DOI:10.7544/issn1000-1239.2014.20130852
51(11):2448-2457,2014

OpenFlow 网络冗余控制报文消除机制研究

左青云　陈　鸣　丁　科　邢长友　张国敏　许　博
(解放军理工大学指挥信息系统学院　南京　210007)
(mingchennj@163.com)

Eliminating Redundant Control Messages in OpenFlow Networks

Zuo Qingyun, Chen Ming, Ding Ke, Xing Changyou, Zhang Guomin, and Xu Bo
(*College of Command Information Systems, PLA University of Science and Technology, Nanjing* 210007)

Abstract　The data plane of OpenFlow networks sends packets which don't match any flow entry to the controller, and in such case connectionless burst traffic is encapsulated as control messages, until the data plane receives a response message from the controller. This process produce redundant control messages for both the control and data planes, which influences the performance of the entire network. However, current OpenFlow protocol doesn't give definite explanation or handle this problem. In this paper, we propose two approaches of eliminating redundant control messages on the control plane (referred as ERCMC) and on the data plane (referred as ERCMD). ERCMC constructs latest packet-in message view (referred as LPMV) on the controller to mark the latest packet-in messages, so as to eliminate redundant control messages on the control plane. ERCMD adds marks directly on the data plane, so that the burst packets are directly buffered and never encapsulated as control messages. We implement these two approaches in NOX and Open vSwitch respectively, and evaluate the performance of the two approaches. The results of our experiments show that ERCMC can eliminate redundant control messages, but it will add extra processing overheads; ERCMD can not only eliminate redundant control messages, but also alleviate the load of the controller and OpenFlow switches.

Key words　OpenFlow network; OpenFlow protocol; control messages; UDP flow; controller

摘　要　OpenFlow 网络数据平面将未匹配流表的数据包发送给控制器,其中的无连接突发流量将产生冗余控制报文,对网络性能造成不良影响,而目前的 OpenFlow 协议并未对此进行处理.研究了在控制平面和数据平面分别消除冗余控制报文的方法 ERCMC(eliminating redundant control messages on the control plane)和 ERCMD(eliminating redundant control messages on the data plane),分别在 NOX 和 Open vSwitch 上进行实现,并进行性能评价.实验结果表明,ERCMC 方法能够消除冗余控制报文,但增加了额外的处理开销;ERCMD 方法在减少冗余控制报文数量的情况下能够减小控制器和 OpenFlow 交换机负载.

关键词　OpenFlow 网络;OpenFlow 协议;控制报文;UDP 流;控制器

中图法分类号　TP393

收稿日期:2013-06-19;**修回日期**:2014-02-26

基金项目:国家"九七三"重点基础研究发展计划基金项目(2012CB315806);国家自然科学基金项目(61379149,61070173,61103225)

通信作者:陈　鸣(mingchennj@163.com)

软件定义网络(software-defined networking, SDN)[1]将网络控制平面从传统的分布式网络设备中独立出来,使得网络管理员能够通过在控制器上编制软件来灵活地控制和部署网络功能,实现了网络的可编程性,加速了网络创新. OpenFlow[2]作为软件定义网络 SDN 控制平面和数据平面之间的一种接口标准,是当前学术界和工业界公认的一种实践 SDN 理念的范例.

目前,控制平面的扩展性是 OpenFlow 网络的研究热点之一. 该问题的主要根源在于控制器的集中管控容易造成性能瓶颈,相关的研究包括 DIFANE[3], DevoFlow[4], HyperFlow[5], Onix[6], Kandoo[7]等. 控制报文包含很多种,其中最频繁的两种控制报文为数据平面的流请求报文和控制平面定期获取的统计报文. 统计报文的数量取决于控制器应用程序的具体设置,因此流量大小是可控的;流请求报文与端主机产生的流数量和当前的网络规模有关,网络规模越大流请求报文也越多. 对于面向连接的数据流(如 TCP 流)而言,每条流首先要发送 1 个连接请求报文(如 TCP 的 SYN 报文),当控制器接受该请求后就会在交换机上安装相应流表项,这种情况下流请求报文与流的数量相等,不会产生冗余控制报文. 然而对于无连接的数据流(如 UDP 流),OpenFlow 交换机可能在短时间内收到大量未匹配数据包,从而使得控制器接收到大量冗余控制报文,对控制器和 OpenFlow 交换机的性能同时造成不良影响. 值得注意的是,目前的 OpenFlow 协议(1.3 版本)并未明确提出相关处理机制.

在此基础上,本文分析了冗余控制报文产生的原因,研究了消除冗余控制报文的相关机制,提出了分别在控制平面和数据平面消除冗余控制报文的方法 ERCMC(eliminating redundant control messages on the control plane)和 ERCMD(eliminating redundant control messages on the data plane). ERCMC 方法通过在控制器实现最新 packet-in 报文视图(latest packet-in message view, LPMV),使得冗余控制报文在控制器端直接被丢弃,从而消除冗余控制报文. 这种方法属于"半程"优化方法,即通过软件方式消除冗余控制报文,符合软件定义网络的初衷. ERCMD 方法直接在 OpenFlow 交换机上实现,能够直接在源端减少冗余控制报文,减轻控制器负载,性能更好,但需要增加 OpenFlow 交换机的流识别功能.

1 相关工作

在 OpenFlow 网络中,流表的安装方式包括 2 种:主动模式(proactively)和被动模式(reactively). 主动模式是指控制器主动将流表部署到 OpenFlow 交换机中,以避免流表项的临时建立过程,一般用于包含通配符、匹配粒度较粗的流表项;被动模式是指数据包到达 OpenFlow 交换机后,再通过控制器实现流表安装的操作,一般应用于匹配粒度较细、需要实现动态调度的流表项. 从 OpenFlow 网络目前的应用平台来看,被动模式在实际的控制逻辑下发过程中不可避免,符合实际网络应用的需求,也更适应网络数据包的动态转发过程. 当网络中大部分流表的建立安装过程由被动模式产生时,集中管控的控制器很容易成为性能瓶颈.

目前针对 OpenFlow 网络的可扩展性研究主要分为纵向扩展和横向扩展[8] 2 个方面. 纵向扩展主要通过给 OpenFlow 交换机增加部分控制功能,从而减少控制器和 OpenFlow 交换机之间的信息交互,分担控制器的处理开销,提高控制器的可扩展性能. DIFANE 和 DevoFlow 都属于这种方式. DIFANE 对流表规则进行区域划分,通过权威交换机来实现规则安装功能;DevoFlow 则将采样、触发报告和近似统计等功能增加到 OpenFlow 交换机中,从而减少控制器定期收集的流表统计信息. 横向扩展则通过多控制器的分布式管控平面,实现 OpenFlow 网络的分域管理,并利用相对成熟的分布式系统解决方案来实现控制平面的一致性需求,从而实现整个控制平面的可扩展性. Onix 和 HyperFlow 属于横向扩展方式.

本文的方法属于纵向扩展. 和上述 2 种方法相比,DIFANE 侧重于将流请求报文导向权威交换机,减少数据平面和控制平面的信息交互;DevoFlow 侧重于减少数据平面统计报文的产生. 本文针对的是无用的冗余控制报文,是对 OpenFlow 处理机制的优化,在实现性质上类似 NOX-MT[9]对控制器 NOX[10]的优化.

2 冗余控制报文分析

当新的数据流到达 OpenFlow 交换机时如果未匹配任何流表项,OpenFlow 交换机将对其进行封装,生成流请求报文(packet-in message)并通过

OpenFlow 安全信道发送给控制器. 控制器收到 packet-in 报文,首先对每个报文进行解析和路由运算,然后向该流对应路径上的所有 OpenFlow 交换机发送流操作报文(flow-mod 报文,包括流增加、修改和删除操作). 随后,当每个 flow-mod 报文到达 OpenFlow 交换机时,若 flow-mod 报文设置了 OFPFF_CHECK_OVERLAP 位,则需要首先检查现有流表中是否存在重叠流表项,当存在重叠流表项时 OpenFlow 交换机将向控制器返回错误代码;若未设置该位,OpenFlow 交换机将删除重叠的流表项,并插入新的流表项,同时将超时时间和计数器清零(有可能造成流表统计信息的偏差).

冗余控制报文产生的原因,来源于 OpenFlow 标准中规定的,未匹配流表的数据包将转发给控制器. 在目前的 OpenFlow 标准(1.3 版本)中,并没有要求针对每条流仅发送第 1 个数据包. 当 OpenFlow 交换机流表安装请求时延过长,非连接数据流的大量数据包都将被封装为 packet-in 报文发送给控制器. 控制器单位时间处理 packet-in 报文的能力与控制器部署的硬件环境和控制器软件有关. 因此大量冗余控制报文的产生将给控制器增加处理开销,从而影响时延敏感型应用的性能. 控制器在处理这些冗余 packet-in 报文后将生成 flow-mod 报文,这也将引发 OpenFlow 交换机对冗余 flow-mod 报文的进一步处理. 具体流程如图 1 所示,当主机 A 产生 1 条非连接数据流时,由于安装流表时延的影响,在 flow-mod 报文到达 OpenFlow 交换机前,大量冗余的控制报文将发送给控制器.

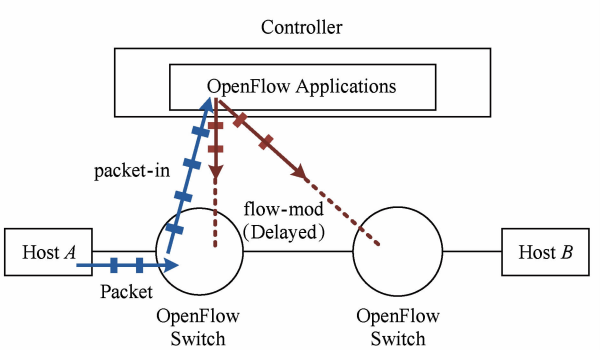


Fig. 1 Processing of redundant control messages.

图 1 冗余控制报文处理过程

实际上,在真实网络中流的数量是非常多的,尤其在面向 OpenFlow 应用的数据中心中,控制器的处理时延和大量流请求报文的排队时延都无法忽略不计. 在广域网中部署控制器还需要考虑网络传输过程中的传播时延和传输时延,这进一步增加了流表安装过程的往返时延. 为了验证 OpenFlow 交换机和控制器之间往返时延的大致范围,并定量分析非连接数据流产生的冗余报文数量,我们构建如图 1 所示的拓扑环境,其中 OpenFlow 交换机由 Open vSwitch[11] 实现,控制器由 NOX 实现,OpenFlow 交换机和控制器之间通过实验室真实生产网络下的二层硬件交换机以带外模式进行连接. 具体的软硬件环境如表 1 所示:

Table 1 The Software and Hardware Configuration in the Lab

表 1 实验环境软硬件配置

Device	Configuration
NOX Controller	Dual-Core CPU 2.5 GHz, 2 GB memory, 1 Gbps NIC, CentOS 6.0, NOX 0.9.1
OpenFow Switch	Dual-Core CPU 2.8 GHz, 2 GB memory, 1 Gbps NIC, Ubuntu 10.04, Open vSwitch 1.9
Hardware Switch	Cisco 2960

首先,我们通过 cbench[12] 工具对流请求报文的平均往返时延进行测量. cbench 通过仿真一定数量的 OpenFlow 交换机向控制器发送 packet-in 报文,并根据响应报文(flow-mod)来记录控制器的实际性能. 默认情况下,cbench 支持 2 种模式的操作:时延模式和吞吐量模式. 在时延模式中,每 1 个仿真的交换机向控制器每次只发送 1 个流建立请求,在收到答复之后再发送下 1 个请求;在吞吐量模式中,每个仿真的交换机发送尽可能多的 packet-in 报文,直到耗尽 TCP 缓存.

为了测量不同数量 OpenFlow 交换机与控制器进行交互时的往返时延,我们进行 6 组时延模式的测量实验,分别为 10 台、20 台、30 台、40 台、50 台、60 台 OpenFlow 交换机向 NOX 控制器发送 packet-in 报文. 每组实验 16 个周期,每个周期持续 5 s. 第 1 个周期是控制器的热身时间,因此将去掉这组数据. 每组实验均采用 10 万个唯一的 MAC 地址(代表 10 万个仿真的端主机). 图 2 给出了 6 组实验中每组控制报文的往返时延累计分布图. 可以看出,packet-in 报文的平均往返时延在 2~10 ms 之间. 随着 OpenFlow 交换机数量的增加,packet-in 报文的平均往返时延也不断增加. 这是由于控制器性能所限,packet-in 报文在控制器处形成了排队导致时延增加. 而据文献[4,13]所述,流表安装时延在 5 ms 左右,但一般不会超过 10 ms. 如果考虑到广域网中的传播延迟,如文献[14]提到的 OpenFlow 交换机和控制器之间的延迟甚至将达到 10~200 ms,这对时延敏感的流

表安装操作,尤其当产生大量非连接的数据流时,OpenFlow 网络性能将受到极大影响.

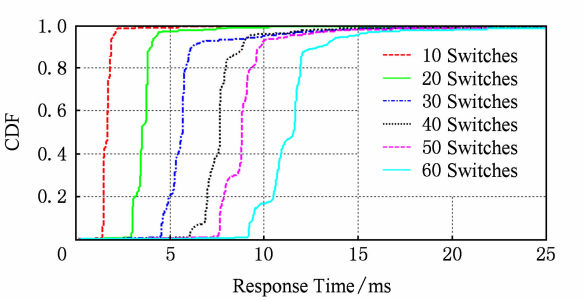
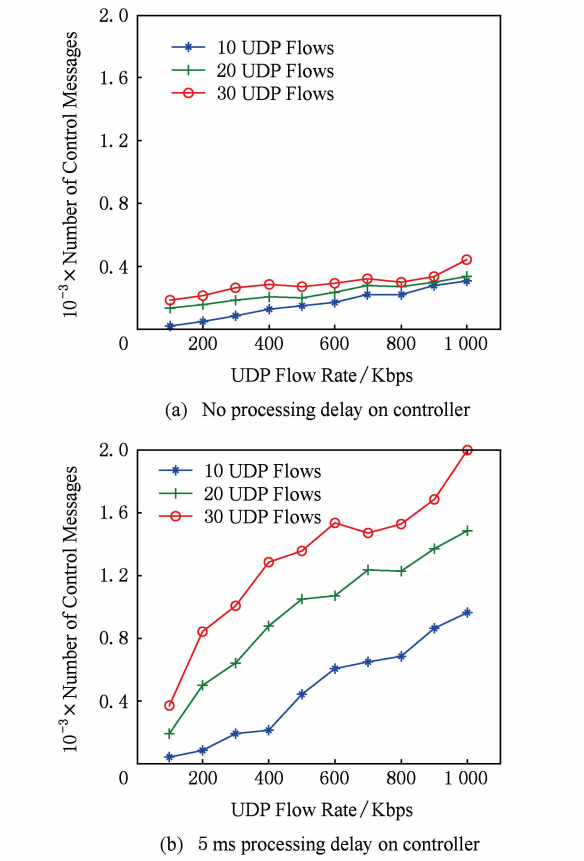


Fig. 2 Response time CDF of packet-in messages.

图 2 控制报文响应时间累计分布图

为定量分析非连接数据流产生的冗余报文数量,我们通过主机 A 向主机 B 发送 UDP 数据流,其中 UDP 数据包大小为 400 B. 控制器端应用程序在收到 packet-in 报文后进行计数,并向 OpenFlow 交换机安装 A 到 B 的流表路径. UDP 流的数量分别为 10,20,30 条,并以不同的速率进行发送. 由于我们的部署环境相对简单,控制器处理时延可忽略不计,我们测得的往返时延在 3 ms 左右波动. 另外,我们在给控制器增加 5 ms 处理时延的情况下,测量实际的控制报文数量. 为了减小误差,每组实验进行 10 次,结果取平均值,如图 3 所示.



(a) No processing delay on controller

(b) 5 ms processing delay on controller

Fig. 3 The number of control messages in different scenarios.

图 3 不同场景下控制报文数量

从图 3 可以看出,随着 UDP 流发送速率的增大和 UDP 流数量的增加,控制报文的数量也不断增加. 实际上,在图 3(a)中,当 10 条 UDP 流的发送速率为 100 Kbps 时,平均控制报文数量为 12,即仅产生了 2 个冗余控制报文;而 30 条 UDP 流分别发送 1 000 Kbps 的数据包时,平均控制报文数量达到了 444,即冗余控制报文数量为 414. 当图 3(b)控制器增加了 5 ms 处理时延后,冗余控制报文的数量则急剧增加. 可以预见的是,若控制器处于高负载下,大量的非连接数据流产生的冗余控制报文将进一步对控制器的性能造成影响.

3 方法设计

根据目前的 OpenFlow 标准(1.3 版本),默认情况下,未匹配流表的数据包首先在 OpenFlow 交换机上进行缓存,随后报头的前 128 B 将封装在 packet-in 报文中转发给控制器. 既然数据包都将缓存在 OpenFlow 交换机上,那么冗余的控制报文完全可以避免发送. 当然,这需要在 OpenFlow 交换机上对未匹配数据包进行识别和标记,需要改动 OpenFlow 的处理机制,相当于给 OpenFlow 交换机增加部分控制功能. 为此,我们首先考虑在控制器端实现冗余控制报文的消除方法 ERCMC,然后再考虑如何在数据平面利用 OpenFlow 自身的机制来实现冗余控制报文消除方法 ERCMD.

我们的实现方法以第 2 节中的实验环境为基础. 控制器是 NOX(版本 0.9.1),它是基于 OpenFlow 的 SDN 最早实现的控制器软件,目前很多控制器的实现都以它为基础和模板. 针对 OpenFlow 交换机的实现我们选择的是 Open vSwitch(版本 1.9),它是目前应用最广泛的软件 OpenFlow 交换机. 因此,选择这 2 种通用的 OpenFlow 平台实现冗余控制报文的消除具有一般性和通用性.

3.1 控制器实现

ERCMC 方法的实现需要通过软件方式实现对冗余报文的识别和标记功能. 当冗余报文到达控制器,路由应用程序在生成 flow-mod 报文之前,标记这个流表项的特征. 为此,我们在控制器 NOX 中维护一张最新 packet-in 报文视图(latest packet-in

message view, LPMV),上面记录了所有 OpenFlow 交换机最新发送过来的控制报文信息. 在冗余控制报文到达后,控制器能够通过查询 LPMV 检测该控制报文是否已经执行过流表安装操作.

为了实现快速的 LPMV 查询功能,我们将 LPMV 保存在一张 Hash 表中. 每个 packet-in 报文在控制器上进行解析并 Hash 后,首先查询 LPMV. 若 LPMV 中没有该 Hash 值,则将该控制报文对应的流信息保存在 LPMV 中;若检测到该报文是冗余控制报文,则直接作丢弃处理. 这种情况下,每个报文到达控制器后只需进行一次 Hash 生成和查找工作,而无需再进行路由计算和相应的流表安装操作. OpenFlow 网络中 ERCMC 方法实现的大致结构如图 4 所示:

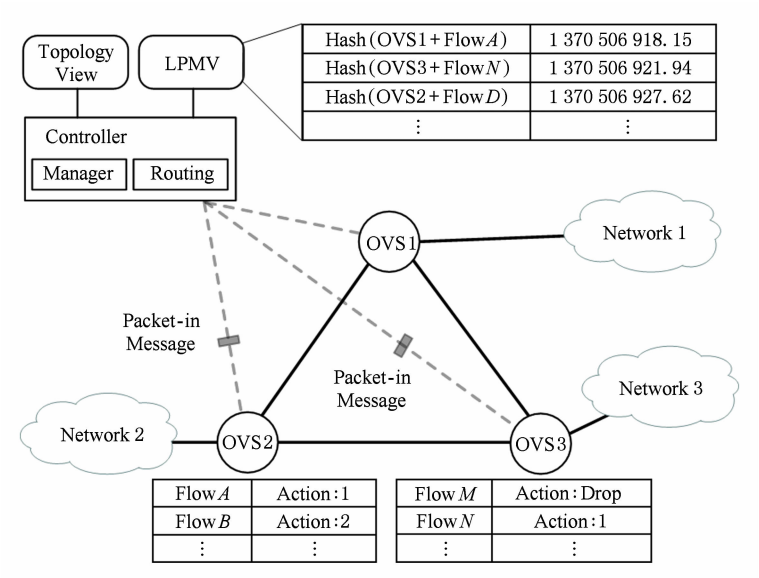


Fig. 4 The implementation of ERCMC approach.

图 4 ERCMC 方法实现

这里需要注意的是,OpenFlow 交换机中的流表项在删除后,该流表项对应的数据流将重新发送到控制器,此时若 LPMV 中还保存有对应的 Hash 值,则导致无法生成正常的安装流表操作. 考虑到 Open vSwitch 中流表项的最短超时时间为 5 s(数据通道内核态超时时间),因此我们在保存每个控制报文的 Hash 值时,将当前时刻保存下来,同时设置 LPMV 中这个 Hash 值的超时时间为 2 s(流请求报文在控制器和 OpenFlow 交换机之间的往返时延很难超过 2 s). 在这种情况下,短时间涌入控制器的冗余控制报文将能够直接检测出来,而在流表项安装成功后,该流的 packet-in 报文将至少在 5 s 后才能到达控制器,此时 LPMV 中该报文对应的 Hash 值已经由于超时而被删除. 具体处理流程如算法 1 所示. 行④~⑥表示将报文的流特征和对应的 OpenFlow 交换机 ID(即 dpid)放在一起取 Hash 值,这是因为任何一个 OpenFlow 交换机有可能产生同样的 packet-in 报文,增加 dpid 字段能够进行有效区分. 行⑩表示取当前时间,以实现 LPMV 的超时检查机制.

算法 1. ERCMC 方法实现流程.

① $H \leftarrow \varnothing$;
② for each incoming message p do
③ if p is packet-in message then
④ extract flow feature F from p;
⑤ record dpid D of p;
⑥ $h \leftarrow hash(D+F)$;
⑦ if $h \in H$ then
⑧ continue;
⑨ else
⑩ $t \leftarrow time()$;
⑪ $H \leftarrow H \cup \langle h, t \rangle$;
⑫ end if
⑬ end if
⑭ end for

3.2 Open vSwitch 实现

在 NOX 上添加 LPMV 的方法无法在根源上解决冗余控制报文的方法. 我们通过在 Open vSwitch 上增加对未匹配数据包的识别和标记,实现了 ERCMD 方法.

Open vSwitch 的实现主要分为 2 部分:用户空间进程和内核模块. 用户空间进程(ovs-vswitchd)用于维护和控制器的连接,保存流表转发规则,并管理 Open vSwitch 的 datapath 端口. 内核模块实现交换机的 datapath 功能,接管从网卡接收到的数据包进行处理,实现数据包的快速转发. Open vSwitch 内部实现流程如图 5 所示. 在 Open vSwitch 中,针对数据包的处理分为快路径和慢路径. 内核模块实现对数据包的快速处理,维护一张精确匹配的快路径转发表,超时时间为固定的 5 s. 用户空间进程属于慢路径处理流程,它维护一张规则表(rule),即包含通配符的流表规则. 用户空间进程还包含一张精确匹配的慢路径转发表,负责维护慢路径和快路径的流表映射,为一对多的映射关系.

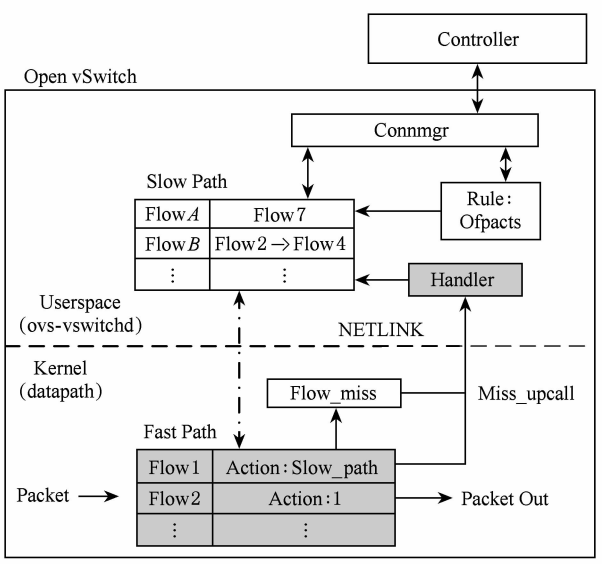


Fig. 5 The internal process of Open vSwitch.

图 5 Open vSwitch 内部处理流程

当内核模块接收到数据包时,它首先查找快路径转发表,若匹配则按相应操作转发;若未匹配(flow_miss)或操作指向慢路径(slow_path),则数据包将通过 Linux 进程间通信机制 NETLINK 发送给用户空间进程. 这里的慢路径操作主要指几种需要由用户空间进程进行处理的数据包,包括 CFM,LACP,STP 等协议数据包和需要发往控制器处理的数据包. 因此,当某条突发流产生的大量数据包到达快路径,将会触发流表不匹配(flow_miss)事件而发送给用户空间进程,随后将被用户空间进程封装为 packet-in 报文发送给控制器. 同时,在控制器流表安装操作还未返回前,内核模块快路径流表将安装操作指令为 slow_path 的流表项,即将该流数据包直接导向用户空间. 很显然,这些突发流不管是由于流表不匹配,还是由于流表操作指令为慢路径,都被用户空间进程在缓存数据包后发送给控制器. 因此,我们主要通过修改快路径的流表和用户空间进程的相关处理代码(我们称之为 handler,实际过程较复杂),来实现冗余控制报文的消除功能. 所作修改的模块在图 5 中用阴影标出,下面给出修改后 handler 的大致流程,如算法 2 所示:

算法 2. Open vSwitch 内部 handler 实现流程.

① for each incoming packet p from kernel do
② extract flow feature F from p;
③ if F not match in slow path then
④ lookup rule R to match F;
⑤ create flow A in slow path;
⑥ if $R.action=$ *OFPACT_CONTROLLER* then
⑦ $A.slow \leftarrow slow\_controller$;
⑧ send p to the controller;
⑨ $A.has\_sent \leftarrow$ true;
⑩ end if
⑪ install flow A in fast path;
⑫ else
⑬ lookup flow A to match F;
⑭ if $A.has\_sent=$ true then
⑮ continue;
⑯ end if
⑰ end if
⑱ end for

算法行③~⑤表示若数据包中释放出来的流特征未在慢路径中得到匹配,则根据 rule 的操作规则在慢路径中创建流表项. 行⑥~⑩选出需要交给控制器处理的流表项,将其慢路径原因标记为控制器处理,随后发送给控制器,并标记其已经发送. 由于慢路径和快路径流表的映射关系,行⑪表示将慢路径流表项安装到快路径中,其流表项操作指令为慢路径. 行⑬~⑯表示在后续数据包到达时,查看该流是否已经发送给控制器,若已经发送则跳过该数据包的处理阶段,在这种情况下数据包只做缓存处理.

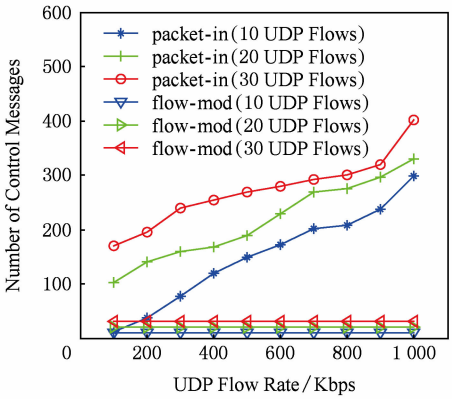
4 性能评价

为了比较本文提出的 ERCMC 和 ERCMD 方法的实际效果,我们将未作修改的 OpenFlow 网络 NOX 和 Open vSwitch 正常处理流程和 ERCMC,ERCMD 方法进行对比,主要从控制报文数量、控制器吞吐量和 CPU 利用率 3 方面进行统计. 实验的部

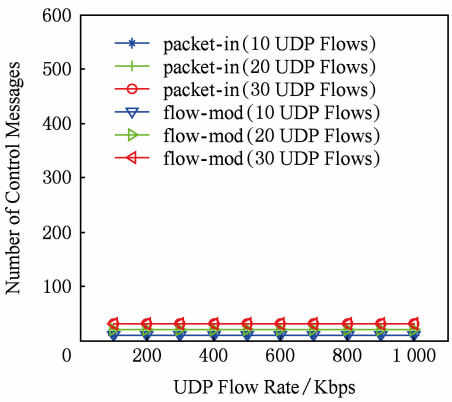
署以第 2 节中的实验环境为基础,并测量不同流量背景下的相关性能参数.

4.1 控制报文数量

ERCMC 和 ERCMD 方法的目的在于减少冗余控制报文数量,因此我们首先统计在这 2 种方法下控制报文的数量.实验的流量设置与第 2 节一样,即端主机分别发送 10,20,30 条 UDP 流,并使发送速率从 100 Kbps 增加到 1 000 Kbps.由于在控制器或往返链路中增加时延对这 2 种方法的实现并无影响,因此下面的实验没有增加往返时延和处理时延.为了区分 ERCMC 和 ERCMD 方法,我们同时统计 packet-in 报文和 flow-mod 报文的数量.实验结果如图 6 所示:



(a) ERCMC



(b) ERCMD

Fig. 6 The number of control messages for ERCMC and ERCMD.

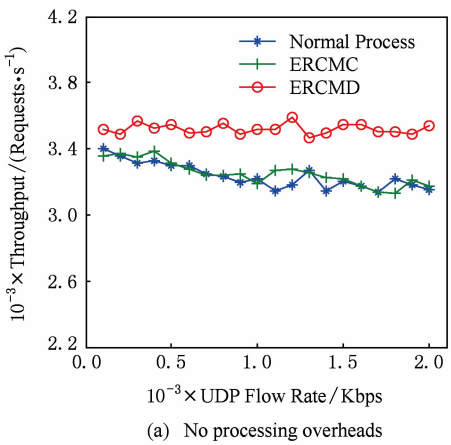
图 6 ERCMC 和 ERCMD 控制报文数量

在 ERCMC 方法中,由于冗余控制报文的消除发生在控制器,因此 packet-in 报文的数量随着流量的增加而增多,而 flow-mod 报文数量与流的数量相等.在 ERCMD 算法中,packet-in 报文和 flow-mod 报文都与流的数量相等,即未产生任何冗余控制报文,这也是直接在 Open vSwitch 上消除冗余控制报文的结果.

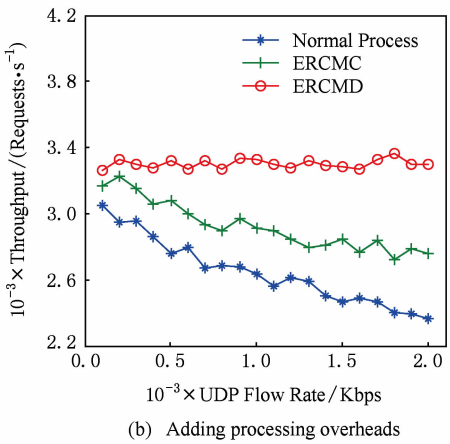
4.2 控制器吞吐量

冗余控制报文的产生必将对控制器的性能造成影响,下面我们将正常处理流程(未消除冗余控制报文)、ERCMC 和 ERCMD 3 种方法进行比较.为了最大化比较不同方法之间的差异,我们重复 4.1 节中 30 条 UDP 流的实验,并设置每秒钟产生 30 条新的 UDP 流,发送速率从 100 Kbps 增加到 2 000 Kbps,每组流量维持的时间为 16 s.在不同的 UDP 流发送速率下,我们使用 cbench 工具时延模式来测量这 3 种不同方法下控制器的实际吞吐量,即每秒钟能够处理的流请求数量.

另一方面,为了比较控制器处理时延对不同方法的影响,在相同的实验环境下,我们适当增加 UDP 流产生的控制报文在控制器端的处理开销,并重新测量这 3 种方法的吞吐量,以比较处理开销对不同方法的影响.在实验中,每组不同的 UDP 流量实验进行 5 组,取平均值.通过 cbench 仿真软件仿真 16 台 OpenFlow 交换机,设置 10 万个唯一的 MAC 地址.实验结果如图 7 所示:



(a) No processing overheads



(b) Adding processing overheads

Fig. 7 Comparison of throughput in different scenarios.

图 7 不同场景下吞吐量比较

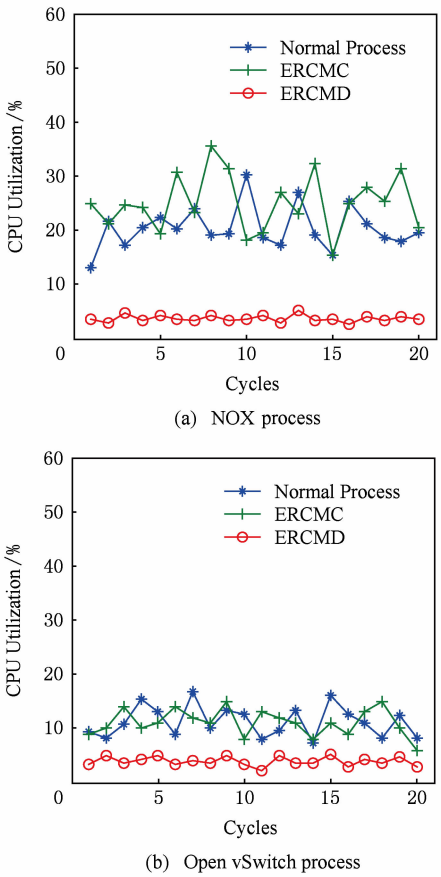
在图 7(a)未增加控制报文的处理开销时,ERCMD 方法吞吐量最高,并基本保持稳定.而正常处理流程和 ERCMC 方法的吞吐量基本一致,并维持下降趋势.这是因为 ERCMD 方法在 UDP 流数量一定的情况下控制报文数量不变,因此性能基本维持不变.正常处理流程需要处理冗余控制报文,由于这里的控制报文处理方式仅是生成 flow-mod 报文,开销不大,因此吞吐量保持缓慢下降状态.ERCMC 方法和正常处理流程的吞吐量基本一致,这是由于 ERCMC 方法需要在解析每个报文后,对报文头部字段进行提取组装并进行 Hash 操作,增加了每个报文的处理开销,使得其吞吐量也保持缓慢下降趋势.在图 7(b)中增加了控制器对于每个控制报文的处理开销后,随着 UDP 流发送速率的增大,ERCMD 方法的吞吐量有所减小但依然保持稳定状态,而正常处理流程和 ERCMC 方法吞吐量大幅下降,但 ERCMC 方法在这种情况下比正常处理流程性能要好.这是由于增加了报文的处理开销后,正常处理流程需要承担每个报文的处理开销,因此吞吐量急剧下降.ERCMC 方法虽然要针对每个报文进行 Hash 处理,但消除了冗余控制报文的处理开销,因此吞吐量比正常处理流程要高.

可以看出,ERCMD 方法能够从根本上消除冗余控制报文,减小控制器的开销,从而提高控制器性能的可扩展性.通过 ERCMC 这种轻量级的软件实现方法虽然消除了冗余控制报文,但增加了每个报文的处理判断开销,需要考虑实际的应用需求,如网络中 UDP 流较多、控制器对控制报文进行深度报文检测等控制报文处理开销较大的应用.

4.3 CPU 利用率

为了进一步比较 ERCMC 和 ERCMD 方法对控制器和 OpenFlow 交换机性能的影响,我们在 4.2 节实验中某种特定流量情况下(每秒生成 30 条 UDP 流,流的发送速率为 1 500 Kbps)测量控制器 NOX 进程(nox_core)和 Open vSwitch 进程(ovs-vswitchd)的 CPU 利用率,以检测 ERCMC 和 ERCMD 方法对主机性能的影响.测量时间为 20 个周期,每个周期 3 s,测量结果如图 8 所示.

在图 8(a)中,对于控制器 NOX 进程,正常处理流程需要对冗余控制报文进行操作,而 ERCMC 方法需要对每个报文进行 Hash 操作,因此控制器的 CPU 均维持在 10%~35%之间,开销都较大.ERCMD 方法中控制器处理的控制报文较少,因此 NOX 进程的 CPU 利用率保持在 3%~5%之间.可



(a) NOX process

(b) Open vSwitch process

Fig. 8 Comparison of CPU utilization.

图 8 CPU 利用率比较

以看出,ERCMC 的 Hash 处理将会占用控制器的 CPU 资源,而 ERCMD 方法能够直接减小控制器负载.在图 8(b)中,ERCMC 方法虽然减少了 Open vSwitch 接收的冗余 flow-mod 报文数量,但这对 Open vSwitch 的性能影响不大,因此 CPU 利用率和 Open vSwitch 未作处理的正常处理流程接近.ERCMD 方法通过修改 Open vSwitch 消除了冗余控制报文,使得 ovs-vswitchd 进程减少了控制报文的处理发送过程,同时也减少了对冗余 flow-mod 报文的响应处理,因此 CPU 利用率从 12%左右减小到 4%左右.

可以看出,在 NOX 端实现的 ERCMC 方法在减小控制器负载时也将引进新的处理开销,这种方法需要考虑实际具体应用的部署.而 ERCMD 方法从根源上减少了冗余控制报文,在降低控制器负载的同时也进一步减小了 Open vSwitch 自身的处理开销.

5 结 论

OpenFlow 协议只将未匹配数据包发送给控制

器处理,而未根据流的定义只发送流的第 1 个未匹配数据包,这将导致非连接数据流产生冗余控制报文.在流请求报文往返时延较大的情况下,冗余控制报文将会急剧增加.本文提出了在控制器和数据平面分别消除冗余控制报文的方法 ERCMC 和 ERCMP. ERCMC 方法通过在控制器上标记每个 packet-in 报文的到达过程,减少了对冗余控制报文的处理; ERCMP 方法则直接在 Open vSwitch 的快路径上对流进行标记,并在用户空间进程和内核模块通信时进行相关处理,从根源上减少了冗余控制报文的产生.实验结果表明,在控制器减少冗余控制报文将会增加额外的处理开销,需要考虑特定的应用场景;而在 Open vSwitch 上减少冗余控制报文,由于运用的相关处理操作是 Open vSwitch 自身的库函数,因此能够在减少冗余控制报文的同时减小自身负载.

参 考 文 献

[1] Software-defined networking [EB/OL]. [2013-05-13]. https://www.opennetworking.org

[2] 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

[3] Yu Minglan, Rexford J, Freedman M, et al. Scalable flow-based networking with DIFANE [C] //Proc of the ACM SIGCOMM 2010. New York: ACM, 2010: 351-362

[4] Mogul J, Tourrilhes J, Yalagandula P, et al. DevoFlow: Scaling flow management for high-performance networks [C] //Proc of the ACM SIGCOMM 2011. New York: ACM, 2011: 254-265

[5] Tootoonchian A, Ganjali Y. HyperFlow: A distributed control plane for OpenFlow [C] //Proc of Internet Network Management Workshop/Workshop on Research on Enterprise Networking (INM/WREN). Berkeley, CA: USENIX Association, 2010: 3-3

[6] Koponen T, Casado M, Gude N, et al. Onix: A distributed control platform for large-scale production networks [C] //Proc of the 9th USENIX Conf on Operating Systems Design and Implementation (OSDI). Berkeley, CA: USENIX Association, 2010: 351-364

[7] Yeganeh S, Ganjali Y. Kandoo: A framework for efficient and scalable offloading of control applications [C] //Proc of the ACM SIGCOMM 2012 Workshop on Hot Topics in Software Defined Networking (HotSDN). New York: ACM, 2012: 19-24

[8] Zuo Qingyun, Chen Ming, Zhao Guangsong, et al. Research on OpenFlow-based SDN technologies [J]. Journal of Software, 2013, 24(5): 1078-1097 (in Chinese)
(左青云,陈鸣,赵广松,等. 基于 OpenFlow 的 SDN 技术研究[J]. 软件学报, 2013, 24(5): 1078-1097)

[9] Tootoonchian A, Gorbunov S, Ganjali Y, et al. On controller performance in software-defined networks [C] //Proc of the 2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE). Berkeley, CA: USENIX Association, 2012: 10-10

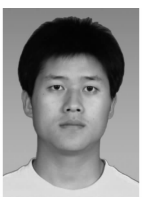
[10] Gude N, Koponen T, Pettit J, et al. Nox: Towards an operating system for networks [J]. ACM Computer Communication Review, 2008, 38(3): 105-110

[11] Pfaff B, Pettit J, Koponen T, et al. Extending networking into the virtualization layer [C] //Proc of the ACM SIGCOMM 2009 Workshop on Hot Topics in Networks (HotNets-Ⅷ). New York: ACM, 2009

[12] Sherwood R. Cbench: An OpenFlow controller benchmarker [CP/OL]. [2013-05-13]. http://www.openflow.org/wk/index.php/Oflops

[13] Tavakoli A, Casado M, Koponen T, et al. Applying NOX to the datacenter [C/OL] //Proc of the ACM SIGCOMM 2009 Workshop on Hot Topics in Networks (HotNets-Ⅷ). New York: ACM, 2009. [2014-02-16]. http://conference.sigcomm.org/hotnets/2009/papers/hotnets2009-fina2103.pdf

[14] Heller B, Sherwood R, McKeown N. The controller placement problem [C] //Proc of the ACM SIGCOMM 2012 Workshop on Hot Topics in Software Defined Networking (HotSDN). New York: ACM, 2012: 7-12



Zuo Qingyun, born in 1986. Received his BSc degree in software engineering from Tianjin University, Tianjin, China in 2008. Currently PhD candidate in PLA University of Science and Technology, Nanjing, China. His research interests include network management and measurement, software-defined networking.

Chen Ming, born in 1956. Professor and PhD supervisor of PLA University of Science and Technology, Nanjing, China. His research interests include network architecture, network performance analysis and modeling, network management and measurement, future Internet, software-defined networking.

Ding Ke, born in 1978. Received his MSc degree from PLA University of Science and Technology, Nanjing, China in 2005. He is currently both assistant professor and PhD candidate in the Laboratory of Military Network Technology at PLA University of Science and Technology. His research interests include hardware-aided computing, cloud computing, and software-defined networking.

Xing Changyou, born in 1982. Received his PhD degree from PLA University of Science and Technology, Nanjing, China in 2009. His research interests include network modeling, P2P multimedia communications, and software-defined networking.

Zhang Guomin, born in 1979. Received his PhD degree from PLA University of Science and Technology, Nanjing, China in 2009. His research interests include network management and measurement, software-defined networking.

Xu Bo, born in 1980. Received his PhD degree from PLA University of Science and Technology, Nanjing, China, in 2006. His research interests include network flow identification, network performance measurement, and software-defined networking.