很好的算法会失效. 此外，本文提出的算法框架中，认为控制器端模块因 CPU 达到 100%，可能会因等待 CPU 分配计算资源造成延时，从而使算法有效性降低. 实验中，在控制器和 Mininet 承载服务器端观测 CPU 负载情况. 在流数目为$[900,10^7]$区间，承载服务器 CPU 运行没有达到 100%. 在流数目 10^8 时，Mininet 端无影响，控制器端有间歇性达到 100%，不过持续时间少于 1000 ms. 在流数目 10^9 时，因控制流的并发发生数目过大，控制器和 Mininet 承载服务器都发生 CPU 过载情况，此种情况下因 Mininet 模拟开始失效，比较难以分析算法有效性，故在实验中没有流数目 10^9 情况.

6　总结与展望

本文详细讨论了 SDN 集中控制和控制能力扩展特性，并对当前转发模式进行了分析. 分析了静态特性的优缺点，引进了节点状态、可选路径、流状态 3 个评价体系，通过在控制器端编写 APP 或者更改基础模块来实现. 在系统运行时各个模块独立线性工作，收集特定信息并加以优化计算，得出优化统计信息提供给相应供控制器调度模块使用. 与一般算法比较，因为有了额外的计算优化结果，控制器不需要额外计算最优值，因此能够迅速反应，保持长时间的最优状态. 通过 Mininet 模拟实际交换机情况，设计 POX 相关 APP，部署后经过实验得出结论：在控制器能力满足额外模块运行需要下，改进后的体系结构和算法能够在系统阈值能力范围内有效的提高传输性能.

随着作者对该问题的研究深入，发现还有许多工作未来值得注意和跟进.

实验改进. 由于 Mininet 的限制，一些实际数据的测量比较难以进行，如：(1) 直接通过测试数据中心的实际流量值来确定算法性能和有效性是最直接的. 但因实际数据中心拓扑庞大，流量节点分散，Mininet 目前还不能达到该扩展度；(2) Mininet 目前不支持多控制器，同时本身不支持多物理服务器承载，因此本文算法中的独立线程模块 scale-out 的特性在少样本量情况下只能体现少量优势.

静态模型扩展. 本文的根据流特性产生的启发性算法，算法复杂度比较小，原因是部署时延比较小. 更精确性的算法，算法复杂度增加，也会增加控制器模块复杂度. 自适应启发算法的上限分析、准确度分析需要进行跟进研究.

动态模型. 在考虑到算法效率时，同时也要考虑到额外模块的复杂度和重新部署的时延，需要折中平衡. 系统本身符合一些 Markov 转换特性，可以通过系统动态建模来实现该模型，通过随机过程的一些特性，在统计学特性上进行预测分析.

流复杂分析. SDN 流量预测模型运行方式是在控制器端增加流信息统计模块，根据流被监控到的历史统计特性来预测其未来行为. 本文对流的分析限于流带宽和频率，流也可从内容特性来推测传输性质. 如根据协议标签对流进行统计性取样标记进行长短流分析，这样也会增加模块复杂度.

模块的扩展. 本文算法是以 POX 为基础，基于 Openflow 协议，所以一些模块是以该系统的 APP 来实现的. 目前 SDN 也有其他控制器系统如 Floodlight、Beacon、Trema、Maestro、SNAC、OpenDaylight 等. 各种控制器系统体系结构并不一样，APP 实现功能的方式不一致，统一的 API 接口能够更加有利于模块的扩展.

模块的性能. 本文提到的模块都运行在控制器主机上，这些计算消耗不算入转发消耗中. 但是控制器主机能力有限，目前没有一个很好的平台能够分散计算能力和数据处理能力. 因此模块数量过多时会影响处理效率.

参　考　文　献

[1] McKeown N, Anderson T, Balakrishnan H, et al. OpenFlow: Enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 2008, 38(2): 69-74

[2] Dixon C, Olshefski D, Jain V, et al. Software defined networking to support the software defined environment. IBM Journal of Research and Development, 2014, 58(2): 1-14

[3] Sherwood R, Gibb G, Yap K K, et al. FlowVisor: A network virtualization layer. OpenFlow Switch Consortium, Technical Report, 2009

[4] Jain S, Kumar A, Mandal S, et al. B4: Experience with a globally-deployed software defined WAN. ACM SIGCOMM Computer Communication Review, 2013, 43(4): 3-14

[5] Hong C Y, Kandula S, Mahajan R, et al. Achieving high utilization with software-driven WAN. ACM SIGCOMM Computer Communication Review, 2013, 43(4): 15-26

[6] Handigol N, Seetharaman S, Flajslik M, et al. Plug-n-Serve: Load-balancing web traffic using OpenFlow//Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication. Toronto, Canada, 2009: 1-2

[7] Wang R, Butnariu D, Rexford J. OpenFlow-based server load balancing gone wild//Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services. Boston, USA, 2011: 12-12

[8] Macapuna C A B, Rothenberg C E, Magalhaes M F. In-packet Bloom filter based data center networking with distributed OpenFlow controllers//Proceedings of IEEE Global Telecommunications Conference. Miami, USA, 2010: 584-588

[9] Handigol N, Seetharaman S, Flajslik M, et al. Aster*x: Load-balancing Web traffic over wide-area networks. GENI Engineering Conferences, 2009, 12(3): 31-34

[10] Koerner M, Kao O. Multiple service load-balancing with OpenFlow//Proceedings of the 13th IEEE Conference on High Performance Switching and Routing (HPSR). Belgrade, Serbia, 2012: 210-214

[11] Guo Y, Wang Z, Yin X, et al. Traffic engineering in SDN/OSPF hybrid network//Proceedings of the 22nd IEEE International Conference on Network Protocols (ICNP). North Carolina, USA, 2014: 563-568

[12] Moy J. OSPF version 2. Network Working Group, RFC 1247, 1991

[13] Koponen T, Casado M, Gude N, et al. Onix: A distributed control platform for large-scale production networks// Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation. Vancouver, Canada, 2010: 1-6

[14] Fortz B, Thorup M. Optimizing OSPF/IS-IS weights in a changing world. IEEE Journal on Selected Areas in Communications, 2006, 20(4): 756-767

[15] Fortz B, Thorup M. Robust optimization of OSPF/IS-IS weights//Proceedings of the 1st International Network Optimization Conference. Paris, France, 2003: 225-230

[16] Applegate D, Cohen E. Making intra-domain routing robust to changing and uncertain traffic demands: Understanding fundamental tradeoffs//Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocolsfor Computer Communications. Karlsruhe, Germany, 2003: 313-324

[17] Elwalid A, Jin C, Low S, et al. MATE: MPLS adaptive traffic engineering//Proceeding of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies. New York, USA, 2001: 1300-1309

[18] Mitra D, Ramakrishnan K G. A case study of multiservice, multipriority traffic engineering design for data networks// Proceedings of IEEE Global Telecommunications Conference. Rio de Janeireo, Brazil, 1999: 1077-1083

[19] Long H, Shen Y, Guo M, et al. LABERIO: Dynamic load-balanced routing in OpenFlow-enabled networks//Proceedings of the 13th IEEE Conference on Advanced Information Networking and Applications (AINA). Barcelona, Spain, 2013: 290-297

[20] Koerner M, Kao O. Optimizing openflow load-balancing with L2 direct server return//Proceedings of the 4th International Conference on the Network of the Future (NOF). Pohang, Korea, 2013: 1-5

[21] Curtis A R, Mogul J C, Tourrilhes J, et al. DevoFlow: Scaling flow management for high-performance networks// Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication. Toronto, Canada, 2011: 254-265

[22] Wu C, Zhao Y, Wang Z. The median absolute deviations and their applications to Shewhart control charts. Communications in Statistics-Simulation and Computation, 2002, 31(3): 425-442

[23] Fredman M L, Tarjan R E. Fibonacci heaps and their uses in improved network optimization algorithms. Journal of the ACM, 1987, 34(3): 596-615

[24] Al-FaresM, Loukissas A, Vahdat A. A scalable, commodity data center network architecture. ACM SIGCOMM Computer Communication Review, 2008, 38(4): 63-74

WAN Kao, born in 1982, Ph. D. candidate. His research interests mainly include computer network architecture, next generation Internet, cloud computing, software defined networking.

LUO Xue-Feng, born in 1989, M. S. candidate. His research interests mainly include next generation Internet, software defined networking.

JIANG Yong, born in 1975, Ph. D., professor, Ph. D. supervisor. His research interests mainly include computer network architecture, next generation Internet, mobile Internet, Internet application.

XU Ke, born in 1974, Ph. D., professor, Ph. D. supervisor. His research interests mainly include architecture of next-generation Internet, high performance router, P2P and overlay network, Internet of Things.

Background

Over the past few years, Software Defined Networking (SDN) has been a key research field of next generation network, for its unlimited potential for revolutionizing the networking by enabling programmability, easier management. SDN possesses a fully novel architecture where control and data plane functionalities are separated and simply designed switches in high-speed data plane forward traffic based on rules installed by control plane program. This separation of concerns leads to a promising combination of the programmability and fine-grained management of network with relatively simple switch design.

Load balancing and route optimizing are hot issues in traffic management with the rapid growth of network traffics. The load balancing aims at distributing traffic evenly among multiple paths, in order to transmit all of the flows in a network using less time, while the purpose of route optimizing is to choose the best routes for a particular flow so as to process the flow with less time. The programmability and complete visibility of SDN control plane enable us to deploy sophisticated management policies in network. By default, there are not any predefined routing algorithms strictly limited to SDN's design strategies, which are recommended to be realized by reconstructing or programming in the control layer. NOX, the first controller designed by researchers in Stanford University, presents a simple shortest path routing algorithm based on OpenFlow, i. e. NOX routing module. But it hasn't fully taken the advantages of SDN, such as sensing and detecting status of the network.

To make full use of central control of SDN and its scalability of controller, we try to design an evaluation system to estimate the status of the network, including load of nodes, load of links, and scale of flows. For the best balancing of load in the network according the dynamic fluctuation of the flows, we presents adaptive heuristic online algorithms, with a module to collect real-time status information and another module to assess this status. With sophisticated design, the online algorithms are deployed as an application across the control plane, greatly improving the efficiency of flow forwarding.