DOI:10.7544/issn1000-1239.2018.20160743
55(2):346-357,2018

基于流量特征的 OpenFlow 南向接口开销优化技术

郑 鹏 胡成臣 李 昊
(西安交通大学计算机科学与技术系 西安 710049)
(智能网络与网络安全教育部重点实验室(西安交通大学) 西安 710049)
(zeepean@gmail.com)

Reducing the Southbound Interface Overhead for OpenFlow Based on the Flow Volume Characteristics

Zheng Peng, Hu Chengchen, and Li Hao
(*Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an* 710049)
(*Key Laboratory of Intelligent Networks and Network Security (Xi'an Jiaotong University), Ministry of Education, Xi'an* 710049)

Abstract Software defined networking (SDN) decouples the control plane from the switch in the data plane, which forms the SDN controller. This paradigm introduces many benefits, e.g., openness, management simplicity, etc. Nevertheless, the separation of the SDN switch and the controller also leads to great communication overhead between them due to controlling the network (the number of the control message and Table-Miss packets), and the overhead becomes the major bottleneck of SDN. On the one hand, each Table-Miss event can produce multiple Flow-Mod messages which add extra bandwidth overhead as well as delay to the southbound interface. On the other hand, controller has no awareness of flow characteristic information behind the Flow-Mod messages which make the overhead worse. This paper proposes a new architecture uFlow (split up Flow) to mitigate the overhead at the controller side based on the flow volume characteristics. We implemented the prototype of uFlow system both in software-based platform mininet and hardware-based platform ONetSwitch. Experimental results driven by the real traffic show that uFlow can significantly reduce the communication overhead between control plane and data plane, the number of the control message has a decrease of 70% off on average, eliminate redundant update of flow entries in switch and reduce the transmission delay of packets.

Key words software defined networking (SDN); southbound interface; communication overhead; flow optimization; OpenFlow; Table-Miss

摘 要 软件定义网络(software defined networking, SDN)分离的数据平面和控制平面,给网络管理带来了开放性和灵活性.但同时控制器与交换机之间的接口(控制器南向接口)需要更频繁的交互各种消

收稿日期:2016-10-11;**修回日期**:2017-10-25
基金项目:国家自然科学基金项目(61672425,61702407);国家重点研发计划项目(2016YFB0800101,2017YFB0801703);微软亚洲研究院合作研究项目(2016JM6066);国家电网研究项目(DZ71-16-030)
This work was supported by the National Natural Science Foundation of China (61672425, 61702407), the National Key Research and Development Program of China (2016YFB0800101, 2017YFB0801703), the Microsoft Research Asia Collaborative Research Program (2016JM6066), and the State Grid Corporation of China Research Program (DZ71-16-030).
通信作者:胡成臣(huc@ieee.org)

息以实现对网络的控制.一方面,数据平面触发 Table-Miss 的数据包需要通过 Packet-In 消息往返于交换机与控制器之间,时延增大的同时也给控制器南向接口带来繁重的通信开销,数据平面和控制平面之间的交互容易成为网络性能的瓶颈.另一方面,控制器在下发新的流表项时,由于缺乏新表项对应的数据流特征信息,易出现已有的大流表项被下发的小流表项替换的情况,造成冗余的 Flow-Mod 消息(流表更新消息)和 Packet-In 消息,进一步加重了南向接口的通信开销,降低了网络的整体性能.提出一种基于流量特征的 OpenFlow 南向接口开销优化技术 uFlow,在控制器上通过对 Packet-In 消息中数据流量特征的识别以及对小流的直接转发,达到消除南向接口冗余开销的目的.对 uFlow 的原型系统进行了实现,并通过真实网络中的流量对 uFlow 优化效果进行了验证.实验结果显示:与传统的 OpenFlow 网络处理方式相比,uFlow 消除了冗余的交换机流表项更新,显著地降低了 OpenFlow 南向接口的交互开销:在不同的网络负载和流表容量的情况下,uFlow 平均能减少 70%以上的 Flow-Mod 消息.

关键词　软件定义网络;南向接口;交互开销;流量优化;OpenFlow;流表失配

中图法分类号　TP393

软件定义网络(software defined networking,SDN)通过分离的控制平面和数据平面,给网络管理提供了统一的编程接口,带来了极大的灵活性.OpenFlow 作为最具影响力的控制平面和数据平面的通信协议,成为了业界的事实标准[1].但是这种控制平面与数据平面完全分离的方式需要在 OpenFlow 交换机和控制器之间频繁地交互各种控制器南向接口(控制器与交换机之间的接口)消息.而过于繁杂的南向接口交互开销对控制器的处理能力、数据通路处理延时、南向接口的信道带宽等造成很大的压力,成为网络性能提升的瓶颈.

在 OpenFlow 网络中,交换机和控制器之间开销主要体现在控制器通过交互指挥交换机完成对数据包的处理.OpenFlow 网络通过控制器下发到交换机中的流表来确定数据包的转发等操作.当新的数据包达到 OpenFlow 交换机时,典型的处理过程如图 1 所示.抵达网络的数据包首先与交换机中流表项的匹配域(match field)依次匹配,如图 1 数据流路径 1)与控制流路径①.若匹配成功,则按照该流表项的动作域(action field)处理数据包,比如转发、丢弃等;若匹配失败,则触发 Table-Miss 事件,

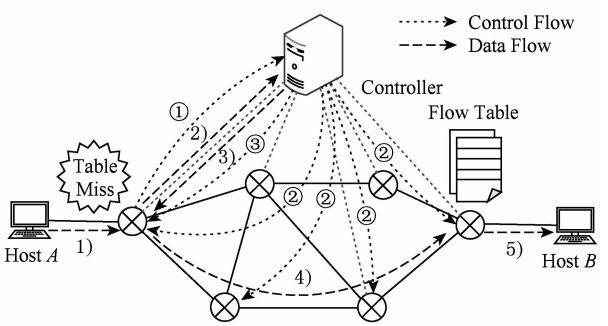


Fig. 1　OpenFlow procedure of control flow and data flow

图 1　典型的 OpenFlow 网络控制流和数据流示意图

交换机通过 Packet-In 消息将数据包交给控制器处理,如图 1 中数据流路径 2).控制器计算数据包的转发路径,并通过 Flow-Mod 消息将流表项安装到交换机,如图 1 中控制流路径②.同时控制器通过 Packet-Out 消息将数据包返回给交换机,如图 1 中数据流路径 3)和控制流路径③.数据包回到交换机后,匹配这条流的数据包都按照新下发的流表项抵达目的地,如图 1 中数据流 4)和 5)所示.

由于交换机中的流表资源容量有限,无法匹配到所有的数据包.对于无法匹配到正常流表项的数据包,都可能会给交换机和控制器的交互带来大量的 Packet-In 消息和 Flow-Mod 消息,同时也会导致如下问题:1)分离的数据平面和控制平面使得 Table-Miss 的数据包需要往返于交换机与控制器之间,给网络数据流带来较大的时延和冗余的转发路径,两者之间的交互容易成为网络的瓶颈,影响网络的性能[2].文献[3]也指出数据流路径建立时的 Table-Miss 是造成包传输时延的一个重要原因.2)流表"更新振荡"问题.当交换机中流表不足时,新流产生的 Flow-Mod 流表项找不到空间,则会替换掉交换机中现有的流表项.然而控制器并不知道新下发的流表项对应的数据流信息.如果新流表项对应的数据流是一条小流(总字节数或总包数很小的流,具体定义见 2.2 节),那么小流表项的下发很可能会替换掉原流表中大流(其定义见 2.2 节)对应的流表项.此后,交换机中的小流表项只会匹配到少量的数据包,造成流表利用率降低,流表资源浪费;而被替换的大流表项很大概率会有后续的数据流到来,触发更多的 Packet-In 消息和 Flow-Mod 消息,新到大流流表项的再次下发导致交换机中的流表项频繁的变化和冗余的更替.这种流表更新振荡现象

进一步加重了控制器南向接口的通信开销,降低了网络的整体性能,难以满足当前的云数据中心发展对网络的带宽和时延有着越来越高的要求[4].

对于增大的时延和冗余的转发路径,分析数据包转发过程可以发现,除正常的转发路径之外,数据包需要额外的往返于交换机和控制器之间. 如图 2 所示,传统 OpenFlow 网络数据流的转发路径 3)和 4)是存在冗余的. 这种冗余路径的根源在于,控制器在计算数据包的转发路径时忽略了控制器这个特殊的"交换节点"的存在. 而事实上,控制器在网络中是承载着"交换节点"的功能的,因为网络中不断会有触发 Table-Miss 的数据包(或包头)到达控制器,这些数据包只能由控制器来处理. 直观上,如果将图 2 中数据包经过的路径 3)和 4)替换为 3′),冗余的路径 4)就可被消除,数据包的传输时延就能降低. 问题的困难之处在于我们无法简单地使用路径 3′)替换掉路径 3)和 4),因为控制器根本不可能承受全网的流量. 而流表更新振荡问题是由控制器下发流表项时缺乏表项对应的数据流信息导致,因此在控制器上实现对数据包流量的细粒度识,是防止大流表项被小流流表项替换的关键所在.

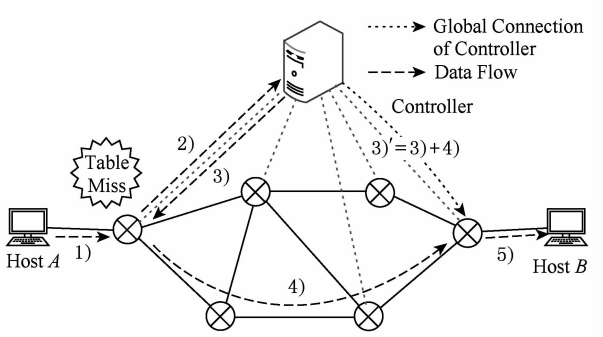


Fig. 2 The redundant forwarding path of OpenFlow data flow

图 2 传统 OpenFlow 网络中的转发路径冗余

真实网络中的流量分布具有显著的不均衡特性[5]:为数不多的大流占据着大量的网络带宽,但消耗的控制指令并不多;而占据少量网络带宽的小流,由于其数量巨大,需要消耗更多的控制指令和流表资源,甚至会抢占大流的资源导致重复的控制指令导致流表更新振荡问题. 这不仅消耗了大量的数据平面和控制平面之间的信道资源,造成了数据平面和控制平面的冗余交互,同时也会造成交换机流表资源的浪费,增加数据包的时延.

本文基于网络流量分布不均衡的特征,重点针对仅占据少量网络带宽但是种类繁多的小流,消除冗余路径,降低传输时延,优化控制器南向接口交互开销.

1 相关工作

为了减小控制器的压力,降低南向接口开销,文献[3]在交换机上利用额外的流表资源,记录允许上传给控制器的数据流信息,过滤出重要的信息转发给控制器,减轻控制器压力. 但是该机制不仅消耗了过多流表资源,还需要对交换机进行修改,很难在真实网络中使用. 文献[5]深入分析了网络流量的分布特性,并且提出了利用流量分布不均衡特性来进行路由器设计的策略. 作者针对单个网络设备上的控制面和数据转发面,提出了 TFO(traffic-aware flow offloading)算法,用于提升转发表的命中率,减小本地控制平面与数据平面的通信开销. 但其针对单个网络设备"紧耦合"的数据-控制平面设计的算法无法用于解决 OpenFlow 网络中数据-控制平面之间的交互开销问题,原因如下:

1) 适用的网络体系结构有本质的区别. TFO 算法是对其先前工作基于硬件加速的软件路由器原型[6]的系统描述,其算法适用于路由器内部的控制平面和数据平面,如果网络中有多个转发设备,那么每个设备上都需要运行其算法. 而本文研究适用于数据-控制平面物理分离的 OpenFlow 网络,能部署于整个网络的集中式控制器上的算法.

2) 适用的体系系统不同导致算法的开销有巨大的差异. 由于 TFO 算法仅仅运行在单个转发设备上,因此需要处理的流量规模仅限于单个设备上未命中转发表的新流,单节点失配的流量规模允许转发设备上的控制器维护在不同时间片内(1 s, 10 s, 10 min)所有流的大小排序;但是 OpenFlow 控制器面向全网所有的转发设备,难以维护每个设备上的流量信息. 另外,单个设备上靠近的数据-控制平面也允许 TFO 算法的控制器与转发表实时的交互,例如在每一次更新转发表之前,实时计算现存于转发表的表项与拟更新的表项的差异. 但是在 OpenFlow 网络中要求控制器实时维护每个设备上的转发表,或者在每次更新转发表之前都向数据平面的设备请求当前的转发表内容,不仅会对控制器的性能提出了更高的要求,同时也给数据-控制平面之间的交互引入了新的开销,这也与本文想要解决的问题存在冲突. 因此,针对 OpenFlow 网络南向接口开销优化需要探索新的方法.

文献[7]提出控制器按照主动(proactive)方式执行规则下发:根据已知信息以及收集到的网络特征,预测将要发生的转发等事件,并提前安装对应规则.这种方法的优点在于,由网络中新流到达导致的 Packet-In 消息以及流安装的事件被大大减少,但是它需要更大的硬件流表支持才能安装足够多的流预装,在 OpenFlow 版本更新到 1.3 版本之后[8],交换机将在流表溢出时直接用新安装的流表项覆盖原有流表项,因此无法达到预期的目标,同时它对小流导致的控制指令数量并没有改善.文献[9]希望能够结合 2 种策略的优点,根据流特征使用不同策略下发规则.但是由于实际流量特征很难简单预测,在很多情况下并不能有效运行.在控制器资源有限的情况下,文献[10]主动抛弃用户设置的 Packet-In 消息,以保证基础消息可以得到正常的响应.文献[11]也有队列机制,防止关键消息受到一般消息的阻塞.这种队列机制都以牺牲响应速度为代价来保证 SDN 控制平面的基本功能可以正常运行.在大规模的网络中,控制平面需要每秒能够处理百万条流产生的 Packet-In 消息[12].文献[13]中提出了利用控制器直接将数据包转发到目的交换机的策略,但这种策略要求控制器转发数据包的同时也下发对应的流表更新消息,因此无法解决流表冗余更新的问题,而这正是本文着重解决的问题.

2　系统设计

2.1　可抑制的繁重的交互开销

本文优化的交互开销重点是指往返于控制平面和数据平面之间的"可抑制的繁重的交互开销",其中"可抑制"表示当前的 OpenFlow 模型有可改进的空间,而"繁重"意味着相关开销可成为网络性能瓶颈,对交互开销的优化具有较大的价值.通过对当前的 SDN 网络的分析,我们可以归纳出如下 2 类典型的可抑制的繁重的交互开销:

第 1 类是控制器对数据平面重复性控制导致的开销.OpenFlow 网络中交换机完全按照控制器的指令对流量进行转发.指导数据包转发行为的指令存储在格式为规则-行为(rule-action)流表中,然而交换机中的流量资源是稀缺的,不可能存储所有的转发信息.因此随着网络事件的不断发生,完成数据包的处理需要越来越多的指令;在交换机有限的资源限制下,旧的指令由于新的决策加入而被替换或者删除.一个难以避免的问题是,这些被替换的指令信息并不是毫无用处,随着后续网络事件的发生,交换机可能还需要使用与被删除指令同样的信息来进行决策.这会导致同样的网络事件重复发生的时候,控制器将不断重复进行数据包识别,决策以及控制命令下发,从而导致冗余的交互开销.而我们对流量特征的分析结果(见 3.1 节)显示,仅占总带宽 1% 的小流数量超过了总流数的一半,这就会消耗超过一半的控制器的转发指令.因此,重复性控制指令的开销有很大优化空间.

第 2 类是数据平面流量从交换机穿越到控制器导致的开销,主要是由数据包触发 Table-Miss 事件导致.这类开销数量巨大,当交换机流表资源不足导致的流表更新振荡现象会让这类消息变得尤为繁重.

2.2　开销优化方案

本文提出一种基于流量特征的 OpenFlow 南向接口开销优化技术 uFlow,通过控制器对数据流进行细粒度的识别和记录,在不影响控制器性能的情况下,利用控制器的可达性对占据少量网络带宽但是种类繁多的小流直接转发,达到消除 OpenFlow 南向接口冗余开销,节省交换机流表资源,同时减少时延的目的.

uFlow 通过控制器上的流量实时识别算法,将需要处理的数据流识别为 2 类:大流和小流.

定义 1. 小流.在 Δt 时间内,对应的数据包数量小于 N 且总字节数小于 S 的流.

定义 2. 大流.在 Δt 时间内,对应的数据包数量大于 N 或总字节数大于 S 的流.参数 $\Delta t, N, S$ 作为流量识别算法的参数,可以根据网络运行状况动态确定.

如果控制器识别到数据包对应的是一条小流,那么控制器直接将该流对应的所有数据包送达目的地交换机,如图 3 所示.这样既减少了该流的传输时延,又节省了交换机的流表空间.如果数据流被识别为大流,那么控制器首先将识别大流过程中到达的数据包直接送达目的交换机,同时立即下发流表项

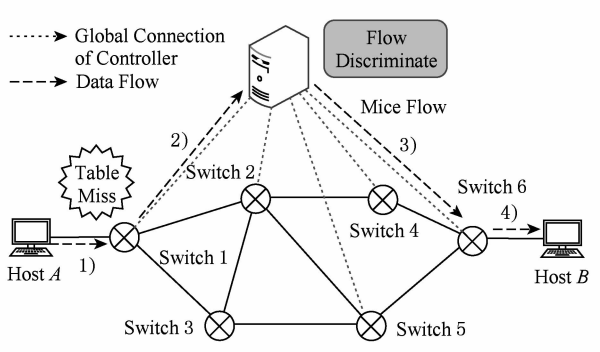


Fig. 3　Mice flow procedure of uFlow

图 3　uFlow 对小流的处理流程

到交换机,使得后续的数据包都能够通过流表项来转发,如图 4 所示,由此减少冗余的流表更新.

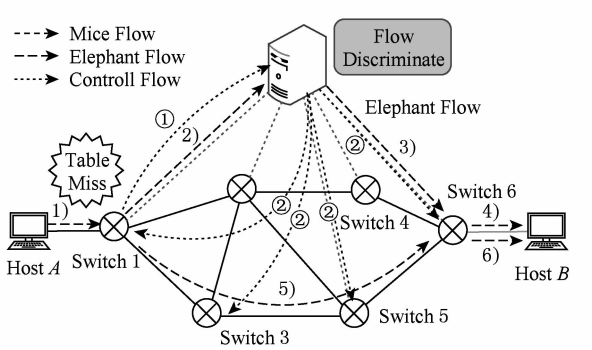


Fig. 4 Elephant flow procedure of uFlow

图 4 uFlow 对大流的处理流程

这种解决方案主要有如下优点:

1) 极大地降低 OpenFlow 网络中数据平面和控制平面的交互开销,消除交换机和控制器之间的冗余交互.

2) 节省大量流表资源. 值得注意的是,本文提出的方案与文献[14]中的 Rule-Caching 算法并不冲突,二者是在 2 个不同的维度上节省流表资源. 本文的方案中节省流表资源的原因在于,通过控制器对小流直接送达目的地,减少了交换机中小流对应流表项的安装,消除了潜在的流表冗余更新.

3) 节省 OpenFlow 交换机中的缓存空间,释放控制流量. 经典的处理方式中,交换机必须要缓存 Packet-In 消息与 Flow-Mod 消息这段时间内的数据包的内容,浪费了宝贵的缓存资源. 而在本文的处理方式中,直接将整个数据包交给控制器来送达目的地. 直观上看来,这会消耗额外的交换机与控制器之间的带宽资源,增加南向接口的开销,但是我们的实验(见 3.2 节)表明并非完全如此:因为这种方案消除了流表振荡和冗余交互.

4) 这种设计可以大大降低小流时延.

2.3 uFlow 系统设计

uFlow 在不影响 OpenFlow 控制器上其他应用程序对大流转发的前提下,消除小流反复进入退出对控制器网络资源与计算资源的消耗. uFlow 系统在控制器中的结构如图 5 所示,位于 Packet-In 消息入口和控制器核心服务之间. 来自 OpenFlow 交换机的 Packet-In 消息首先由大小流识别模块处理. 如果 Packet-In 消息内容被识别为大流数据包,则直接将数据包交还给控制器,按照正常流程处理,比如后续由路由应用计算并下发相应的流表项.

如果被识别为小流,数据包则转交给小流匹配表处理. 小流匹配表中记录了该模块已处理过的小流信息,包括交换机端口与数据包目的地址的对应关系等. 若匹配成功,则更新小流匹配表中的计数信息,并通过 Packet-Out 消息直接将数据包送达目的地;若匹配失败,则向控制器请求全局拓扑信息,并将获取到目的交换机 ID 和端口信息加入查找匹配表,最后再通过 Packet-Out 消息将数据包送达目的地,从而消除了小流量反复进入退出对控制器网络资源与计算资源的消耗.

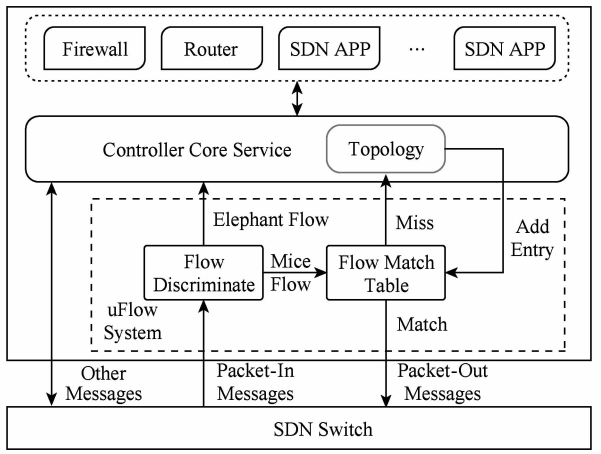


Fig. 5 Architecture of uFlow system

图 5 uFlow 系统架构

下面我们通过与传统 OpenFlow 网络中数据包处理流程对比,来说明 uFlow 系统对于数据包处理的过程及特点.

1) 传统 OpenFlow 网络中数据包处理流程

在传统的 OpenFlow 网络中,正如图 6 所示,当数据包到达 OpenFlow 交换机后,首先与交换机上的流表进行匹配,若匹配成功,数据包按照匹配表项的指令进行处理;若匹配失败,数据包则被 Packet-In 消息转发给控制器处理. 控制器通过全局拓扑信息计算出数据包的转发路径,并通过 Flow-Mod 消息将新的流表项下发给交换机.

交换机收到 Flow-Mod 添加流表的消息后,如果流表还有剩余空间,则直接将新的流表项插入流表中;如果流表已经装满,则按照某种替换策略删除一条旧的流表项,然后再插入新的流表项. 常用的替换策略包括最近最少使用算法(least recently used, LRU)、最不经常使用算法(least frequency used, LFU)等. 本文实验中采用替换策略为实际应用中常用到的 LRU 替换方法. 数据包处理的整体流程如图 6 所示.

2) uFlow 系统中数据包处理流程

与传统的 OpenFlow 网络处理流程不同,控制器收到 Packet-In 消息后并不直接进行路由计算或

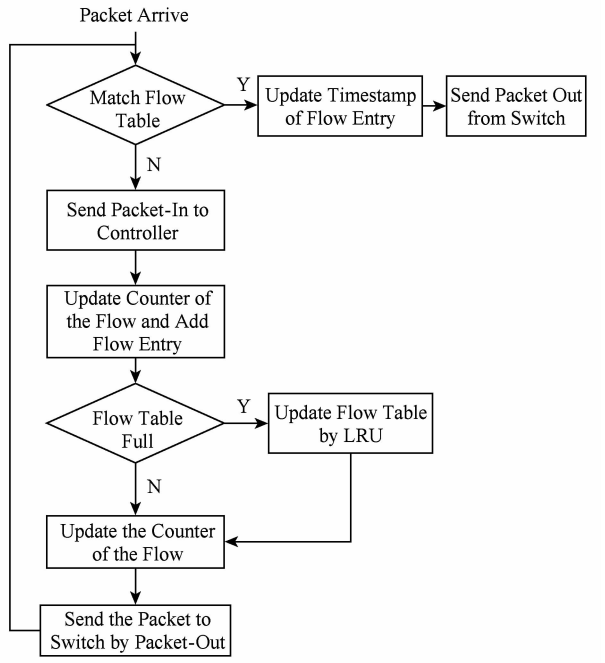


Fig. 6　Packet processing flow in OpenFlow Network

图 6　传统 OpenFlow 网络中数据包处理流程

者下发流表,而是首先对数据包进行大小流判断,根据判断的结果作相应的处理. 当数据包被判断为大流之后,控制器才会下发 Flow-Mod 消息,达到节省控制信令、减少流表替换的效果. 当数据包被判断为小流之后,控制器则直接将其送达目的地. uFlow 系统对数据包处理的整体流程如图 7 所示:

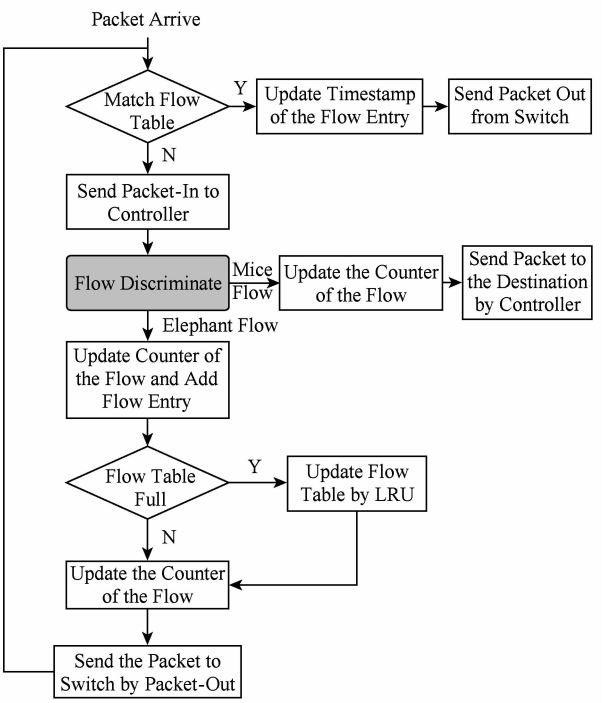


Fig. 7　Packet processing flow in uFlow Network

图 7　uFlow 系统中数据包处理流程

2.4　大小流识别算法

大小流识别模块是 uFlow 系统对数据包分析的入口,负责处理所有 Packet-In 消息并区别大小流,根据识别的结果决定不同的处理方式. 当 Packet-In 到达控制器后,小流代理系统首先将该数据包交给大小流识别模块. 为了满足数据包快速转发的要求,我们设计了简单高效的大小流识别算法:根据到达数据包的在一段时间内的总数目和字节数来区分大小流. 初始阶段,我们认为所有的数据包都是小流,由控制器直接转发.

大小流识别模块记录着当前有效的大流集合和所有流的计数信息. 数据包到达控制器之后,首先更新该数据包所在流的统计信息. 如果该流在 Δt 时间内累计的数据包总数超过 N,或者数据包累计总字节数超过 S,则数据包直接被判断为大流数据包;否则判断为小流数据包. 最后,根据超时信息再次更新流的统计信息. 大小流识别伪代码描述如算法 1 所示:

算法 1. 大小流识别算法.

输入:待识别的数据包 pkt;流的总包数,大小流区分参数 N;流的总字节数,大小流区分参数 S;控制器中记录每条流包数量 $Ncnt$;控制器中记录每条流的总字节数 $Scnt$;

输出:0,数据包被判断为小流;1,数据包被判断为大流.

过程:FLOW DESCRIM (pkt,N,S,$Ncnt$,$Scnt$)

```
BEGIN
  flowID ← get_flowID(pkt);
  pkt_num←get_pkt_num(pkt);
  pkt_size←get_pkt_size(pkt);
  update_Ncnt(Ncnt,flowID,pkt_num);
  update_Scnt(Scnt,flowID,pkt_size);
  IF flowID.num>N ‖ flowID.size>S
    RETURN 1;
  ELSE
    RETURN 0;
  ENDIF
  Check_timeout(Ncnt,Scnt).
END
```

3　实验验证

3.1　流量分布特征实验

本小节将通过对实际网络流量的统计来说明流量分布极度不均衡的特征,从而给 uFlow 的系统设计提供理论依据.

我们总共分析了 22.71 GB 来自 ISP 核心路由器上的流量[15]的分布特征. 通过固定长度的目的 IP 前缀(本文使用目的 IP 的前 16 b)来区分一条流,我们分别得到了在不同的时间窗口下,真实流量中每条流对应的数据包总数量和总大小的分布特征. 图 8 和图 9 分别展示了在 30 min 内通过核心路由器上的流量中每条流对应的数据包总数目以及流数目的累计分布图. 如图 9 所示,占比为 1.3%的 863 条流对应的数据包总数达到了 90%. 表 1 展示了在不同的时间窗口下网络流量的分布特征.

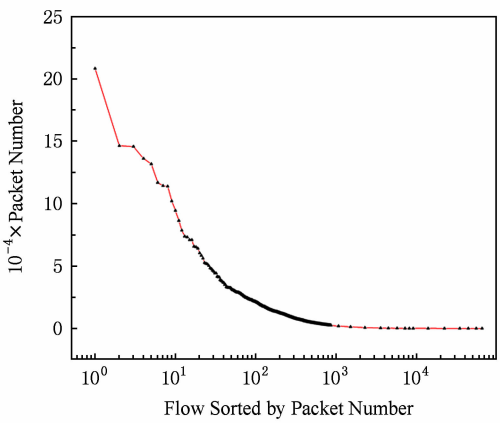


Fig. 8 Packet number distribution in real traffic

图 8 数据流对应的包数目分布图

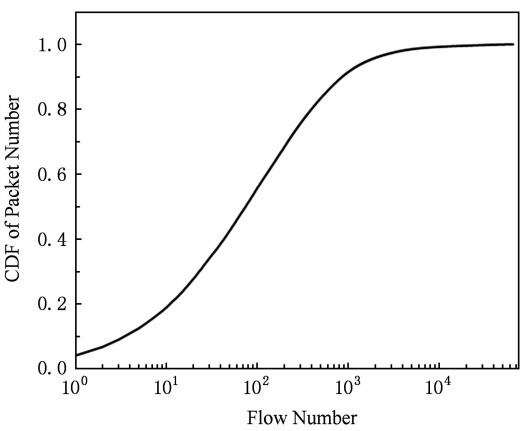


Fig. 9 CDF of packet number in real traffic

图 9 数据流对应包数目的累计分布图

这些数据都表明真实流量的分布特征是极度不均衡的. 网络中的小流种类繁多,占比非常高,因此与小流相对应的流表项数量很大,需要消耗的控制平面与数据平面交互开销也非常高. 通过消除这些小流流表项的安装来降低控制器南向接口交互开销,节省交换机中的流表资源,符合 uFlow 系统设计的预期.

Table 1 Distribution of Traffic in Different Time Slot

表 1 不同时间窗口下网络流量分布的不均衡特征

Time Slot /s	Average Flow Number	Bandwidth of Top 1% Flow/%	Bandwidth of Top 1% Flow/%	Flow Number of Least 1% Bandwidth/%
1	2910	38	87	50
10	10350	62	96	71
600	64095	86.55	98.90	88.38

3.2 交互开销优化实验

为了验证 uFlow 系统对 OpenFlow 南向接口开销优化的效果. 我们将交互消息划分为如下 2 个类别:第 1 类控制平面发出的控制指令,主要是 Flow-Mod 消息,我们称之为控制流交互开销;第 2 类是由数据平面发出的携带数据包的消息,主要是 Packet-In 消息,我们称之为数据流交互开销. 我们在 SDN 物理交换机 OnetSwitch[16]平台上对 uFlow 原型系统进行了实验和测试. 通过对真实网络流量回放的方式,对 uFlow 系统的性能进行了测试,主要测试指标为:

1) 控制流交互开销. 控制器向交换机下发流表的总条数,即 Flow-Mod 消息的总数量.

2) 数据流交互开销. 数据平面穿越控制器的总流量,即 Packet-In 消息中数据包的总量.

作为对比,我们也使用同样的流量对传统的 OpenFlow 网络处理系统交互开销进行了测试.

3.2.1 实验参数

在本文实验中,为了保持大小流区分算法的简单与快速. 我们设置小流参数为:在 $\Delta t=10$ s 时间内,最多包数目 $N=16$,最大字节数 $S=12\,800$ B. 这些参数的设置来自于 3.1 节分析的数据流量中有近 80%左右的数据流对应的包数和字节数都不高于此. 由于篇幅限制,关于参数的自适应调整和分析可以作为一步的工作,不在此展开.

3.2.2 测试数据

本实验中测试用到的数据来自新西兰 WAND 网络研究组提供的某 ISP 核心路由器上的真实网络流量①. 这些数据流由 DAG 3.7 GB 捕获卡[16]接入 ISP 核心路由器持续捕获得到,包含了连续多天的流量. 我们从不同时间段,根据链路负载程度选择出

① http://wand.net.nz/wits/index.php

3 段具有代表性的流量进行实验,每段流量的持续时间均为 30 min.

1) Trace1:表示 30 min 重度负载流量.包含当地时间 15:30－16:00 时间段内通过核心路由器上所有数据包.流量总大小为 22.71 GB,数据包总数为 40 311 405.

2) Trace2:表示 30 min 中度负载流量.包含当地时间 18:30－19:00 时间段内通过核心路由器上所有数据包.流量总大小为 16.13 GB,数据包总数为 28 710 953.

3) Trace3:表示 30 min 轻度负载流量.包含当地时间 02:00－02:30 时间段内通过核心路由器上所有数据包.流量总大小为 6.01 GB,数据包总数为 11 155 171.

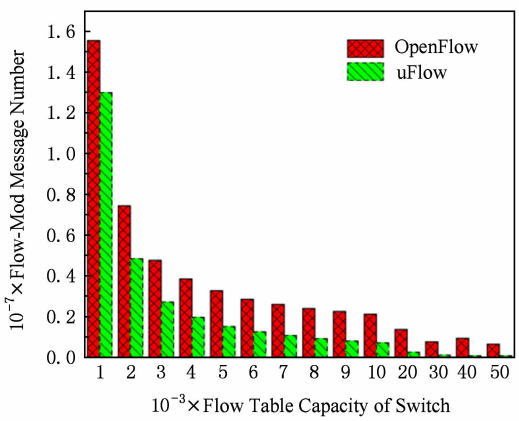
3.2.3　开销优化实验结果

下面是 OpenFlow 南向接口交互开销优化的实验结果,包括控制流交互开销和数据流交互开销.

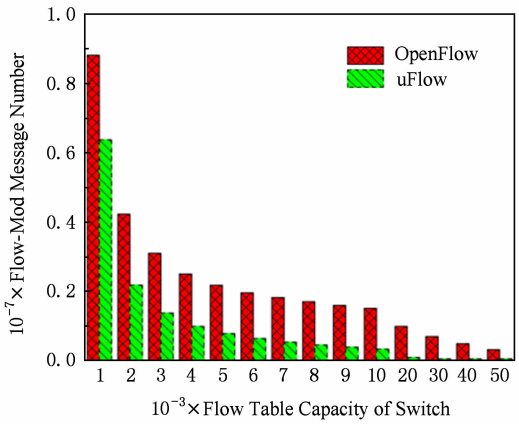
1) 控制流交互开销

控制流交互开销优化效果如图 10 所示,图中展示了 uFlow 与传统 OpenFlow 网络中处理方式(LRU)消耗的控制流交互开销总量对比.图 10(a)(b)(c)分别表示在重度负载流量、中度负载流量和轻度负载流量这 3 种情况下,uFlow 系统与传统 OpenFlow 网络消耗的 Flow-Mod 消息数量对比.结果表明处理 30 min 网络流量,uFlow 系统消耗的 Flow-Mod 数量远小于传统 OpenFlow 网络中的处理方式.对于重度负载流量,uFlow 在平均情况下能够节省 60%左右的 Flow-Mod 消息数量;对于中度和轻度负载的数据流量,uFlow 平均情况下分别能够节省 80%和 90%左右的 Flow-Mod 消息数量.

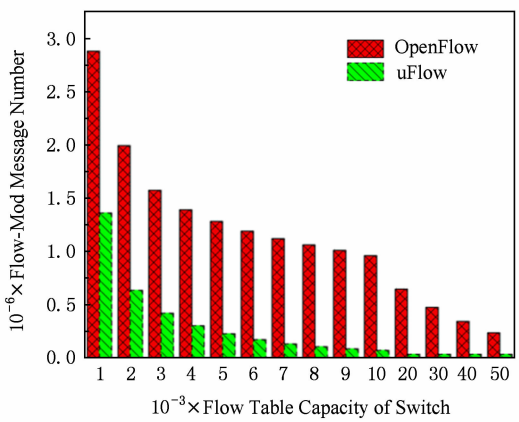
进一步分析可以发现,在不同的交换机容量从 1 000～50 000 条不断增加的情况下,uFlow 对控制流交互开销的优化程度(即 uFlow 系统对 Flow-Mod 消息的降低比例)呈现出先增后降的趋势,如图 11 所示.当交换机流表容量小于 20 000～30 000 条时,交互开销优化比例随流表增加而增大.这是由于在这段区间内,uFlow 优化主要体现在流表命中率的提高,这极大地减少了 Table-Miss 事件的数量,抑制了大量的重复性控制指令.而后续当流表容量继续增大后,流表的数量越来越接近网路中总流数目,Table-Miss 事件本身会随着流表容量的增加而大大减小.此时 uFlow 对小流识别并直接转发会需要消耗一定的交互开销,因此优化比例会随着流表容量的继续增大而降低.当然,这种下降趋势可以通过动态调整流量识别算法来做进一步的改进.



(a)　Heavy overhead traffic:Trace1



(b)　Medium overhead traffic:Trace2



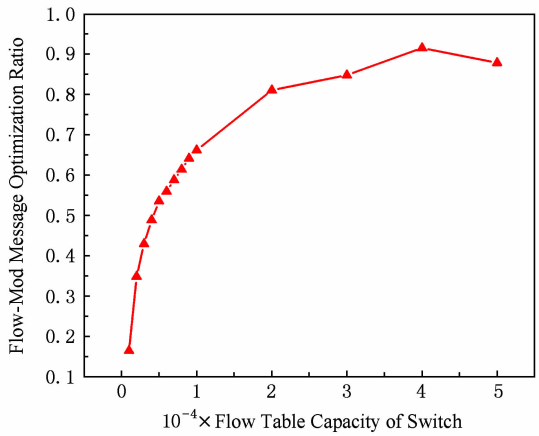
(c)　Light overhead traffic:Trace3

Fig. 10　Control flow overhead of uFlow and OpenFlow

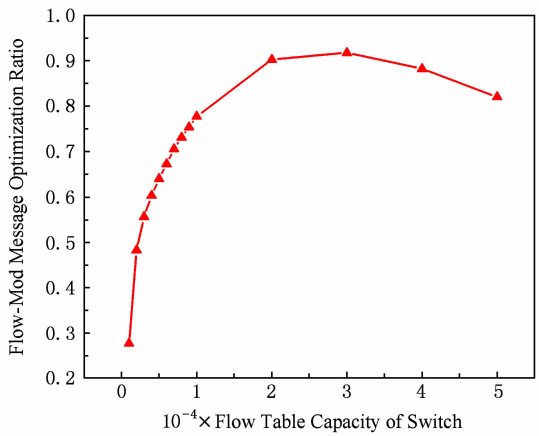
图 10　控制流交互开销对比图

2) 数据流交互开销

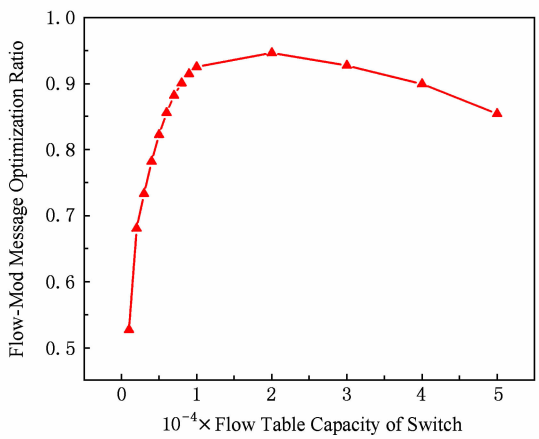
我们对传统 OpenFlow 网络和 uFlow 系统中的 Packet-In 消息携带的数据包总流量进行了统计,结果如图 12 所示.图 12 展示了在不同的交换机流表容量下,uFlow 系统与传统的 OpenFlow 网络中数据流的开销(即控制器需要处理的数据包总



(a) Heavy overhead traffic: Trace1



(b) Medium overhead traffic: Trace2



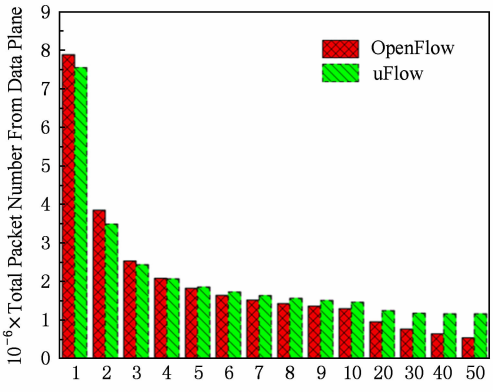
(c) Light overhead traffic: Trace3

Fig. 11 Flow-Mod message optimization ratio with flow table capacity

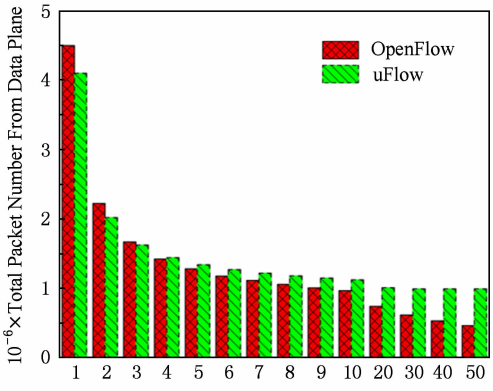
图 11 不同流表容量下对 Flow-Mod 的消息优化比例

量). 从图 12 中可以发现,当流表容量较小时(小于 5 000 条左右),uFlow 系统产生的数据流开销比传统 OpenFlow 网络更小,这是因为 uFlow 消除了大流表项的冗余更新;当流表容量继续增大时,uFlow 系统的数据流开销相对于传统 OpenFlow 网络略有增加,这是由控制器对小流的直接转发会消耗额外的数据流开销所导致. 但可以发现此时数据流开销的总量仍然非常小,这并不会给数据平面和控制平面的交互带来压力.

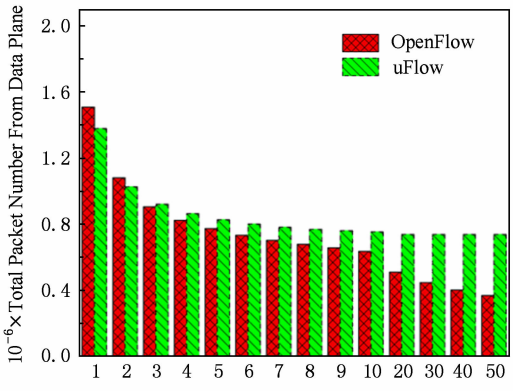
对此,我们对控制平面的交互开销与数据平面的交互开销相加的总开销进行了分析,结果如图 13 所示. 从实验结果过中可以看到,uFlow 系统对 OpenFlow 数据平面与控制平面的交互开销总和仍然有明显的优化效果:当交换机流表容量小于 30 000



(a) Heavy overhead traffic: Trace1



(b) Medium overhead traffic: Trace2

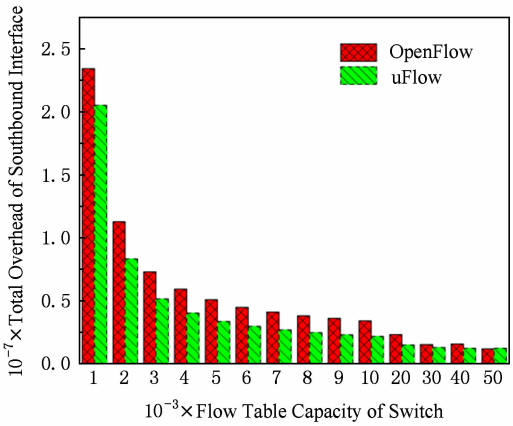


(c) Light overhead traffic: Trace3

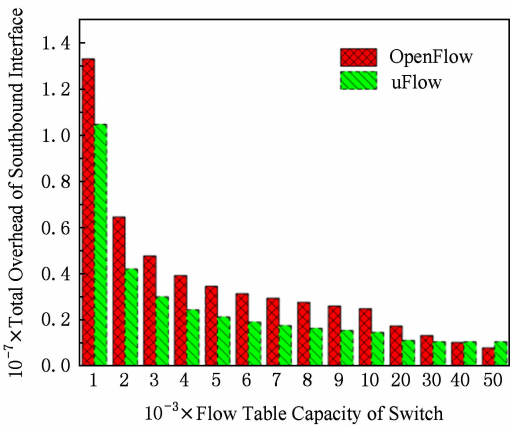
Fig. 12 Data flow overhead of uFlow and OpenFlow

图 12 数据流交互开销对比图

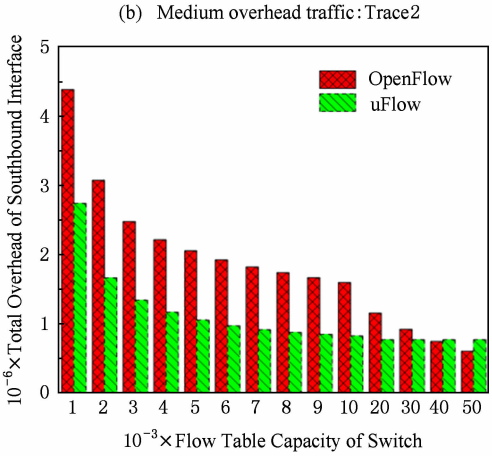
条时,uFlow 对重度、中度、轻度负载流量的交互开销分别能到达 34%,40%,47%的优化效果. 只有当流表容量超过 50 000 条时,uFlow 的交互开销会略高于传统 OpenFlow 网络,但由于我们在实验中使用了目的 IP 的 16b 前缀来区分每条流,网络中所有的流总数量为 2^{16},即 65 500 条,因而 50 000 条的流表容量已经接近网络中所有流的总数量,这在真实的网络设备中是不可能有如此多的资源的.



(a) Heavy overhead traffic: Trace1



(b) Medium overhead traffic: Trace2



(c) Light overhead traffic: Trace3

Fig. 13　The total southbound overhead of uFlow and OpenFlow

图 13　南向接口交互开销总和对比

关于控制器的负载情况,从直观上来看,控制器对小流数据包的直接转发确实增加了控制器的负载,但由于直接转发在其他方面带来的优势,使得控制器总负载不一定会增加. 控制器的负载与数据平面的流量特征有很大的关系,不同的流量情况下控制器负载会有所不同. 具体来说,在流量具备分布不均衡特征的情况下,控制器的总负载甚至很可能由于以下 2 个原因而有所降低:1)大小流区分与控制器转发的方式极大地减少了流表频繁更替导致的控制开销. 控制器需要处理的消息数量,尤其是 Packet-In 消息和 Flow-Mod 消息数量大大减小. 2)控制器处理每条 Packet-In 消息的开销大大减小. 控制器在与数据平面的交互过程中,处理 Packet-In 消息需要控制器计算路由并生成相应的流表项,然后将流表项安装到对应的交换机上,并通过 Packet-Out 消息将数据包再次发送到数据平面. 在本文提出的方法中,控制器在计算路由、下发流表项这 2 个最消耗控制器资源和带来时延的过程均有所简化. 实验的结果也说明,即使在比较坏的情况下,uFlow 系统中控制器处理的消息总数量也不会增加,因此综合来看,控制器的总负载是可控的.

3.3　其他问题

1) 流表开销优化效果

从 3.2 节中对交互开销的实验结果中可以看出,对于相同的流量和相同容量的流表来说,uFlow 系统中流量触发的 Table-Miss 次数以及控制器下发 Flow-Mod 消息数量均有明显的减少. 这说明在同等条件下:①数据平面与控制平面交互带宽一定;②控制器的处理总流量一定,交换机处理同样的流量使用更少的流表资源就可以完成.

我们以图 10 中 Trace1 为例来说明 uFlow 系统对交换机流表资源的优化效果. 如果要求在消耗的 Flow-Mod 消息不超过 2×10^6 的情况下完成对 Trace1 的处理,传统 OpenFlow 网络模型至少需要使用流表容量为 20 000 条的交换机,而使用 uFlow 技术进行开销优化之后,只需要流表容量为 5 000 条的交换机就可以满足设计要求. 从图 11 也可以看出,存在一个最优的交换机流表容量使得南向接口开销最小,这也是值得进一步深入研究的工作之一.

2) 小流时延优化效果

对于时延优化效果,我们在 Mininet[17] 中建立图 1 所示的网络拓扑,分别测试数据包通过控制器转发与通过交换机转发的时延大小,测量得到小流

的平均时延效果如表 2 所示:

Table 2 Optimization of Mice Flow Delay

表 2 小流时延优化实验结果

Scenarios	Average Delay/ms		Optimization Ratio/%
	With uFlow	Without uFlow	
Host *A* to Host *B*	0.344	0.722	52.4
Host *B* to Host *A*	0.328	0.748	56.1

仿真结果显示,uFlow 系统中小流的端到端时延明显小于传统 OpenFlow 网络中对数据包转发处理的时延. 主机 *A*,*B* 之间的小流时延达到了超过 50%的降低幅度,这与小流在网络中经过的转发节点数减小程度基本一致.

我们还测量了通过控制器直接转发情况下的小流转发时延. 实验中使用的控制器配置为 Intel® Core™ i7-2600 @ 3.40 GHz 处理器,16 GB DDR3 内存. 用于构建 SDN 网络的交换机为 ONetSwitch[16]. 为了保持时钟同步,我们在另一台多网口的服务器上运行 2 台虚拟机,作为接入网络的 2 台主机. 多次测量结果显示通过控制器直接转发数据包的时延平均值为 1.6 ms.

3) 部署方式与兼容性

uFlow 系统可以直接部署于控制核心服务中,不会对数据平面做任何修改,运行时对其他控制平面的应用保持透明,因此不会影响其他控制器应用的正常功能. 这说明 uFlow 系统与其他控制器应用能够保持良好的兼容性,例如控制器对于小流数据包的直接转发需要遵循访问控制规则,通过查询访问控制表来实现;而对于负载均衡应用,uFlow 系统也可以在正常处理数据包后再做负载均衡处理,避免了数据包的处理冲突.

4 总结与结论

本文提出一种优化的 SDN 流量处理模型 uFlow,在控制器上实现对数据流特征信息的记录、识别和转发. 弥补了控制器在下发新的流表项时,缺乏新表项对应的数据流特征信息的不足. 控制器对识别后的大小流分开处理,将种类繁多但是总流量较少的小流数据包直接送达目的地,减少了小流的传输时延,消除潜在的流表更新振荡和冗余的 Packet-In 消息,并且节省了交换机的流表空间. 同时本文也通过真实流量对 uFlow 系统的效果进行了验证,uFlow 系统与传统 OpenFlow 网络相比,能显著的减少数据包的时延,降低控制平面和数据平面的交互开销,节省交换机的流表项.

参 考 文 献

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

[2] Curtis A R, Mogul J C, Tourrilhes J, et al. DevoFlow: Scaling flow management for high-performance networks [J]. ACM SIGCOMM Computer Communication Review, 2011, 41(4): 254-265

[3] Kotani D, Okabe Y. A packet-in message filtering mechanism for protection of control plane in OpenFlow networks [C] //Proc of the 10th ACM/IEEE Symp on Architectures for Networking and Communications Systems. New York: ACM, 2014: 29-40

[4] Wang Binfeng, Su Jinshu, Chen Lin. Review of the design of data center network for cloud computing [J]. Journal of Computer Research and Development, 2016, 53(9): 2085-2106 (in Chinese)
(王斌锋, 苏金树, 陈琳. 云计算数据中心网络设计综述[J]. 计算机研究与发展, 2016, 53(9): 2085-2106)

[5] Sarrar N, Uhlig S, Feldmann A, et al, Leveraging Zipf's law for traffic offloading [J]. ACM SIGCOMM Computer Communication Review, 2012, 42(1): 16-22

[6] Sarrar N, Feldmann A, Uhlig S, et al. Towards hardware accelerated software routers [C] //Proc of the 8th ACM CoNEXT Student Workshop. New York: ACM, 2010: No. 2

[7] Advait D, Fang Hao, Sarit M, et al. Elasticon: An elastic distributed SDN controller [C] //Proc of the 10th ACM/IEEE Symp on Architectures for Networking and Communications Systems. New York: ACM, 2014: 17-28

[8] Open Networking Foundation. OpenFlow Switch Specification Version. 1.3.2 [S/OL]. 2012 [2017-06-12]. https://www.opennetworking.org/software-defined-standards/specifications

[9] Marcial P F. Comparing openflow controller paradigms scalability: Reactive and proactive [C] //Proc of the 27th IEEE Advanced Information Networking and Applications. Piscataway, NJ: IEEE, 2013: 1009-1016

[10] Wallner R, Robert C. An SDN approach: Quality of service using big switch's floodlight open-source controller [C] //Proc of the 35th Asia-Pacific Advanced Network. Hong Kong: APAN, 2013: 14-19

[11] Jan M, Robert V, Anton T, et al. Opendaylight: Towards a model-driven SDN controller architecture [C] //Proc of the 15th Int Symp on World of Wireless, Mobile and Multimedia Networks. Piscataway, NJ: IEEE, 2014: 1-6

[12] Theophilus B, Aditya A, David A M. Network traffic characteristics of data centers in the wild [C] //Proc of the 10th ACM SIGCOMM Conf on Internet Measurement. New York: ACM, 2010: 267-280

[13] Kim T, Lee T, Kim K H, et al. An efficient packet processing protocol based on exchanging messages between switches and controller in OpenFlow networks [C] //Proc of the 10th IEEE Int Conf and Expo on Emerging Technologies for a Smarter World. Piscataway, NJ: IEEE, 2013: 1-5

[14] Katta N, Omid A, Jennifer R, et al. CacheFlow: Dependency-aware rule-caching for software-defined networks [C] //Proc of the 2nd Symp on SDN Research. New York: ACM, 2016: 1-12

[15] Alcock S, Lorier P, Nelson R. Libtrace: A packet capture and analysis library [J]. ACM SIGCOMM Computer Communication Review, 2012, 42(2): 42-48

[16] Hu Chengchen, Yang Ji, Gong Zhimin, et al. DesktopDC: Setting all programmable data center networking testbed on desk [J]. ACM SIGCOMM Computer Communication Review, 2014, 44(2): 593-594

[17] Oliveira R L S, Shinoda A A, Schweitzer C M, et al. Using mininet for emulation and prototyping software-defined networks [C] //Proc of the 7th Colombian Conf on Communications and Computing. Piscataway, NJ: IEEE, 2014: 1-6

Zheng Peng, born in 1992. PhD candidate. His main research interests include software-defined networking.

Hu Chengchen, born in 1981. PhD, professor and PhD supervisor. His main research interests include network measurement, cloud data center networking, software defined networking.

Li Hao, born in 1987. PhD, assistant professor. His main research interests include network measurement, deep semantics in the traffic flowing in the network(hao. li@mail. xjtu. edu. cn).

2018 年《计算机研究与发展》专题(正刊)征文通知

——数据挖掘前沿进展

在当前大数据时代,海量数据的挖掘和分析尤为重要,数据挖掘技术在媒体、金融、医疗、交通、电商等领域都取得了广泛的应用.但是,大数据的复杂多样性以及数据挖掘技术在各行业应用的特殊性也为数据挖掘提出了新的理论和技术挑战.2018 年《计算机研究与发展》"数据挖掘前沿进展"专辑将重点关注数据挖掘理论进展以及在各行业应用的技术前沿,征集国内研究同仁在上述问题取得的研究成果.

征文范围(但不限于)

- 数据挖掘理论与方法:数据挖掘和机器学习新理论新方法,分类、聚类、集成学习、强化学习、关联分析、链接分析、频繁模式挖掘、动态数据挖掘、并行与分布式挖掘、大规模数据挖掘等领域的理论和方法;
- 特定数据类型的挖掘和分析:关系数据挖掘、图模式挖掘、多媒体数据挖掘和分析、社交网络数据挖掘、文本挖掘、隐私保护数据挖掘、生物信息数据挖掘、推荐系统、数据仓库等;
- 数据挖掘技术应用:数据挖掘技术在金融、管理、市场营销、生物信息、教育、旅游、电子商务、电信、社会网络等领域和行业应用的新技术和新方法.

征文要求

1) 论文应属于作者的科研成果,数据真实可靠,具有重要的学术价值与推广应用价值,且未在国内外公开发行的刊物或会议上发表,不存在一稿多投问题.作者在投稿时,需向编辑部提交版权转让协议.
2) 论文一律用 Word 格式排版,论文格式体例参考近期出版的《计算机研究与发展》的要求(http://crad. ict. ac. cn/).
3) 论文须通过期刊网站(http://crad. ict. ac. cn)投稿,投稿时提供作者的联系方式,留言中务必注明"数据挖掘 2018 专题"(否则按自由来稿处理).
4) 论文预录用后,至少有一位作者必须注册 CCDM2018(http://ccdm2018. sdufe. edu. cn/index. htm),并到 CCDM2018 做口头报告.否则,视为退出专刊.

重要日期

征文截止日期: 2018 年 3 月 31 日　　预录用通知日期: 2018 年 5 月 15 日

修改稿提交日期:2018 年 5 月 31 日

特邀编委

尹义龙　教授　山东大学　ylyin@sdu. edu. cn

钱宇华　教授　山西大学　jinchengqyh@sxu. edu. cn

联系方式

编　辑　部:crad@ict. ac. cn,010-62620696,010-62600350

通信地址:北京 2704 信箱《计算机研究与发展》编辑部　　邮政编码:100190