Nimble：一种适用于OpenFlow网络的快速流调度策略

李 龙[1),2)] 付斌章[2)] 陈明宇[1),2)] 张立新[1),2)]

[1)](中国科学院大学 北京 100049)

[2)](中国科学院计算技术研究所计算机体系结构国家重点实验室 北京 100190)

摘 要 突发流量是导致网络拥塞和丢包的重要原因之一.减少网络拥塞的一种方法是在多条可达路径间均衡网络流量,如等价多路径(Equal-Cost Multi-Path,ECMP) 路由.然而,大多数等价多路径路由或者静态地将不同的流/数据包哈希到不同的路径,或者依赖于局部的/过时的路径状态信息.OpenFlow 技术利用集中式控制器控制网络行为,为控制器根据全局网络状态信息进行动态的数据流优化提供了可能.然而,采用基于轮询的网络状态探测机制在处理突发流量问题上面临诸多困难.文中提出一种用于 OpenFlow 网络的快速流调度策略,称为 Nimble.Nimble 架构扩展了 OpenFlow 协议的 packet-in 消息,由网络设备自主监测设备状态,并在网络出现拥塞时通过扩展的 packet-in 消息主动向控制器通告拥塞信息.模拟结果显示 Nimble 策略能够以近于零的时延检测网络链路拥塞,从而有效提高网络性能.

关键词 数据中心网络;OpenFlow;流调度;负载均衡

中图法分类号 TP393 **DOI 号** 10.3724/SP.J.1016.2015.01056

Nimble: A Fast Flow Scheduling Strategy for OpenFlow Networks

LI Long[1),2)] FU Bin-Zhang[2)] CHEN Ming-Yu[1),2)] ZHANG Li-Xin[1),2)]

[1)](*University of Chinese Academy of Sciences, Beijing* 100049)

[2)](*State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190)

Abstract Bursty traffic is one of the most important reasons that cause network congestion and packet loss. One way to reduce network congestion is to load balance traffic among multiple paths, such as ECMP (Equal-Cost Multi-Path) routing. However, most of ECMP routing algorithms either statically hash different flows/packets to separate paths, or depend on local/stale path load information. OpenFlow provides a new possibility to dynamically schedule flows according to global network status using centralized controllers. However, the poll-based mechanism to sense network status makes it hard to handle bursty traffic. In this paper, we propose a fast flow scheduling strategy for OpenFlow networks, namely the Nimble. The Nimble detects congestion by switches themselves, and exploits the extended packet-in message to notify the controllers as soon as congestion occurs. Simulation results show that the Nimble strategy could detect link congestion at nearly zero delay and could significantly improve network performance.

Keywords data center network; OpenFlow; flow scheduling; load balancing

---

收稿日期:2013-12-21;最终修改稿收到日期:2014-11-05.本课题得到国家自然科学基金(61221062,61331008,61202056)和中国科学院战略性先导科技专项(XDA06010401)资助.**李 龙**,男,1987 年生,博士研究生,中国计算机学会(CCF)会员,主要研究方向为数据中心网络中的服务质量保证、软件定义网络.E-mail:lilong_lee@ict.ac.cn.**付斌章**,男,1982 年生,博士,副研究员,中国计算机学会(CCF)会员,主要研究方向为高性能和高可靠性互连网络.**陈明宇**,男,1972 年生,博士,研究员,中国计算机学会(CCF)会员,主要研究领域为高性能计算机体系结构、操作系统、高性能算法优化.**张立新**,男,1971 年生,博士,研究员,中国计算机学会(CCF)会员,主要研究领域为数据中心系统、内存系统、体系结构模拟器、并行计算、性能评估和负载特性等.

1 引　言

近年来，以 Web 搜索、社交网络、电子商务、云存储和云计算等为代表的网络应用正处于高速发展阶段. 网络应用的发展、壮大推动网络应用服务提供商在全球范围内建立数量众多的数据中心. 当今，数据中心中通常包含数百万服务器节点[1]，并且通过数据中心网络实现服务器节点间的通信.

为了高效地互连数量庞大的服务器节点，研究人员提出了许多新的网络架构，如 FatTree[1]、Dragonfly[2]、BCube[3]、Jellyfish[4]、VL2[5] 以及 DCell[6] 等. 这些网络架构在网络核心提供多条可达路径. 也就是说，服务器通信对间存在多条可达路径. 利用网络架构的多路径，数据中心网络管理员通过在多条可达路径间均衡网络负载的方式提高网络性能.

按照路由算法在进行路由决策时是否考虑当前的网络状态，多路径路由算法可以分为两大类：无类路由算法（oblivious routing）和自适应路由算法（adaptive routing）. 无类路由算法在进行路由决策时不考虑当前的网络状态，如最短路径算法等. 与之相反，自适应路由算法在对数据包进行路由决策时，会根据当前的网络状态动态地调整数据包的路由路径[7].

由于大多数多路径路由算法对数据包的包头信息执行静态的哈希计算，并根据计算出的哈希值进行路由选择①，而不考虑路由决策时的网络状态信息，所以此类路由算法属于无类路由算法. 例如，等价多路径路由协议对数据包的包头信息（源 IP 地址、目的 IP 地址、源端口号、目的端口号和协议类型）执行哈希计算，并将计算得到的哈希值作为输出端口的索引进行路由选择. 对于均匀流量（uniform traffic），无类路由可以充分利用网络流量的均衡特性取得较好的网络性能. 然而，对于非均匀流量，由于网络流量在不同路径上的非均匀分布，无类路由可能导致严重的网络性能降低. 例如，将网络中的多个大流（elephant flow）路由到相交的路径上，那么同一链路上的大流冲突会导致网络性能急剧下降[8].

为了解决无类路由导致的性能损失问题，研究人员提出最小负载路由. 最小负载路由算法在进行路由决策时，首先检测网络设备各输出端口的负载信息，并将数据包通过负载最小的端口转发. 因为最小负载路由根据当前的网络状态进行路由决策，所以最小负载路由属于自适应路由的范畴. 然而，由于网络流量的突发特性，精确地获取每条路径的最新负载信息几乎是不可实现的. 因此，自适应路由算法通常利用局部负载信息进行路由决策. 但是，利用局部负载信息所做的路由决策通常会导致全局网络流量分布不均衡.

与标准网络协议（如等价多路径路由协议 ECMP）不同，软件定义网络为该问题提供了新的解决途径. 在软件定义网络中，网络行为由网络控制器集中控制. 网络控制器根据全局网络状态信息调度数据流，以达到优化网络性能的目的. 基于软件定义网络，UCSD 的研究人员提出 Hedera 网络架构[8]. Hedera 利用集中控制器检测网络中出现的大流，同时评估其带宽需求，并将大流重新调度到满足其带宽需求的低负载链路. 为了降低大流检测的开销，Hedera 采用基于轮询的检测策略. 例如，Hedera 控制器每 5 s 向网络设备发送一次请求消息，并接收来自网络设备的响应. 然而，由于周期性的拥塞检测方案在检测链路拥塞时存在时延，所以其检测由突发流量导致的拥塞时效率低下：（1）当网络中发生链路拥塞时，由于尚未达到检测周期，所以一直延迟到检测周期到达时才能检测到链路拥塞；（2）由于网络流量的突发特性，当到达拥塞检测周期时，链路的拥塞状态可能已经发生改变.

为了解决上述基于周期性检测的策略在应对突发流量时的低效性问题，本文提出一种适用于 OpenFlow（软件定义网络的一种典型实现）网络的快速流调度策略，称为 Nimble. Nimble 采用基于陷入的方式由交换机②主动通告拥塞信息给控制器. Nimble 架构中，由交换机主动检测链路是否发生拥塞. 当交换机检测到网络链路拥塞时，交换机通过 OpenFlow 协议中的 packet-in 消息向控制器发送拥塞通告. 控制器接收该消息，并解析消息得到以下信息：（1）拥塞发生的地点；（2）导致拥塞的网络流. 通过交换机自主检测拥塞并主动通告拥塞的方式，Nimble 可以快速检测网络拥塞，使其成为解决网络拥塞，尤其是由突发流量导致的拥塞的有效方案. Nimble 架构扩展了交换机的基本功能，以支持交换机端的拥塞检测. 同时，Nimble 通过扩展 OpenFlow 协议消息的方式实现由交换机向控制器的拥塞信息通告. Nimble 架构中，交换机的每个输出端口队列

① Analysis of an Equal-Cost Multi-path Algorithm（AECMA）. http://tools.ietf.org/html/rfc2992 2012,6,21

② 软件定义网络中，交换机可以实现二层交换功能和三层路由功能，所以本文中不加区别的使用路由器和交换机.

关联一个计数器,用于记录当前的队列长度.当计数器值超过某个既定阈值时,交换机产生中断.该中断触发拥塞处理例程.拥塞处理例程通过扩展的OpenFlow消息向控制器通告拥塞信息.控制器接收源自交换机的消息,解析消息内容.当控制器解析该消息为拥塞通告消息时,控制器为拥塞链路上的大流计算低负载链路,并将其调度到上述低负载链路.

采用由交换机自主检测链路拥塞,并主动通告拥塞信息的方式,Nimble能够快速响应网络拥塞,并且有效地解决网络中由突发数据流引发的链路拥塞.

本文的主要贡献如下:

(1)提出一种快速解决网络拥塞的网络流调度策略;

(2)提出一种基于陷入的拥塞通知技术,该技术通过扩展的packet-in消息将拥塞信息通告给控制器;

(3)提出一种分布式的、完全可控的大流检测方案;

(4)提出一种保证路径更新一致性的方案.

本文第2节介绍与本文相关的背景知识;第3节描述Nimble总体架构并讨论Nimble的具体实现细节;第4节讨论Nimble实现过程中遇到的实践性问题;第5节对Nimble架构进行实验评估;第6节介绍本文的相关工作;第7节展望本文的未来工作;第8节总结全文.

2 背景介绍

随着网络技术的发展,网络设备复杂度急剧增加.这给网络的维护、管理和开发带来极大的挑战.软件定义网络的出现和发展,给网络操作人员带来新的曙光.同时,大数据以及新的应用框架的出现(如MapReduce)改变了数据中心网络的负载特征.本节,我们首先回顾软件定义网络以及OpenFlow的基本概念.随后,我们讨论数据中心网络负载的基本特征,并基于此阐述Nimble方案的可行性.

2.1 OpenFlow网络

随着网络技术的不断发展,网络设备需要完成越来越多的复杂功能,如防火墙、网络地址转换(NAT)、渗透检测等.这不仅使得网络设备变得越来越"臃肿",同时给网络的操作、维护和开发带来极大的挑战.首先,网络设备的复杂性增加了硬件设计的复杂度以及网络操作人员的操作成本.其次,传统的网络设备采用闭源方式,不公开其操作系统API.网络设备的闭源性,使得网络开发人员不能按照自己的意志"随心所欲"地控制网络行为.最后,不同的网络设备提供商提供互不兼容的操作命令接口,增加了网络操作人员的学习复杂度.软件定义网络(Software-Defined Networking,SDN)采用控制平面(control plane)和数据平面(data plane)相分离的实现方案,给用户开发自定义网络控制程序提供了足够的灵活性和便利性,并且有效地降低了网络设备开发以及网络维护的难度.这种灵活性和便利性促使越来越多的研究人员参与到软件定义网络的研究领域.OpenFlow网络[9],是由斯坦福大学提出的一种软件定义网络典型实现方案.OpenFlow网络由一个或多个控制器以及大量的OpenFlow交换机组成.OpenFlow网络中,交换机仅实现转发平面的功能,同时将复杂的控制功能转移到软件控制器实现,极大地简化了设备实现的复杂度.由于OpenFlow网络中,网络的控制功能由软件实现,所以可以采用各种成熟的软件工程学方法进行软件开发,从而简化网络开发、管理的复杂度.同时,OpenFlow网络采用OpenFlow协议实现控制器和交换机之间的交互,为上层网络开发人员提供统一的编程接口.

根据OpenFlow交换机规范①,OpenFlow交换机由流表、安全信道和OpenFlow协议3个主要部分组成.图1展示了OpenFlow网络的基本组成.OpenFlow控制器用于实现管理网络行为的各种应用,如拓扑收集、路由、防火墙、NAT等.OpenFlow交换机的软件层主要实现安全信道和OpenFlow协议.OpenFlow控制器使用OpenFlow协议并通过安全信道同交换机交互信息.交换机硬件主要实现数据包转发.OpenFlow交换机的流表由流表项组成.每个流表项包含头域、相关的计数器以及作用于匹配该流表项的数据包的动作.如图1所示,头域为十元组信息②,用于匹配特定的数据包.流表项中的计数器统计匹配该表项的数据包数目以及总字节数.操作域指示作用于匹配该表项的数据包的行为,如转发数据包到指定端口、转发数据包到控制器、丢弃数据包、按照常规二层交换机功能处理等等.控制器通过增加、修改、删除流表项控制交换机的行为.

① OpenFlow Switch Specification 1.3.1 (OFSS). https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf, 2013,6,21

② OpenFlow协议处在发展阶段,新版本中的头域信息可能发生变化.

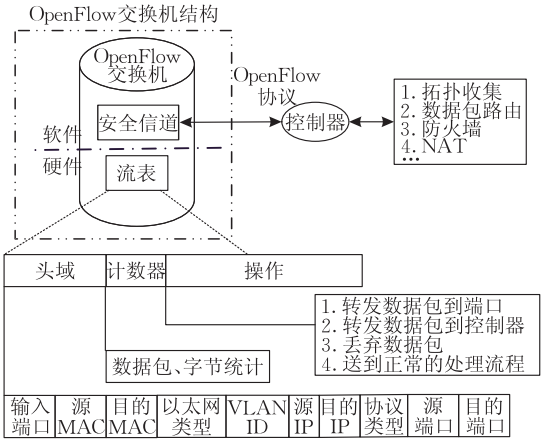

图 1　OpenFlow 基本组成图

OpenFlow 协议支持 3 种类型的消息：(1) 控制器到交换机的消息；(2) 异步消息；(3) 对称消息. 控制器到交换机的消息是由控制器发送到交换机的消息，通常用于查询 OpenFlow 交换机的状态信息. 异步消息由 OpenFlow 交换机发送到控制器，主要用于更新交换机的状态信息，如交换机端口状态改变消息等. 对称消息可以由 OpenFlow 交换机或者控制器发出. 控制器通过 OpenFlow 协议发送消息，从而实现控制交换机行为的目的. 这种由控制器控制整个网络行为的结构使得网络的管理、控制变得更加简单、灵活.

2.2　数据中心流量特征

通过对真实数据中心网络流量的分析，Greenberg 等人[5]和 McKeown 等人[9]的研究，均表明数据中心流量同传统的局域网和广域网流量有着极大的不同. 不同于传统的南北流量模式，数据中心网络中，内部节点间的东西流量(数据中心内部节点间的通信流量)，在数据中心网络所有流量中占主导地位. Greenberg 等人[5]指出，数据中心中 80% 的数据流量为数据中心内部的流量. 为了满足数据中心内急剧增长的东西流量需求，当今的数据中心网络，如胖树(FatTree)网络[1]，通常在网络拓扑中为通信节点对提供多条可达路径，以增加网络的对剖带宽(bisectional bandwidth)，提高网络吞吐量.

不幸的是，尽管数据中心网络采用多路径的方法提高网络性能，但是由于缺乏有效的多路径路由协议，数据中心网络中仍然存在链路拥塞，特别是网络的核心层链路[10]. 然而，尽管一些核心层链路拥塞严重，网络中仍然存在大量空闲的核心层链路. 例如，Benson 等人[10]指出，75% 的核心层链路处于未充分利用状态. 上述事实表明：(1) 解决数据中心网络中的拥塞问题仍然十分重要；(2) 数据中心网络中存在大量低负载链路可以用来均衡网络负载，以提高数据中心网络的性能.

由于网络中存在大量空闲链路，所以，当网络出现拥塞时，通过将拥塞链路上的数据流重新调度到低负载链路，可以有效地解决链路拥塞问题. 基于上述观察，Nimble 通过重新调度拥塞链路上的数据流的方法解决网络拥塞问题.

2.3　大流标识

数据中心网络中，同时存在着大流和小流. 通常来讲，小流(如搜索引擎的查询请求等)包含的数据量小，带宽需求也相对较小. 然而，其对网络延迟较为敏感. 与之相反，大流(如文件备份、传输等)通常包含较大的数据量，并且对网络带宽有较高的需求. 然而，大流通常对网络延迟信息不敏感. 为了有效地均衡大小流对网络带宽和网络延迟的不同需求，需要进行大流检测.

为了检测网络中的大流，通常可以采用以下方案[11]：

(1) 基于采样的检测. 此方案中，集中式的大流检测程序周期性地向交换机发送请求信息，以获取交换机所有端口的采样信息. 采样信息可以通过简单网络管理协议(SNMP)获得. 采用此种方式，只需要采样部分数据包(如每 5 min 执行一次数据包采样)，并且只需要传输必要的包头信息. 当对某个数据流检测到足够的字节后，将该流标记为大流. 然而，基于采样的大流检测策略准确度较低. 同时，由于采用集中式的处理策略，大流检测的开销较大.

(2) 基于应用的检测. 由于开发者对应用程序的特征具有最为准确的理解，所以，基于应用的大流检测方案准确度最高. 然而，传统的应用自身并没有实现大流检测逻辑. 所以，为了实现基于应用的大流检测，需要对数据中心中部署的应用进行修改. 但是，数据中心中部署的应用种类繁多，并且许多应用无法获取其源代码. 所以，对应用进行修改以支持大流检测无法兼容大量现存应用.

(3) 基于统计的检测. 不同于基于采样的大流检测方案，基于统计的检测方案为每个数据流维护精确的数据统计. 该种策略下，控制器周期性地向边缘层交换机发送数据流统计请求，并接收来自交换机的响应，从而获取各条数据流的精确数据统计. Hedera[8]是基于统计的大流检测实例之一. 然而，基于统计的大流检测策略采用集中式方案，无法扩展到大规模网络. 同时，基于统计的检测方案，需要交换机为经由该设备的每条数据流维护一条表项，

记录该流的数据统计值.但是,流表表项是交换机的稀缺资源,为每个数据流记录统计信息需要消耗大量的交换机资源.另外,采用基于统计的大流检测策略,交换机和控制器间产生大量的请求/响应消息.在带宽受限的网络中,这需要消耗大量的带宽资源,进而加剧网络拥塞.最后,基于统计的大流检测策略只有到达检测周期时才能检测到大流,这会增加大流检测的时延.

3 Nimble 架构

基于轮询的拥塞检测机制,周期性地向交换机发送统计请求信息,并根据收到的响应消息计算链路负载.然而,基于轮询的拥塞检测策略在检测拥塞时存在时延.与之不同,Nimble 架构由交换机自主检测拥塞,并在检测到拥塞时主动通告控制器.控制器接收到拥塞通告后,将拥塞链路上的数据流调度到低负载链路.

本节,我们首先介绍 Nimble 的总体结构.然后,讨论 Nimble 的具体实现细节.

3.1 总体结构

图 2 展示了 Nimble 的总体结构图.如图所示,Nimble 主要包括三个模块:拥塞通知模块、网络状态维护模块(A)和调度模块(B).拥塞通知模块的主要功能是检测网络中的拥塞状况,并将拥塞信息通告给网络控制器.Nimble 架构中,拥塞通知模块实现在 OpenFlow 交换机中(图中没有单独列出).不同于周期性轮询的拥塞检测机制,Nimble 使用基于

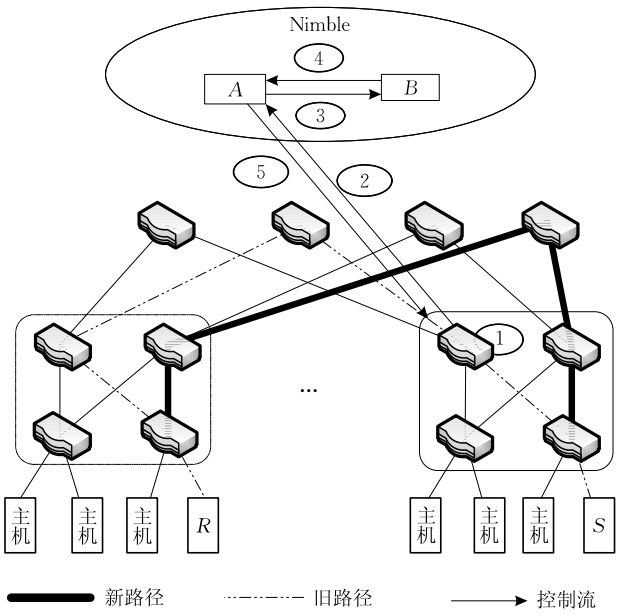

图 2 Nimble 总体架构图

陷入的拥塞检测策略检测拥塞信息.一旦检测到链路拥塞,拥塞通知模块立即将拥塞信息通告给网络控制器.拥塞通告通过 Nimble 中扩展的 OpenFlow 协议消息实现.OpenFlow 控制器维护当前网络的所有状态信息,如网络拓扑、物理链路利用率等.状态信息的更新和维护由位于控制器中的网络状态维护模块完成.

接收到拥塞通知消息后,网络状态维护模块更新其自身的状态信息,并将拥塞信息转发到调度模块.调度模块从网络状态维护模块中查询网络状态信息,获取拥塞链路上的大流信息,并为这些大流计算低负载路径.最后,调度模块通过 OpenFlow 协议修改交换机表项,将新的路由路径分发到相应的交换机.之后,拥塞数据流的后续数据包将按照新的路径路由.

为了提高网络性能,研究人员提出许多具有多路径的网络拓扑.数据中心采用此类拓扑提供更高的网络对剖带宽(bisectional bandwidth).为了简化设计,本文将 Nimble 实现于胖树拓扑.图 3 给出了一个具有 16 台服务器节点的胖树拓扑实例.胖树拓扑中,网络设备可以划分为不同的 Pod.图 3 中将汇聚层和边缘层交换机划分为 4 个 Pod.Nimble 架构中,网络中所有的交换机均兼容 OpenFlow 功能.Nimble 架构实现了 OpenFlow 控制器,以实现 Nimble 的管理功能.该控制器可以运行于一台独立于网络拓扑的服务器节点中,也可以运行于网络拓扑中的某台服务器节点或者虚拟机中.控制器维护当前网络的所有信息,比如网络拓扑结构、链路容量、数据流路由路径以及链路利用率等.另外,控制器决定数据包的路由路径.

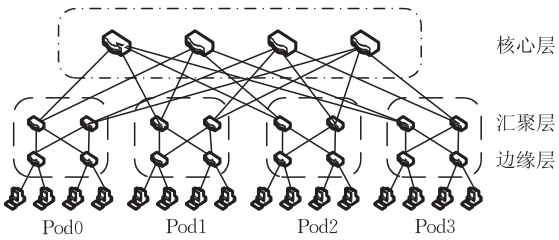

图 3 包含 16 台服务器节点的胖树拓扑

对于使用具有 K 个端口的交换机构成的胖树拓扑,网络中包含 $K^2/4$ 台核心交换机[1].胖树拓扑中,任意位于不同 Pod 的源、目的服务器节点间存在 $K^2/4$ 条可达路径.每条路径经由唯一一台核心层交换机.所以,在胖树拓扑中,任意通信节点对间的路径可以通过简单的计算获得,因为一旦确定了

核心层交换机，通信对间的路径便是确定的(5.1 节评估了胖树拓扑的路径计算开销). 当网络规模过大时(如数十万服务器节点)，网络的性能受限于单个控制器的处理能力. 考虑到扩展性问题，Nimble 架构中的 OpenFlow 控制器可以采用分布式策略实现，如 HyperFlow[12] 和 Onix[13] 等.

3.2　拥塞通知

为了实现拥塞通告，首要的任务是检测网络拥塞. 传统的网络协议，如传输层控制协议(TCP)，利用终端节点检测拥塞信息. 这种拥塞检测方案存在如下缺点：当终端节点检测到拥塞时，网络中已经出现拥塞，甚至是数据包丢失. 更为糟糕的是，网络流量的突发特性导致当终端节点检测到链路拥塞时，链路状态可能已经发生改变. DCTCP[14] 在交换机出口队列长度超过某个阈值(比如说，队列容量的 80%)时提前预测拥塞. Nimble 采用同样的方式执行拥塞预测，如过程 1 所示. 基于阈值检测链路拥塞的优势是，交换机不需要在每个数据包到来时都执行拥塞检测. 当某个队列的长度超过给定阈值时，交换机将此信息通告给控制器. 控制器接收到来自交换机的通告后，即可获知在该交换机的某个端口上发生了拥塞. 拥塞通告消息中包含导致拥塞的端口信息，控制器可以根据发生拥塞的端口信息计算出发生拥塞的链路. 由于网络控制器维护了网络的所有信息，控制器可以推测出经过该拥塞链路的所有数据流. 为了实现基于阈值的拥塞通告机制，Nimble 扩展了 OpenFlow 标准交换机的功能. Nimble 架构中，为交换机的每个物理端口队列关联一个计数器. 如过程 1 所示，当数据包进入交换机的某个端口队列时，交换机依据数据包的大小更新计数器值. 当交换机检测到队列计数值超过给定阈值时，产生拥塞信号. 拥塞信号触发交换机发送本文扩展的 OpenFlow 协议消息向控制器通告拥塞信息.

过程 1.　交换机拥塞通知流程.

数据包 P 进交换机端口队列 M

//判断数据包进队列后队列长度是否超过阈值

//*size*( )函数求得数据包的字节总数

//*CurrentCount* 表示当前的队列计数器值

CurrentCount+=*size*(*P*)；

IF *CurrentCount*>=*Threshold*

　　//产生拥塞通知消息

　　Message=*GenerateMessage*()；

　　//发送拥塞消息到控制器

　　SendMessage (*Message*)；

　　RETURN；

OpenFlow 协议中，没有向控制器通告拥塞信息的消息类型. 一种简单的拥塞信息通告实现方式是定义一种新的交换机和控制器间的消息类型. 然而，定义新的消息类型需要 OpenFlow 控制器开发人员更新控制器程序以支持新定义的消息类型. 鉴于此，Nimble 通过扩展 OpenFlow 协议消息的方式实现拥塞信息通告. OpenFlow 规范中，存在由交换机产生并发送到控制器的称为 OFPT_PACKET_IN 的消息(以下简称 packet-in 消息). Packet-in 消息用于由交换机向控制器通告消息，比如流表项匹配缺失等. OFPT_PACKET_IN 结构体中，reason 域用来指示发送消息的原因. OpenFlow 协议中，reason 域可以取以下两种值：(1) OFPR_NO_MATCH；(2) OFPR_ACTION. 概括地讲，OFPR_NO_MATCH 值表明数据包到达交换机，但是交换机的流表中没有可以同该数据包相匹配的流表项. OFPR_ACTION 值表明流表项的操作域指示将该数据包发送到控制器.

本文中，Nimble 通过在 reason 域中定义新的取值 OFPR_CONGESTION 的方式扩展标准的 packet-in 消息. 图 4 显示了扩展的 packet-in 消息的 reason 域. 当交换机检测到拥塞时，其发送 packet-in 消息给控制器. 该消息的 reason 域赋值为 OFPR_CONGESTION，表明发送该消息的原因是链路发生拥塞. 控制器接收到该消息后，通过解析消息的 reason 域，可以获知网络中相应的链路出现拥塞. 之后，控制器可以根据其维护的网络信息推测出该拥塞的所有相关信息，包括拥塞发生的地点以及该拥塞链路上的数据流等等. 采用扩展协议的方式，只需要更改拥塞处理程序，增加解析、处理该 reason 域的逻辑，而不需要对控制器底层代码进行修改.

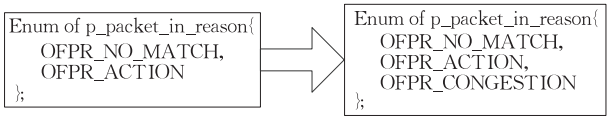

图 4　扩展的 packet-in 消息 reason 域

3.3　流调度

如 3.1 节所述，对于包含 K 个 Pod 的胖树网络，网络中存在 $K^2/4$ 台核心交换机. 其中，每个核心层交换机对应于服务器节点对间的一条通信路径. 也就是说，对于有 K 个 pod 的胖树拓扑，不同的通信节点对间存在 $K^2/4$ 条可达路径(针对源、目的节点位于不同 Pod 的通信对). 胖树网络的路径多样性，给我们将拥塞链路上的数据流重新调度到新

的低负载链路提供了理论支持.

为了实现 Nimble 的拥塞数据流重调度功能,需要解决两大主要难题:(1)拥塞发生时对哪条数据流进行重新调度;(2)如何调度这些数据流.考虑到数据中心网络中存在大量的网络数据流(如数以百万计的数据流),对拥塞链路上的所有数据流进行重调度的开销极大.Curtis 等人[11]在 Mahout 中指出,一种可行的方案是只对数据中心中的大流进行重调度.Hedera 将发送数据量超过链路容量 10% 的数据流认为是大流.Hedera 架构中,控制器周期性地向边缘层交换机发送流统计请求,并从交换机接收响应.通过解析响应消息,控制器可以获得边缘交换机中某条流发送的数据量信息,从而检测出大流.不同于 Hedera,Mahout 通过检测终端节点 TCP 缓存的方式检测大流[11].本文中,我们采取 3.4 节所述的方式检测大流.检测到大流后,下一步工作是为这些大流选择新的低负载链路.Nimble 采用全局最先适应(Global First Fit)算法[8]解决大流重调度问题.如算法 1 所示,全局最先适应算法通过迭代源、目的通信对间的所有路径以寻找能够满足数据流带宽需求的路径.如果找到满足需求的路径,数据流采用该路径作为新的传输路径.否则,数据流仍然在原始路径路由.在采用新路径的情形下,控制器将新的路由路径安装到相应交换机的流表中.Nimble 通过 4.1 节中描述的方案解决路径一致性问题.

算法 1. 全局最先适应算法.

```
输入:网络拓扑 Topo 以及数据流 F
输出:满足流 F 需求的路径 P
FOR ALL switch 属于核心层交换机
  //利用 switch 计算源、目的节点对间的路径
  path=computPath(switch, Topo, F);
  Found=True;
  FOR ALL link 属于路径 path
    IF link.bandwidth<F.bandwidth
      Found=FALSE;
      BREAK;
  IF(Found)
    P=path;
    RETURN P;
RETURN NULL;
```

3.4 大流检测

如 3.3 节所述,可行的流调度方案是仅对网络中的大流进行重调度.这就要求 Nimble 能够有效地检测大流.集中式大流检测方案实现简单,但是当网络规模较大时,其扩展性较差.现有的体系中,数据中心通常为单个组织所拥有.所以,数据中心管理员对其内部采用的操作系统/虚拟机管理软件(Hypervisor,如 Xen、KVM 等)拥有完全的自主控制权.为了有效地利用资源,当今的数据中心通常采用虚拟化技术提高资源利用率.采用虚拟化技术后,用户可以自主选择其运行系统的协议栈版本,甚至是自主选择其所运行的操作系统.所以,采用在操作系统或者利用协议栈检测大流的方式,恶意用户可以改变大流检测的原有行为.

考虑到以上因素,本文提出一种新的大流检测方案.该方案中,大流检测逻辑作为虚拟机管理软件(Hypervisor)的模块实现.对于不采用虚拟技术的数据中心,该大流检测逻辑可以作为操作系统的内核模块实现.由于数据中心所有者对其所采用的虚拟机管理软件有着完全的控制权,所以,将大流检测逻辑实现于虚拟机管理软件可以保证数据中心所有者对大流检测的完全控制,以避免恶意用户的破坏.同时,采用分布式的实现方案,保证其可以扩展到具有数十万台服务器节点的网络.

本文采用的大流检测算法如算法 2 所示.

算法 2. 大流检测算法.

```
输入:数据包 pkt
输出:数据包 pkt 所属的流是否为大流
ret=FALSE;
bucket=hashValue(pkt);
flow=findIndex(bucket, pkt);
flow->bytes+=size(pkt);
IF flow->bytes>=THRESHOLD
    ret=TRUE;
now=kernel_now();
IF now-flow->time>=dt
    flow->time=now;
    flow->bytes=0;
RETURN ret;
```

如算法 2 所示,对于接收到的数据包,大流检测算法计算该数据包的哈希值,并根据计算得到的哈希值在对应的哈希链中查找数据包所属的数据流信息.之后,更新该流收到的字节计数值.如果该周期内,该流发送的字节总数超过给定阈值 $THRESHOLD$,则标记该流为大流,并设置函数的返回值为真.最后,获取系统的当前时间,并将当前时间和该流的检测周期起始时间进行比较,若当前时间和该流的周期起始时间之间的差值大于既定值 dt,则将该流的周期起始时间设为当前时间,并清空该流的字节统计.

终端节点检测到大流后，需要将大流信息通告给控制器. 一种简单的实现方式是由终端节点直接将大流信息通告给控制器. 大流通告信息中，只需要包含能够识别大流的必要信息（比如，如果使用数据包的五元组信息标识数据流，则只需将标识该流的五元组信息发送给控制器）. 然而，终端节点将大流直接通告给控制器的方式，面临以下问题：(1) 破坏了控制器只和交换机交互的工作方式；(2) 在某些网络中，终端节点可能和控制器之间没有逻辑相连链路. 为了解决以上两个问题，本文采用了 Curtis 等人[11]提出的方案，由 OpenFlow 交换机向控制器通告大流信息. 该方式下，缺省地为所有边缘层交换机安装具有最低优先级的默认表项，如图 5 所示. Curtis 等人的方案通过匹配 IP 包头中的 tos 域检测大流通告数据包. 与之不同，本文通过匹配数据包的源端口是否为零检测大流通告数据包. 根据 OpenFlow 规范说明，tos 域并不是 OpenFlow 流表必须支持的匹配项. 为了达到普遍适用性的目的，本文中采用匹配源端口是否为零的方式检测大流通告数据包，如图 5 所示.

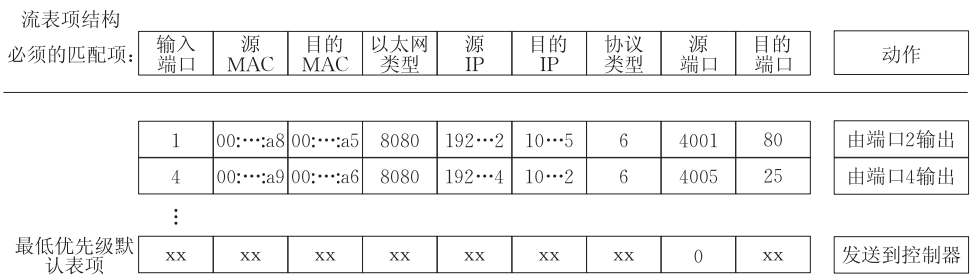

图 5　大流检测表项实例（图中 xx 表示该域可以匹配任意值）

当终端通过大流检测算法检测到某条流为大流时，大流检测模块构造一个新的数据包，并将其发送到与之相连的边缘层交换机. 该数据包的源端口号置为零. 由于零值源端口号为保留值，所以，网络中不会出现源端口号为零的正常数据包. 为了能够得到该数据流的标识信息，数据包负载中包含标识该流的完备信息（如该流的五元组信息）. 交换机匹配该数据包后，按照该表项对应的操作域信息，将数据包发送到控制器. 控制器接收到该数据包后，解析数据包，并从数据包的负载信息中解析该流的完整标识信息. 进而，控制器可以检测出网络中的大流.

4　实践分析

除了上述 Nimble 实现的技术方案外，Nimble 实现过程中还存在一些实践性问题，包括路径更新一致性问题和可行性分析. 本节，我们首先分析 Nimble 实现过程中遇到的这些实践性问题. 然后，在本节结束，我们将通过一个具体的工作实例，更加清晰的展示 Nimble 处理链路拥塞的流程.

4.1　路径更新

如 3.1 节所述，当网络中出现拥塞时，Nimble 控制器需要对拥塞链路上的大流进行重新的路由计算，并将新的路由路径安装到相应的交换机流表中. 然而，控制器对交换机流表的修改是异步的，即控制器只负责发出流表修改消息，并不保证流表信息已经被交换机正确处理. 所以在交换机流表更新过程中，可能会产生流表项重复缺失的问题.

如图 6 所示，路径 S-1-2-3-4-5-D 为原始路径，路径 S-1-6-7-8-5-D 为更新后的路径. 假设新路径计算出来后，控制器按照 1-6-7-8 的顺序安装路由表项. 当交换机和控制器之间的延迟较大时，控制器首先更新交换机 1 上的流表信息，将流的下一跳更改为交换机 6. 之后，数据包被路由到交换机 6，但此时交换机 6 上的流表信息还没有得到及时的更新，所以产生流表项缺失. 这会触发交换机 6 向控制器询问路由信息. 同样地，交换机 7、8 也会向控制器询问该流的路由信息，从而产生重复的表项缺失. 更为糟糕的是，如果交换机 6 上存在该流的“古老”的路径信息，该流会按照交换机 6 上的“古老”流表信息进行路由，从而将数据包路由到错误的路径.

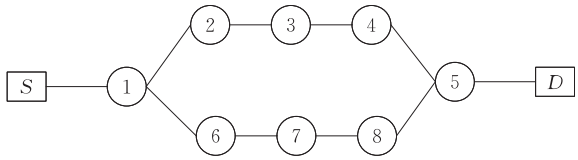

图 6　路径更新问题实例

为了解决路由更新的一致性问题，本文提出一种简单的逆向路径更新方案. 逆向路径更新方案的基本过程描述如下：首先寻找两条路径之间不相交的交换机；其后，按照由目的节点到源节点的逆向顺序向新路径上的交换机安装路由信息. 以图 6 为例，

源节点 S 和目的节点 D 间的新、老路径间的不相交交换机为 2-3-4 和 6-7-8. 按照逆向路径更新方案，依次在交换机 8、7、6 上安装路由信息. 最后，修改交换机 1 中的表项信息，将流的下一跳交换机更改为 6. 之后，数据包到达交换机 1 时，按照新的流表信息，数据包被转发到交换机 6. 由于后继交换机 7、8 上已经安装了最新的路由信息，该流将按照新的路由路径转发.

4.2 可行性分析

Nimble 架构采用基于阈值的方式预测链路拥塞，下面我们分析这种方案在现有网络设备上实现的可行性.

随机早期检测[15] (Random Early Detection, RED)是一种主动队列管理(Active Queue Management，AMQ)机制. 简单地讲，随机早期检测方案中，交换机为每个端口队列关联两个阈值：*Thres*1 和 *Thres*2. 交换机动态地计算队列的平均长度，当交换机平均队列长度超过 *Thres*1 时，交换机按照一定的概率丢弃到达该队列的数据包. 当交换机平均队列长度超过 *Thres*2 时，所有到达该队列的数据包都将被丢弃.

与之不同，Nimble 架构只采用了一个阈值. 另外，不同于随机早期检测方案，Nimble 架构采用的是即时队列长度[14]，而不是平均队列长度. 随机早期检测方案采用式(1)计算队列的平均长度. 其中 W_q 为公式计算的权重，q 为监测的当前队列长度，$avgQ$ 为计算得到的平均队列长度. 由式(1)可以得知，将随机早期检测方案中的权重值设置为 1，则为计算队列即时长度.

$$avgQ=(1-W_q)\times avgQ+W_q\times q \tag{1}$$

综上所述，Nimble 策略在当今交换机的工程实现中，是可行的.

4.3 工作实例

本节中，我们将以一个具体的实例描述 Nimble 的工作过程. 如图 2 所示，假设服务器节点 S 和服务器节点 R 之间存在一条大流 F. 流 F 的当前路径为 P ，如图中点划线所示. Nimble 的拥塞控制流程如下所示：

(1)时刻 T，属于流 F 的数据包到达标记为 1 的交换机. 数据包进队列后，交换机检测到对应的队列长度超过指定的阈值；

(2)交换机通过 3.2 节描述的拥塞通知模块给控制器发送消息，通告拥塞的发生；

(3)网络状态维护模块根据上述收到的拥塞通告消息更新自身的信息. 同时，网络状态维护模块通过 3.4 节描述的机制，检测流 F 为大流，并调用调度模块；

(4)调度模块为流 F 重新计算一条新的路径 P' (图中粗实线所示)，并通告网络状态维护模块；

(5)Nimble 控制器在所有相关的交换机中安装新的流表项.

经过上述所有操作之后，该数据包以及流 F 的所有后续数据包都将经过新路径 P' 路由.

5 实验评估

由于缺乏物理 OpenFlow 交换机，并且需要对交换机硬件和协议进行拓展，以支持 Nimble 的功能，所以本文使用 NS3 模拟器对 Nimble 方案进行模拟评估. 本节中，我们阐述本文实验所采用的实验环境以及相应的实验结果.

5.1 路径计算开销

当网络出现拥塞并通告控制器后，控制器需要遍历对应节点对间的所有路径以找到满足通信带宽需求的路径. 然而，路径计算需要时间开销. 当采用诸如 Dijkstra 算法等图的路径计算算法时，路径计算的时间开销极大. 例如，在服务器节点数目为 27 648 的胖树拓扑中，采用 Dijkstra 算法计算 20 条路径的时间约为 60 s. 这将严重制约 Nimble 方案的可行性. 然而，如 3.3 节所述，由于胖树网络是规则网络，所以胖树网络中任意通信节点对间的路由路径可以通过简单的线性计算获得. 这样，在胖树网络中计算可行路径的开销极小，确保 Nimble 可以快速地响应网络拥塞并执行路径更新. 本节对胖树网络中路由路径计算的开销进行评估.

图 7 展示了胖树拓扑的路径计算开销. 横坐标为胖树网络中服务器节点的数目，纵坐标为路径计算的开销，时间为 μs. 图中，路径数目表示任意位于不同 pod 的节点对间的路径数目. 即核心层交换机的数目. 其中最好情况下的时间对应于第一条路径即满足数据流需求的情形；最坏情况的计算时间指的是需要迭代所有的可达路径才能找到满足数据流需求路径的计算时间. 从最好情况曲线可以看出，不论网络规模为多大，迭代单条路径的时间开销是相同的(实验中约为 20 μs). 同时，由于迭代单条路径的开销是相当的，所以，路径迭代的开销正比于所要迭代的路径数目. 最后，从图中我们可以看出，在胖树网络中，计算路由路径的开销是极小的(除了

27 648 台服务器节点的最坏情形外，路径可以在微秒级时间内计算得到）. 胖树拓扑路径计算的极小开销，保证 Nimble 可以快速地处理网络拥塞.

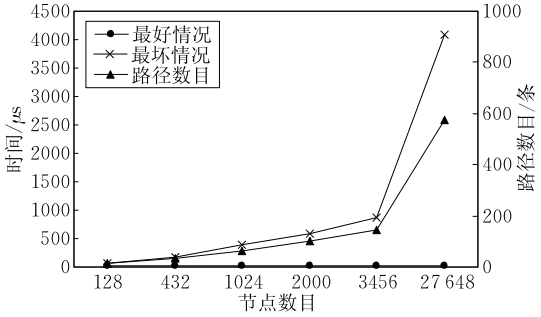

图 7　路径计算开销

5.2　实验设置

本节通过模拟实验比较 Nimble 同 ECMP 以及 Hedera 的性能. 本文中所有的实验结果均由网络模拟器 NS3[16] 获得. NS3 中将 OpenFlow 协议作为单独的模块集成到模拟器中，集成协议版本为 0.8.9. 尽管当前最新的 OpenFlow 协议版本为 1.3.1，OpenFlow 协议的基本功能没有改变. 实验中，我们使用胖树拓扑作为基准网络拓扑，交换机采用 16 端口交换机. 默认配置情形下，网络中包含 128 台服务器节点. 相比于真实的数据中心规模，实验模拟的网络规模相对较小. 之所以采用这种小规模的实验配置，主要有以下原因：首先，NS3 模拟器为数据包级模拟器，因此，对于大规模的网络模拟需要消耗非常长的时间（比如，对于我们的小规模实验，需要超过 24 h 的墙上时间来模拟 20 s 的模拟时间）；其次，模拟器主要用来验证 Nimble 架构是否能够快速有效地处理突发数据流导致的链路拥塞，因此，上述实验规模已经足够用来验证突发数据流导致的拥塞问题. 实验中，我们使用 NS3 环境中的一个或多个应用来模拟服务器节点. 模拟拓扑下所有链路的带宽均配置为 1000 Mbps.

为了聚焦由突发流量导致的拥塞问题，本文中模拟实验采用合成流量驱动模拟器执行. 由于没有公开可以得到的真实数据中心网络 trace 及对应的拓扑说明，本文中没有使用真实的数据中心网络 trace 进行实验验证. 实验采用的流量模式如下：（1）每个服务器节点和具有固定偏移的服务器节点建立 TCP 连接（实验中，偏移值为 16. 采用此偏移值，所有的目的服务器节点和源服务器节点位于不同的 pod 中）作为背景数据流. 背景数据流的速率为 5 Mbps；（2）除了背景数据流，另外的 35 台服务器节点向其目的服务器节点发送块数据. 实验中，数据块的大小为 125 MB. 为了测试 ECMP、Hedera 以及 Nimble 的性能，实验中，分别在 4 s、8 s 和 12 s 发送块数据.

5.3　模拟实验结果

本实验，我们比较了 ECMP、Hedera 以及 Nimble 的网络性能. 实验结果如图 8 所示. 图中，横坐标代表模拟时间，间隔为 0.1 s. 纵坐标代表网络的聚合吞吐量（由于每个数据块发送周期的网络行为类似，图中只列出了两个数据块发送周期对应的情形）.

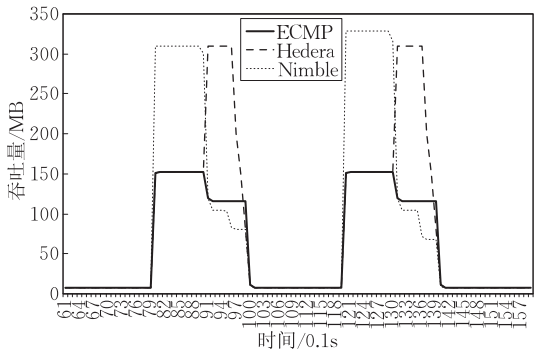

图 8　Nimble 同 ECMP、Hedera 的性能比较

由实验结果，我们可以得到以下结论：

（1）由于存在大流冲突，采用 ECMP 方案网络性能较差；

（2）尽管 Hedera 和 Nimble 都可以取得比 ECMP 更高的网络聚合吞吐量，然而同 Nimble 相比，Hedera 需要经过一段时间的延迟才能响应拥塞以取得较高的聚合吞吐量. Hedera 的响应延迟，是由于 Hedera 采用的是基于轮询的周期性检测方案（实验中，检测周期为 4 s）. 尽管 Hedera 可以采用更小的时间粒度作为检测周期，但是更小的时间粒度会急剧增加拥塞检测的开销；

（3）Nimble 采用交换机主动检测并通告拥塞的方式检测链路拥塞，极大的降低了拥塞检测和响应的时间. 由上述结果得出，Nimble 可以快速地响应网络拥塞并极大提高网络的聚合吞吐量.

注意到，在块数据传输的结束阶段，Nimble 的聚合吞吐量会呈现下降趋势，如图 8 所示. 造成这一现象的原因是，部分服务器节点已经完成数据通信，此时网络中只有部分服务器节点仍然在发送数据.

为了验证不同的队列阈值对网络性能的影响，我们采用不同的阈值进行了一系列实验，实验结果如图 9 所示. 采用带内通信的方式，大量的拥塞通告消息会加剧网络负载，进而降低网络性能. 即使使用带外通信，大量的拥塞通告消息也会极大地增加控制器的 CPU 负载.

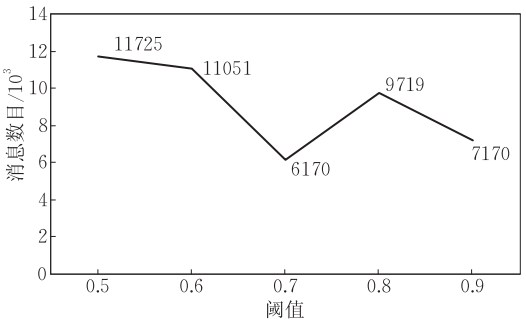

图 9 不同阈值下的消息数目

图 9 中,横坐标代表不同的队列长度阈值,纵坐标代表控制器从交换机收到的拥塞通知消息的数目.从图中可以得出,阈值为 0.7 时发送消息的数目最少.造成这一现象的主要原因如下:(1)当阈值较小时,队列长度很容易达到阈值,从而认为出现链路拥塞,致使交换机发送拥塞通告消息;(2)当阈值较大时,链路拥塞更为严重,需要调度链路上的更多数据流才能解决拥塞问题,进而产生更多的拥塞通告消息.模拟实验中,不同的阈值所获得的网络性能相近,故此不再单独列出.由上面分析可以得出,较大的阈值可以在取得较好的网络性能的同时降低拥塞通告消息的数目.根据实验结果,我们建议的阈值为 0.7,也就是将阈值设置为交换机端口队列容量的 70%.

最后,我们讨论 Nimble 和 Hedera 的网络开销.假设实验中每台服务器节点建立 10 条数据流[17],则本实验中总的网络流数目为 1280(128×10).OpenFlow 协议中,流统计请求消息的大小为 56 字节,响应消息的大小为 100 字节.所以,每次请求、响应消息的大小总计为 156 字节①.实验中,总计执行了 3 次流统计请求/响应,因此 Hedera 机制的总开销为 599040(1280×156×3)字节.对于 Nimble,实验中发送的拥塞通知消息的数目为 6170(阈值为 0.7 时).由于拥塞通知消息的大小为 20 字节,所以 Nimble 拥塞通知的总开销为 123400 字节,约为 Hedera 消息总大小的 20%.在最坏情形下,即阈值为 0.5 时,Nimble 拥塞通知的总开销为 234500,约为 Hedera 开销的 40%.从以上分析,我们得出如下结论:相比于 Hedera,Nimble 的拥塞通知机制极大地降低了拥塞通告的开销.

6 相关工作

近年来,科研人员提出许多方案以提高数据中心网络的带宽.为了在基础设施上提供对高带宽的支持,文献[1-6]提出了在服务器节点对间提供多条可达路径的网络拓扑结构.ECMP 协议是利用网络多路径提高应用性能的典型协议之一.ECMP 通过基于哈希算法的方式将数据流分发到不同的可达路径,以此实现均衡网络负载、提高网络吞吐量的目的.然而,ECMP 算法存在大流冲突问题[8].Dixit 等人[18]提出基于数据包的负载均衡策略以提高网络吞吐量.Sen 等人[19]则提出依据交换机端口负载信息均衡网络负载的方案.上述方案均是通过均衡网络负载的方式提高网络吞吐量,然而这些方案无法避免网络拥塞.与之不同,Nimble 通过快速地检测网络拥塞并调度拥塞链路上数据流到低负载链路的方式实现提高网络吞吐量的目的.

为了降低大流冲突对网络性能的影响,需要检测网络中的大流,并将其调度到低负载链路.Al-Fares 等人[8]使用 Hedera 架构实现更加有效地利用网络多路径的目的.Hedera 架构中,控制器周期性地向交换机发送流统计请求消息以检测网络中的大流,并将大流调度到低负载链路.然而,由于采用集中式的检测策略,制约了 Hedera 的可扩展性.Curtis 等人[11]提出一种新的大流检测机制,称为 Mahout.Mahout 采用分布式思想在终端节点检测大流,极大减小了其实现开销和资源需求.Xi 等人[20]提出一种通过动态改变数据流的 ECMP 哈希值,将数据流由高负载链路调度到低负载链路的机制.XCo[21]通过集中式控制程序控制拥塞链路上发送端的发送速率以避免网络拥塞.与之不同,Nimble 利用 packet-in 消息向控制器通告拥塞信息.同以上工作相比,Nimble 的主要优势是其可以快速地检测和响应链路拥塞.

为了充分利用网络资源,GRIN[22]通过利用网络空闲资源(如端口、链路等)的方式提高网络资源利用率,同时提高网络吞吐量.鉴于应用的不同特征,Webb 等人[23]提出为不同应用建立虚拟拓扑的方案,优化应用的网络性能.MicroTE[24]则通过检测应用的网络特征,预测应用的流量矩阵.同时,其通过调整网络资源分配的方式提高网络吞吐量.Nimble 则通过拥塞检测和数据流重调度的方式提高网络性能,所以 Nimble 可以和上述方案协同工作,以进一步提高网络吞吐量.

7 未来工作

尽管 Nimble 可以将拥塞链路上的数据流调度

① 请求消息和响应消息的大小基于 OpenFlow 规范 1.0.0 得出.由于 OpenFlow 协议处于不断发展阶段,不同 OpenFlow 协议版本,其大小可能存在略微差异.

到网络中的低负载链路，然而为了实现更加智能的调度策略，Nimble 架构需要集成更加有效的调度模块. 胖树拓扑具有良好的路由特性. 因此，对于任意通信节点对，只要确定其通信路径上的核心层交换机，该节点对间的路由路径便随之确定. 所以 Nimble 计算新路由路径的时间开销极小. 对于非胖树网络，只要确定了选用的路由算法，Nimble 可以适用于任意网络拓扑. 然而，对于非结构化拓扑，如 Jellyfish 等，Nimble 计算新路径的时间开销较大. 为此，Nimble 需要更加智能的策略以减少非结构化拓扑的路径计算开销. 一种可能的方案是为每一对通信节点维护一个低负载路径池. 当需要计算新的路由路径时，控制器仅仅从路径池中获取一条新的路径. 总地来讲，更加智能的调度模块以及路径池策略是我们未来需要面对的两个主要挑战.

8　总　结

OpenFlow 网络可以通过将拥塞链路上的大流调度到低负载链路的方式解决链路拥塞问题. 然而，由于当前的解决方案采用基于轮询的方式检测链路状态，致使它们不能有效地处理由突发流量导致的链路拥塞问题. 为了解决突发流量导致的链路拥塞问题，本文提出 Nimble 架构. 首先，Nimble 架构中，交换设备自主检测链路拥塞. 因此，Nimble 可以快速地检测网络拥塞. 其次，Nimble 通过扩展的 packet-in 消息实现拥塞信息通告. 最后，文中提出一种分布式的大流检测方案. 由于 Nimble 架构采用异步消息实现拥塞通告，拥塞通知的延迟和开销得到极大减小. 文中，我们通过模拟实验比较了 Nimble 和 ECMP、Hedera 的网络性能. 实验结果表明，较之 ECMP，Nimble 的网络性能得到极大提高. 同时，与 Hedera 相比，Nimble 的处理拥塞的响应时间得到极大减小.

致　谢　审稿专家和本报编辑为本文提出了许多宝贵的意见和建议，作者在此表示衷心的感谢！

参　考　文　献

[1] Al-Fares M, Loukissas A, Vahdat A. A scalable, commodity data center network architecture//Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication. Seattle, USA, 2008: 63-74

[2] Kim J, Dally W J, Scott S, Abts D. Technology-driven, highly-scalable dragonfly topology//Proceedings of the 35th Annual International Symposium on Computer Architecture. Beijing, China, 2008: 77-88

[3] Guo Chuanxiong, Lu Guohan, Li Dan, et al. BCube: A high performance, server-centric network architecture for modular data centers//Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication. Barcelona, Spain, 2009: 63-74

[4] Singla A, Hong C-Y, Popa L, Godfrey P B. Jellyfish: Networking data centers randomly//Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. San Jose, USA, 2012: 225-238

[5] Greenberg A, Hamilton J R, Jain N, et al. VL2: A scalable and flexible data center network//Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication. Barcelona, Spain, 2009: 51-62

[6] Guo Chuanxiong, Wu Haitao, Tan Kun, et al. DCell: A scalable and fault-tolerant network structure for data centers//Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication. Seattle, USA, 2008: 75-86

[7] Fu Binzhang, Han Yinhe, Ma Jun, et al. An abacus turn model for time/space-efficient reconfigurable routing//Proceedings of the 38th Annual International Symposium on Computer Architecture. San Jose, USA, 2011: 259-270

[8] Al-Fares M, Radhakrishnan S, Raghavan B, et al. Hedera: Dynamic flow scheduling for data center networks//Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation. San Jose, USA, 2010: 281-296

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

[10] Benson T, Akella A, Maltz D A. Network traffic characteristics of data centers in the wild//Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement. Melbourne, Australia, 2010: 267-280

[11] Curtis A R, Kim W, Yalagandula P. Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection//Proceedings of the 30th IEEE International Conference on Computer Communications. Shanghai, China, 2011: 1629-1637

[12] Tootoonchian A, Ganjali Y. HyperFlow: A distributed control plane for OpenFlow//Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking. San Jose, USA, 2010: 13-18

[13] Koponen T, Casado M, Gude N, et al. Onix: A distributed control platform for large-scale production networks//Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation. Vancouver, Canada, 2010: 351-364

[14] Alizadeh M, Greenberg A, Maltz D A, et al. Data center TCP (DCTCP)//Proceedings of the ACM SIGCOMM 2010 Conference on Data Communication. New Delhi, India, 2010: 63-74

[15] Floyd S, Jacobson V. Random early detection gateways for congestion avoidance. IEEE/ACM Transactions on Networking, 1993, 1(4): 397-413

[16] Henderson T R, Lacage M, Riley G F, et al. Network simulations with the ns-3 simulator//Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication. Seattle, USA, 2008: 527-527

[17] Tavakoli A, Casado M, Koponen T, Shenker S. Applying NOX to the datacenter//Proceedings of the ACM Workshop on Hot Topics in Networks. NewYork, USA, 2009: 103-108

[18] Dixit A, Prakash P, Kompella R R. On the efficacy of fine-grained traffic splitting protocolsin data center networks//Proceedings of the ACM SIGCOMM 2011 Conference on Data Communication. Toronto, Canada, 2011: 430-431

[19] Sen S, Shue D, Ihm S, Freedman M J. Scalable, optimal flow routing in datacenters via local link balancing//Proceedings of the 9th ACM Conference on Emerging Networking Experiments and Technologies. Santa Barbara, USA, 2013: 151-162

[20] Xi Kang, Liu Yulei, Chao H J. Enabling flow-based routing control in data center networks using probe and ECMP//Proceedings of the IEEE Conference on Computer Communications Workshops. Shanghai, China, 2011: 608-613

[21] Rajanna V S, Jahagirdar A, Shah S, Gopalan K. Explicit coordination to prevent congestion in data center networks. Cluster Computing, 2012, 15(2): 183-200

[22] Agache A, Raiciu C. GRIN: Utilizing the empty half of full bisection networks//Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Computing. Boston, USA, 2012: 37-42

[23] Webb K C, Snoeren A C, Yocum K. Topology switching for data center networks//Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services. Boston, USA, 2011: 79-84

[24] Benson T, Anand A, Akella A, Zhang M. MicroTE: Fine grained traffic engineering for data centers//Proceedings of the 7th Conference on Emerging Networking Experiments and Technologies. Tokyo, Japan, 2011: 1-12

LI Long, born in 1987, Ph. D. candidate. His main research interests include software-defined networking and QoS guarantees in data center networks.

FU Bin-Zhang, born in 1982, Ph. D., associate professor. His research interests include high-performance and high-reliable interconnection networks.

CHEN Ming-Yu, born in 1972, Ph. D., professor. His research interests include computer architecture, operating systems, and algorithm optimization for high performance computer.

ZHANG Li-Xin, born in 1971, Ph. D., professor. His research interests include data center computing systems, advanced cache/memory systems, architectural simulators, parallel computing, performance evaluation, and workload characterization.

Background

Unlike transitional LANs and WANs, data center traffic is dominated by traffic inside the data center. To improve network performance of modern data centers, a great number of topologies offering high bisectional bandwidth have been proposed. However, traffic congestion will dramatically degrade network performance.

Al-Fares et al. proposed an elephant flow management system called Hedera. Hedera deals with traffic congestion by rescheduling elephant flows to under-utilized paths. However, Hedera only detects and reschedules elephant flows, rather than detecting traffic congestion in the network. Hedera detects elephant flows by the controller periodically. Thus, it cannot detect elephant flows instantly. In addition, limited by the centralized controller, it cannot scale to large data centers. Curtis et al. proposed an end-host based elephant flow detection mechanism, called Mahout. Although it is scalable, Mahout is implemented in TCP stack which is out of the control of data center administrators. Therefore, malicious users may change the elephant flow detection policy by customizing their operating systems. Moreover, Mahout cannot handle traffic congestion in the network.

In this paper, the authors propose a fast flow scheduling strategy for OpenFlow networks, called Nimble, which detects traffic congestion and reschedules elephant flows to under-utilized paths. The authors also propose a simple elephant flow detection mechanism. With the extended OpenFlow protocol, Nimble can detect traffic congestion as soon as possible. Further, Nimble improves network performance significantly by rescheduling elephant flows from congested paths to those under-utilized ones.

This project was partially supported by the National Natural Science Foundation of China under Grant Nos. 61221062, 61331008, and 61202056, and the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant No. XDA06010401.