PiBuffer：面向数据中心的 OpenFlow 流缓存管理模型

毛健彪 卞洪飞 韩 彪 李 韬 孙志刚

（国防科学技术大学计算机学院 长沙 410073）

摘 要 基于 OpenFlow 的 SDN(Software Defined Networking)技术在数据中心中得到广泛研究和应用，如何缓解集中的控制平面成为网络性能的瓶颈是其中的研究热点. OpenFlow 规范提出，当数据平面有缓存能力时，未命中的报文仅需发送少量摘要信息至控制器触发规则下发，从而减少控制平面与数据平面的通信负载. 然而，现有的缓存模型采用报文粒度的缓存方式，使得同一条流的多个未命中报文会被送至控制器造成额外的通信负载，而且交换机处理报文的顺序会导致流内报文乱序，从而降低通信的性能. 针对上述问题，该文提出了一种支持流内报文保序的 OpenFlow 交换机流缓存管理模型. 通过基于流粒度的未命中报文缓存方式，进一步减少控制平面与数据平面的通信开销. 通过设计流动作预处理机制，实现同一条流内报文传输保序. 该文分别基于软件交换机 OFSoftSwitch 与硬件网络实验平台 NetMagic 对该流缓存管理模型进行了原型系统验证.

关键词 数据中心网络；软件定义网络；OpenFlow 交换机；流缓存；下一代互联网

中图法分类号 TP393 **DOI 号** 10.11897/SP.J.1016.2016.01092

PiBuffer: A Model of OpenFlow Flow Buffer Management in Datacenter Network

MAO Jian-Biao BIAN Hong-Fei HAN Biao LI Tao SUN Zhi-Gang

(*College of Computer, National University of Defense Technology, Changsha* 410073)

Abstract The technology of OpenFlow-based SDN (Software Defined Networking) is widely studied and deployed in datacenter network. How to alleviate the centralized control plane becoming the bottleneck of network performance is the hotspot of research. OpenFlow specification points out that the switch can buffer the mis-matched packets and send only abstracts of them to the controller for flow rules, so as to reduce the communication load between the control plane and data plane. However, existing OpenFlow switches adopt the model of packet-granularity buffer, which cause extra communication loads by sending many packets in the same flow to the controller. In addition, the packet buffer affects the sequence of packet processing and incurs packet out-ordering, which will degrade the performance. Aiming at these problems, a novel model of flow buffer management in the OpenFlow switch is proposed. In order to further reduce the communication load, the mis-matched packets are buffered in flow-granularity. By designing the mechanism of flow action pre-processing, packets in the same flow are kept transmitting in order. We validate the correctness of the model by implementing the flow buffer on software switch OFSoftSwitch and NetMagic network experimental hardware platform.

Keywords datacenter network; software defined networking; OpenFlow switch; flow buffer; next-generation Internet

---

收稿日期：2014-12-15；在线出版日期：2015-09-15. 本课题得到国家“八六三”高技术研究发展计划项目基金(2015AA016103)和国家自然科学基金(61202483)资助.**毛健彪**，男，1988 年生，博士研究生，主要研究方向为数据中心网络、软件定义网络. E-mail：maojianbiao@nudt.edu.cn. **卞洪飞**，男，1987 年生，硕士，主要研究方向为网络交换与通信. **韩 彪**，男，1986 年生，博士，助理研究员，主要研究方向为网络路由与交换. **李 韬**，男，1983 年生，博士，助理研究员，主要研究方向为网络处理器、路由与交换. **孙志刚**，男，1973 年生，博士，研究员，主要研究领域为计算机网络体系架构、网络路由与交换.

1 引　言

作为云计算业务基础设施的重要组成部分，数据中心网络承载着云业务内部以及业务之间的通信. 随着不同业务对网络资源和灵活性的需求的不断增加，数据中心网络资源管理和配置变得越来越复杂[1]. 软件定义网络(Software Defined Networking，SDN)思想通过控制与转发分离，将网络中交换设备的控制逻辑集中到一个计算设备上，为提升网络管理配置能力带来新的思路. SDN 很好地契合了数据中心网络的集中网络管理、灵活组网、网络性能优化等方面的需求. 为此，SDN 思想在数据中心中得到广泛研究[2-4]和应用[5-6].

OpenFlow 是 SDN 体系架构中一个主流的南向接口，实现了基于流的转发. OpenFlow 交换机作为“dumb”的弱智能网络转发设备，所有流规则由远程智能的控制平面下发，这使得集中的控制平面成为网络传输性能的瓶颈. 由于受交换机 CPU 性能不足的限制，Devoflow[3]测得某商用 OpenFlow 交换机和控制器之间的流建立负载的可用带宽上限仅为 17 Mbit/s. 如何提高控制平面的可扩展性成为当前研究热点[3,7-11].

在不影响报文转发的前提下，尽量减少控制平面与数据平面的通信负载是提高控制平面可扩展性的一种解决思路. OpenFlow 规范 1.4① 指出，当 OpenFlow 交换机有缓存能力时，可支持不命中流规则的报文缓存，无需将完整报文封装到 Packet_in 消息中，仅需要封装完整报文的摘要信息即可. 这种方式减少了送至控制平面的 Packet_in 消息的报文大小，从而降低了控制平面与数据平面的通信开销.

据我们分析所知，为实现简化交换机，当前 OpenFlow 交换机的缓存管理设计[12-13]采用报文粒度的通用缓存模式，即交换机对连续的未命中报文的处理是独立的. 这种方式存在一些不足. 由于受控制器处理报文时间以及流规则传输时间和装载时间的影响，同一条流的前面多个报文均会因为未命中流规则而产生大量的突发 Packet_in 消息至控制器. 在数据中心流量快速动态变化的特性下[14]，面向连接的 TCP 流也会出现前面多个数据报文未命中流规则而被送至控制器的情况，具体分析见 2.1 节. 但对于控制器下发流规则来说，为保证一致性和减少控制器的处理负载，一条流只需获取一个报文，而无需了解该流的后续报文.

此外，交换机为了防止产生更多的未命中报文 Packet_in 消息，流规则将先被迅速装载至流表中，然后用于释放交换机中的缓存报文. 但这样会导致同一条流的后续报文先于缓存的报文发送，造成流内报文乱序. 乱序报文到达接收方后，需要重新排序后才能交给上层应用，该重组过程会给端系统带来额外开销. 而且对于 TCP 流，报文乱序会导致通信传输带宽显著下降[15].

为进一步减少控制平面与数据平面的通信开销以及解决流内报文乱序问题，本文提出了一种支持流内报文保序的 OpenFlow 交换机流缓存管理模型 Pi(Packet_in)Buffer. 该模型是对原有 OpenFlow 报文处理方式的增强和补充. PiBuffer 通过 PiBT (Packet_in Buffer Table)表，记录同一条流的不命中报文是否已向控制器发送过 Packet_in 消息，控制该流只有一个 Packet_in 消息被送往控制器，并将未命中的报文以流粒度的形式缓存. 此外，通过设计流动作预处理机制，按报文先后顺序调度已缓存报文和新命中规则报文，实现流内报文保序转发. 本文的主要贡献如下：

(1) 分析现有通用缓存模型存在的不足，指出报文粒度的缓存会产生多余的 Packet_in 请求，增加交换机与控制器的负担，同时会引起报文乱序.

(2) 提出流粒度的缓存模型 PiBuffer，以流的粒度存储未命中流的报文，减少 SDN 控制平面与数据平面的通信开销，实现流内保序转发.

(3) 分别基于软件 OpenFlow 交换机 OFSoftSwitch② 和 NetMagic 网络硬件实验平台③对 PiBuffer 进行了原型系统实现与验证，为 OpenFlow 交换机流缓存实现提供参考设计.

2 问题描述

2.1 数据中心 One-way Flow-setup 方式与流量特性

基于 OpenFlow 的数据中心[2-4]在以被动触发式建立转发路径时通常采用“One-way Flow-setup”的方式，以减少流建立的开销. 如图 1 所示，假设源主机 A 向目的主机 B 发送一条报文流，经过 n 跳 OpenFlow 交换机 $S_1, S_2, \cdots, S_n$. 当报文流首报文 P_1 到达第 1 跳 OpenFlow 交换机 S_1 时，若未命中转发

① OpenFlow switch specification, version 1.4.0 (wire protocol 0x05). http://www.opennetworking.org
② http://cpqd.github.com/ofsoftswitch13
③ http://www.netmagic.org

规则，则 S_1 产生一个 Packet_in 消息 *Pkt-in* 并通过已建立好的 OpenFlow 通道发送至集中控制器 C（涉及到消息的表示，本文用斜体字母表示特定的某个消息，用正常字母表示某一类消息）. C 处理后为该流建立从 A 到 B 的双向转发路径，即对路径上所有相关的交换机 $S_1, S_2, \cdots, S_n$ 分别下发包含流规则的 Flow_mod 消息 $Flow\text{-}mod_1$，$Flow\text{-}mod_2$，…，$Flow\text{-}mod_n$. 为防止丢包，C 还需下发一个包含 P_1 相关信息的 Packet_out 消息 *Pkt-out* 至交换机 S_1，使 P_1 转发至对应的端口. 而后，P_1 将沿着已建立好的转发路径被发送至 B.

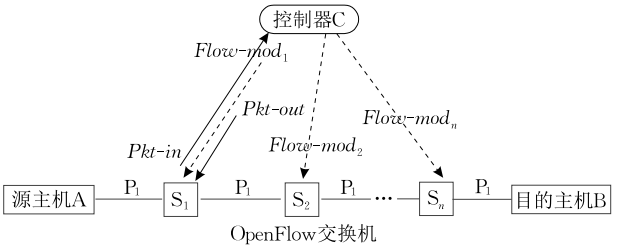


图 1 数据中心 One-way Flow-setup 模式

但是，在装载转发规则的过程中，由于交换机 $S_1, S_2, \cdots, S_n$ 与控制器的距离不同以及不同交换机当前负载不一致，可能出现规则装载慢于报文到达的情况. 例如当 P_1 到达 S_2，而 $Flow\text{-}mod_2$ 中包含的规则还未装载至 S_2. 这就会导致文献[16]中所指出的由于装载规则时延引发了控制逻辑不一致的问题.

为分析“One-way Flow-setup”需满足的条件，本文定义符号如表 1 所示. 为保证 P_1 到达转发路径上时，后续交换机 S_i（$i=2,3,\cdots,n$）均不会因未命中转发规则而产生 Packet_in 消息至控制器，则传输时间需满足如下关系：

$$t_{C\to S_1} + t_{S_1\to S_i} > t_{C\to S_i},\ i = 2,3,\cdots,n \qquad (1)$$

表 1 符号及其定义

符号	含义
n	流从源主机到目的主机经过 OpenFlow 交换机的数量
B	某条流的带宽
t_d	某条流的持续时间
$t_{A\to S_1}$	报文从源主机 A 至第 1 跳交换机 S_1 的传输时间
$t_{S_1\to C}$	未命中流表的报文从交换机 S_1 至控制器 C 的传输时间
t_C	控制器 C 处理报文的时间
$t_{C\to S_i}$	流规则从控制器下 C 装载至转发路径上第 i 个交换机 S_i 的流表中所经过的时间，$i=1,2,\cdots,n$
$t_{S_1\to S_i}$	报文从第 1 跳交换机 S_1 至路径上第 i 个交换机的传输时间，$i=2,\cdots,n$

在该情况下，一条报文流只在 OpenFlow 数据中心网络边缘（即第 1 跳交换机）未命中转发规则. 因此这种“One-way Flow-setup”方式的好处是在以被动触发式建立转发路径时，最小化一条流对控制器产生的开销. 所以目前基于 SDN 的网络操作系统（如 Floodlight①、POX② 等）均默认采用这种方式为未命中报文建立转发路径. 特别地，为了尽可能地满足式(1)的延迟条件，这些控制器在建立转发路径时均采用逆序下发规则的方式[16]. 即先从最后一跳交换机（即 S_n）开始依次下发流规则，最后才将规则装载至第 1 跳交换机（即 S_1）. 现有的研究[2-4]也假设在 OpenFlow 数据中心中满足式(1). 本文将基于式(1)分析面向数据中心的 OpenFlow 流缓存管理模型. 即未命中报文缓存只可能发生在第 1 跳 OpenFlow 交换机上.

需要指出的是，对于 TCP 流，触发流规则下发的首报文 P_1 并非一定是 TCP 建立连接的 SYN 分组或者应用产生的 Keep-alive 分组. 文献[14]指出，在数据中心网络中，一个交换机的新流到达时间间隔为 10 μs，意味着每秒钟有100 000条新流到达一个交换机. 由于瞬时存在的活跃流的数量大于交换机硬件表项数量，使得交换机中已有的 TCP 连接对应的流规则很容易在该流空闲时被替换. 例如假设 A 到 B 之间已建立转发路径，即在 $S_1, S_2, \cdots, S_n$ 均已装载流规则 $R_{A\text{-}B}$. 当数据报文 P_1 到达 S_1 时，$R_{A\text{-}B}$ 可能已被替换，需要重新触发控制器下发流规则. 因此与 UDP 流一样，已建立连接的 TCP 流的数据报文也可能未命中流规则，被送至控制器做进一步处理.

此外，在数据中心环境中，报文的环回时间 RTT（即 $2t_{A\to B}$）通常为 10 μs 到 100 μs. 而文献[17]测得 OpenFlow 流规则的装载时间约为 4 ms（即 $t_{S_1\to C} + t_C + t_{C\to S_1}$），超过了 RTT 至少两个数量级. Linux 操作系统的超时重传时间（RTO）默认为 200 ms，所以增加的报文传送延迟对报文重传影响较小.

2.2 交换机缓存问题分析与可行性论证

当交换机与控制器建立连接时，控制器通过 OFPT_FEATURES_REQUEST 消息获取交换机本地是否有缓存和缓存的大小.

若交换机无缓存，其报文处理流程如图 2(a)所示. P_1, P_2, P_3 和 P_4 是 A 发送至 B 同一条流上的报文. 交换机产生的 *Pkt-in* 消息中将携带未命中流表的完整报文至控制器 C. C 解析 P_1 后下发包含相应

① Floodlight. http://www.projectfloodlight.org/floodlight
② POX. http://www.noxrepo.org/pox/about-pox/

流规则的 *Flow-mod* 消息，消息内包含 A，B 之间的转发规则 $R_{A\text{-}B}$（主要由匹配域 $Match_{A\text{-}B}$ 和动作域 $Action_{A\text{-}B}$ 组成）. 该规则装载至流表后，流后续的报文 P_4 将命中流规则，执行相应的动作. 为了不丢包，每对应一个 *Pkt-in* 消息，C 都需将完整的报文以 *Pkt-out* 消息的形式发送至交换机，消息中携带报文的转发动作 $Action_{A\text{-}B}$. 优化后的 C 会识别出 P_2，P_3 与已收到的 P_1 属于同一条流，所以不再下发 *Flow-mod* 消息. 由于流的首报文到达交换机至流规则下发存在延迟，这个过程对 OpenFlow 通路产生的负载 $L_{无缓存}$ 大约为

$$L_{无缓存} \approx 2B(t_{S_1\to C} + t_C + t_{C\to S_1}) \tag{2}$$

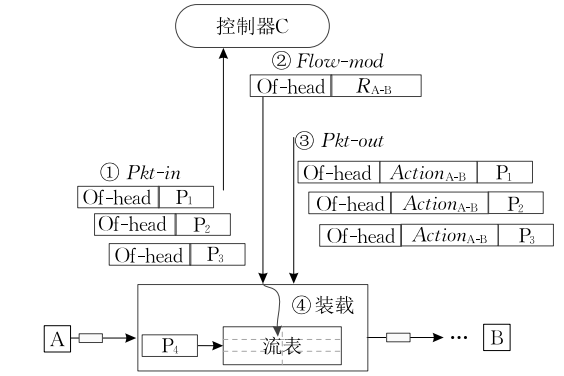
此外，对于大报文，交换机对其进行 OpenFlow 消息的封装容易超过最大传输单元（MTU）. 所以交换机还需对超过 MTU 的报文进行分片. 这也增加了交换机 CPU 的负载.

当交换机支持缓存未命中报文时，交换机在与控制器建立连接时会通告控制器交换机所能缓存报文的最大数量. 而后，控制器下发 OFPT_SET_CONFIG 消息，设置交换机产生未命中 *Pkt-in* 消息的最大字节数，默认为 128 Bytes. 经过对目前已有 OpenFlow 软件交换机的 OFSoftSwitch 源码进行分析可知，通用的报文缓存处理流程如图 2（b）所示. 未命中流表的报文 P_1，P_2 和 P_3 被临时放在缓存中，对应的缓存 id 分别为 $Buf\text{-}id_1$，$Buf\text{-}id_2$ 和 $Buf\text{-}id_3$. 交换机产生的 *Pkt-in* 消息中将携带未命中报文的摘要信息（如图 2（b）中所示的 $\{P_1\}$，默认为 128 Bytes）和缓存 id. 控制器 C 响应首个 *Pkt-in* 消息，下发一条 *Flow-mod* 消息. *Flow-mod* 消息中携带 $Buf\text{-}id_1$ 和 A，B 之间的转发规则 $R_{A\text{-}B}$. 对于该条流后续的 *Pkt-in* 消息，C 产生 *Pkt-out* 消息，其包含未命中报文的缓存 id 和转发动作 $Action_{A\text{-}B}$. 在流表中插入规则后，*Flow-mod* 消息将释放缓存 id 为 $Buf\text{-}id_1$ 所对应的报文 P_1，使该报文通过整条 OpenFlow 的报文处理流水线. 由于产生 Packet_in 消息携带的字节数与报文默认 MTU 的比值约为 1/5（300 Bytes/1500 Bytes），所以整个过程对 OpenFlow 通路产生的负载 $L_{通用缓存}$ 大约为

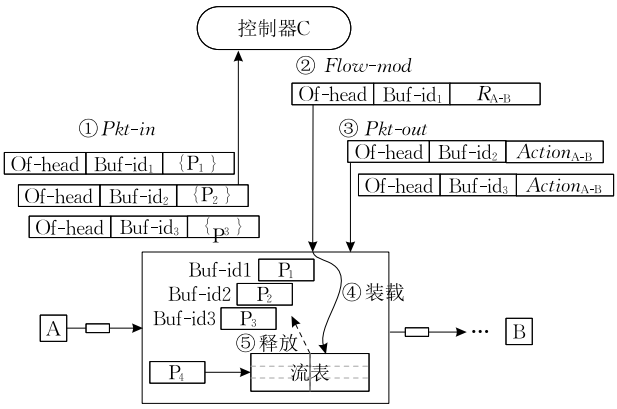
$$L_{通用缓存} \approx \frac{2B(t_{S_1\to C} + t_C + t_{C\to S_1})}{5} \tag{3}$$

因此，相比于图 2（a）中无缓存的 OpenFlow 交换机，两者的控制器-交换机通信负载比值为

$$\frac{L_{无缓存}}{L_{通用缓存}} = 5 \tag{4}$$



（a）无缓存的OpenFlow交换机



（b）通用缓存的OpenFlow交换机

图 2　无缓存与通用缓存的 OpenFlow 交换机工作模式对比

从以上分析可知，OpenFlow 交换机是否缓存未命中流表的报文决定了控制器 CPU 与交换机 CPU 的负载情况. 若无缓存，交换机处理简单，但增加了通信负载与控制器的处理开销；若有缓存，使交换机具备了一些智能，分担了报文缓存的工作，减少了控制平面与数据平面通信的开销，但也增加了缓存设计与管理的难度. 目前商用硬件 OpenFlow 交换机[①]为降低成本、简化报文处理，通常采用无缓存的设计. 尽管在硬件交换机中增加大量缓存会提升成本和设计的复杂性，但 Lu 等人[18]提出采用 CPU＋DRAM 的方式增加交换机的缓存，提升数据中心网络数据平面的功能. 此外，目前数据中心中 SDN 的实现通常基于服务器内部的软件虚拟交换机（例如 Open vSwitch[13]），缓存对于软件交换机来说实现问题不大. 因此，总的说来，在交换机中为未命中流表的报文增加缓存是可行的.

2.3　现有缓存模型存在的不足

如图 2（b）所示，同一条流的报文以 P_1-P_2-P_3-P_4

① Centec V330 Specification. http://www.centecnetworks.com

的顺序到达交换机.假设 P_1,P_2 和 P_3 未命中流表触发控制器下发规则,且在 P_4 到达时,该流的规则已装载至流表中,所以 P_4 查表命中流规则.交换机内一种可能的处理顺序如图 3 所示.OpenFlow 规范中指出,为尽量减少同一条流产生 Packet_in 报文的数量,Flow_mod 消息先装载流规则,然后释放对应 Buffer_id 的报文使其通过 OpenFlow 报文处理流水线.P_4 查表命中流规则转发这一事件先于缓存的 P_1 被释放发生.而后,*Pkt-out*2,*Pkt-out*3 消息分别释放报文 P_2 和 P_3,则该流经过 OpenFlow 交换机后可能的一种报文输出序列为 P_4-P_1-P_2-P_3.文献[15]分析了 TCP 端系统为重组乱序报文所需要的缓存容量以及由此产生的报文延时.分析结果表明.报文重排序要求很大的缓存容量,对端系统提出较高要求;其次,重排序引起的报文延时显著,而且报文延时与 TCP 连接吞吐率和分组大小紧密相关,在较高的连接吞吐率和小尺度分组条件下,重排序操作将对上层网络应用性能产生严重的影响.尽管乱序报文对 UDP 流的影响不如对 TCP 流的影响大,但是保序后的报文减少了应用层重组报文的开销,因而提高应用通信的性能.

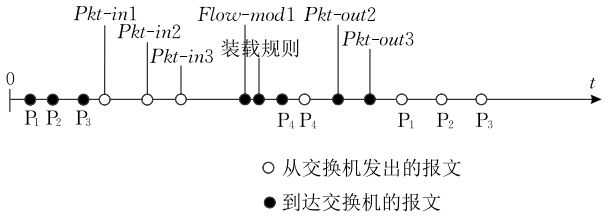


图 3 OpenFlow 交换机内报文的处理先后顺序

OpenFlow 交换机的缓存可有效减少交换机与控制器之间的信息交互负载.但我们发现,若对交换机的缓存实施有效的管理,则可进一步减少交换机与控制器之间的通信开销.例如对于同一条流的未命中报文只需要产生一个 Packet_in 消息至控制器.而且,原有的缓存模型易导致上文中分析出现的报文乱序问题,降低应用通信的性能.为此,本文针对 OpenFlow 交换机流缓存的管理展开研究,以降低 OpenFlow 数据和控制平面的通信开销,实现报文流内保序转发.

3 PiBuffer 流缓存管理模型

由于现有的 OpenFlow 交换机缓存管理存在不足,本文基于原有的 OpenFlow 规范提出了面向数据中心的流缓存管理模型 PiBuffer.该模型对未命中流表的数据报文,按照流粒度进行缓存与管理,使得每条无转发规则的流只产生一个携带该流标识信息的 Packet_in 消息,将其发送至控制器,以进一步减少交换机与控制器之间的信息交互,提高控制平面的可扩展能力.同时该模型通过流动作预处理机制,解决现有缓存方式下流表下发导致的流内报文乱序问题.

基于第 2 节不等式(1)的假设,本文仅考虑在第 1 跳交换机未命中报文的缓存情况.如图 4 所示,在 PiBuffer 模型中,OpenFlow 流水线的基本过程保持不变,即按照 OpenFlow 规范实现报文头的解析、查找流表、根据查表结果执行相应动作.引入 PiBuffer 主要对未命中流表报文的处理过程和流规则的装载过程进行了修改.PiBuffer 重新定义了 3 类消息(Packet_in 消息、携带 Buffer_id 的 Flow_mod 和 Packet_out 消息)的处理过程,但不改变其他 OpenFlow 消息的处理过程.

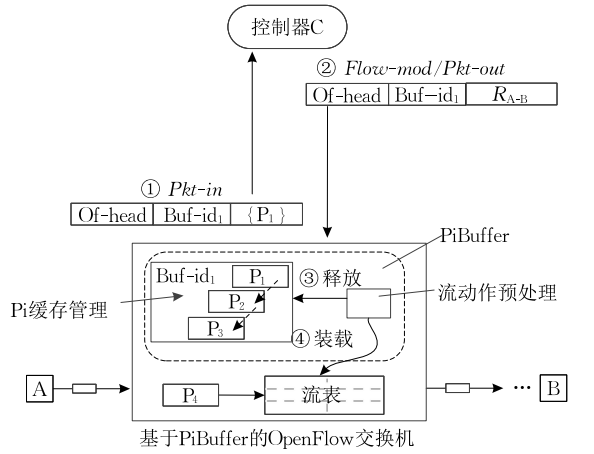


图 4 基于 PiBuffer 的交换机-控制器交互

PiBuffer 流缓存管理模型主要包含两个功能部件:流动作预处理部件和 Pi 缓存管理部件.流动作预处理部件主要功能是识别携带 Buffer_id 的 Flow_mod/Packet_out 消息,先释放缓存中与该 Buffer_id 相关的报文,然后按照 Flow_mod/Packet_out 消息中的流动作进行转发.OpenFlow 规范中,流规则包含了流匹配域和流动作,Packet_out 消息中携带流动作,Flow_mod 消息中携带流规则.所以若为 Flow_mod 消息,流动作预处理部件在等待缓存的流报文全都释放完后,才将该消息内携带的规则装载至流表中,从而实现流内的报文保序.Pi 缓存管理部件主要功能是实现未命中报文的缓存,流缓存区的管理,以及产生流粒度的 Packet_in 报文,减少交换机发往控制器的 Packet_in 请求数目.

当报文未命中流表时，OpenFlow 流水线将未命中的报文输出至 Pi 缓存管理部件. Pi 缓存管理部件根据一定的缓存策略决定是否缓存该报文. 根据文献[14]的测量结果可知，在数据中心网络中，报文大小表现为双峰分布，主要集中在 200 Bytes 左右和 1400 Bytes 左右. 200 Bytes 大小的报文通常为应用的 keep-alive 报文和 TCP Ack 报文. 1400 Bytes 大小的报文为有效数据报文. 这些特性将会影响未命中报文的缓存策略. 若不缓存，则将完整报文封装为 OpenFlow 的 Packet_in 消息，发送至控制器做进一步处理. 若缓存，报文以流为粒度组织存放. 若是流的首报文，则需提取该报文的前 128 Bytes，封装成 Packet_in 消息，发送至控制器. 若报文为流的后续报文，则不产生 Packet_in 消息. 所以，在 PiBuffer 模型下，一条未命中的报文流对 OpenFlow 通路产生的负载为一个 *Pkt-in* 消息和一个 *Flow-mod* 消息(或者 *Pkt-out* 消息). 因此 PiBuffer 对 OpenFlow 通路产生的负载 L_{PiBuffer} 大约为

$$L_{\text{PiBuffer}} \approx 300\,\text{Bytes} \tag{4}$$

所以，无缓存、通用缓存、基于 PiBuffer 的缓存，这 3 种模式在一条带宽为 B 的流不命中规则的情况下，占用控制器-交换机 OpenFlow 通路负载比值为

$$L_{\text{无缓存}} : L_{\text{通用缓存}} : L_{\text{PiBuffer}} = 2BT : \frac{2}{5}BT : 300,$$

$$T = (t_{S_1\to C} + t_C + t_{C\to S_1}) \tag{5}$$

式中单位为字节.

PiBuffer 通过流动作预处理，实现了先释放未命中报文的缓存后装载规则的方式，达到了报文保序的目的. 但这也对交换机的处理能力提出要求. 假设交换机释放报文缓存的速率为 R，释放所需的时间为 t_s，其余标记如表 1 所示. 当流持续时间 $t_d > t_{S_1\to C} + t_C + t_{C\to S_1}$，即规则装载后，报文流仍在传输，则交换机处理缓存的能力需满足

$$R \times t_s \geq B \times (t_{S_1\to C} + t_C + t_{C\to S_1}),$$

即

$$R \geq \frac{B \times (t_{S_1\to C} + t_C + t_{C\to S_1})}{t_s} \tag{6}$$

若交换机缓存处理能力不满足该条件，会产生 PiBuffer 缓冲区溢出的情况.

4　PiBuffer 方案设计

PiBuffer 流缓存管理模型系统方案设计结构如图 5 所示. 该设计基于原有的 OpenFlow 处理流水线，重定义了对未命中报文的缓存操作过程，以及对包含 Buffer_id 的 OpenFlow 消息预处理过程. 4.2 节和 4.3 节将对修改后的这两个过程进行详细描述.

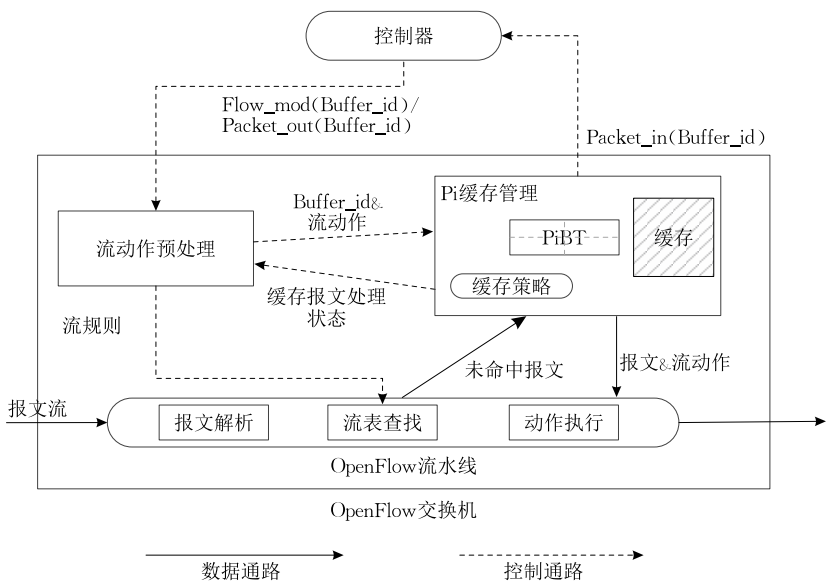


图 5　PiBuffer 流缓存管理模型

Pi 缓存管理部件主要包括缓存创建/更新模块、缓存释放模块、PiBT(Packet_in Buffer Table)和缓存. PiBT 是其中的核心，用于记录未命中报文缓存的相关信息. 创建/更新模块负责将未命中流表的报文根据缓存策略决定是否缓存，产生相应的 Packet_in 消息. 缓存释放模块将流动作预处理部件产生的 Buffer_id 和流动作等信息作为输入，根据 PiBT 表中的标识释放报文流的缓存信息和位置，取

出缓存区报文,与流动作一起输出至 OpenFlow 流水线中的动作执行模块.释放完该流对应的所有缓存后,缓存释放模块向流动作预处理部件发送一个缓存报文处理完成的状态消息.

流动作预处理部件包括动作解析模块和规则装载模块.其中动作解析模块负责解析控制器下发的包含 Buffer_id 的 Flow_mod/Packet_out 消息,并将解析得到的流动作和 Buffer_id 发送至 Pi 缓存管理部件.在接收到缓存报文处理完成的状态消息后,规则装载模块才将流表规则写入至 OpenFlow 流水线的流表中.

4.1 PiBT 组织结构

PiBT 负责记录报文缓存相关的信息,其表项内容包括匹配字段、缓冲区首地址、当前缓冲区地址、缓存报文计数和超时时间字段,格式如图 6 所示.其中,匹配字段为未命中报文流的流标识,用于识别不同流的未命中报文以及区分同一条流的首报文和后续报文.缓冲区首地址为一条未命中流的第 1 个报文的缓存单元地址,也是该流在缓存区的首地址.当前缓冲区地址表示一条未命中流的最近一次缓存的报文的缓存区地址.缓存报文计数表示当前流中缓存的未命中报文个数.流超时字段用于表项的生存时间.若长时间未收到控制器对未命中报文的响应消息,即超时时间超过一定的阈值,就删除该表项并释放对应的缓存空间.

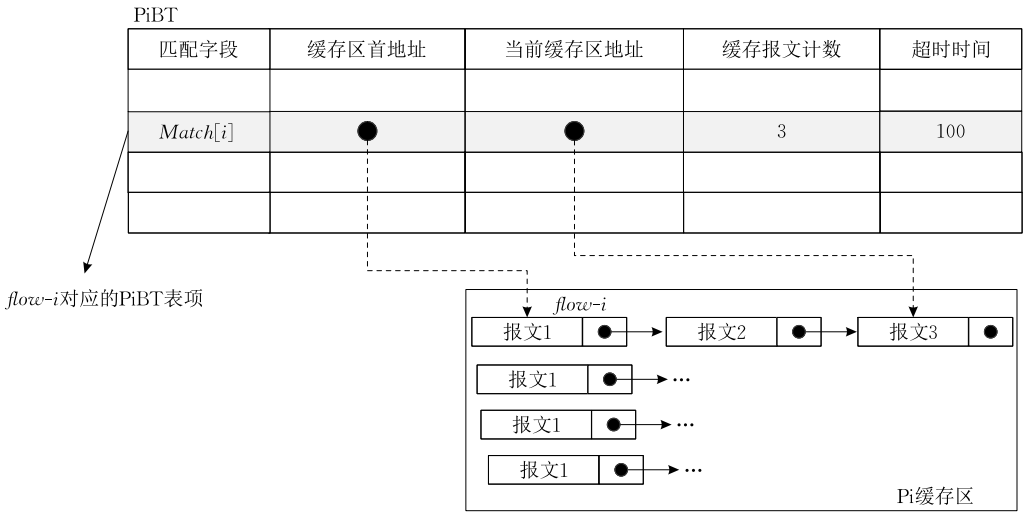


图 6 Pi 缓存表的组织结构

Pi 缓存区以单向链表的结构组织缓存的报文.当缓存一个新报文时,将该新报文缓存单元的地址添加到同一条流缓存区内上一个报文缓存单元的下一个报文指针,同时更新当前缓存区地址,指向该新报文的地址.若为软件实现该缓存管理模型,则无需关心每个报文的大小.而基于硬件实现时,为适应不同大小的报文,每个报文缓存区可按照 MTU 的大小分配.由于之前提到,数据中心中的报文大小分布具有双峰特性.若按照报文大小指定缓存策略,即小报文不缓存(小于 300 Bytes),仅缓存大报文(大于 1400 Bytes),那么硬件实现中按照 MTU 分配每个报文的存储空间是合理的.

PiBT 的表项数量为交换机可同时支持未命中流缓存的数量.数量越多,表明该交换机越能适应高速动态变化的数据中心流量.该数量的大小受到交换机存储资源的限制.

4.2 未命中报文缓存过程

Pi 缓存管理部件的工作流程可分为未命中报文的缓存及释放两个流程.报文缓存流程如下:

(1) 当缓存创建模块接收到 OpenFlow 流线部件中的流表查找模块产生的未命中流表项的报文时,缓存创建模块首先根据未命中报文的头字段查询 PiBT 表.

(2) 当报文命中 PiBT 表时,说明该未命中报文为一条未命中后续报文,缓存创建模块将该报文的缓存区地址链接到上一个报文缓存区的指针,并将报文缓存信息更新到命中的 PiBT 表中.

(3) 当报文未命中 PiBT 表时,说明该未命中报文为一条未命中流的首报文,缓存创建模块将为该报文创建一个 PiBT 表项,并将创建的 PiBT 表项的地址输出到 OpenFlow 流水线中的转发模块,并由其将该 PiBT 地址作为 Buffer_id 封装到 Packet-in 消息到控制器.该流程的形式化描述如过程 1.

过程 1.　未命中报文缓存处理过程.

输入：未命中的报文 P

输出：Packet_in 消息

```
IF(!Policy(P))
    Buffer_id(P)=NONE;
    Packet_In=Generate_PI(P,Buffer_id(P));
ELSE IF (!Storage(P))
    Buffer_id(P)=NONE;
    Packet_In=Generate_PI(P,Buffer_id(P));
ELSE IF(Lookup(PiBT,P))
    Buffer_address=Find(PiBT);
    Update(PiBT,P);
ELSE
    New(PiBT,P);
    {P}=abstract(P);
    Packet_In=Generate_PI({P},Buffer_id(P));
ENDIF
```

4.3　流动作预处理过程

流动作预处理工作流程如下：

(1) 当控制器下发包含 Buffer_id 的 Flow_mod/Packet_out 消息时，流动作预处理部件中的动作解析模块捕获该消息，提取其中的流表规则和 Buffer_id，并输出给 Pi 缓存管理部件.

(2) Pi 缓存管理部件根据 Buffer_id 查找 PiBT 表，将表中对应的 Pi 缓存中的报文依次释放至 OpenFlow 流水线中，并执行相应的报文动作.

(3) 当缓存释放模块释放完该条流的所有缓存报文后，若为 Flow_mod 消息，规则装载模块才将流表规则插入到 OpenFlow 流水线部件中的流表查找模块中的流表中，并更新相应的计数器. 该流程的形式化描述如过程 2.

过程 2.　流动作预处理过程.

输入：包含 Buffer_id 的 Flow_mod/Packet_out 消息

输出：流动作和缓存报文

```
IF (Lookup(PiBT,Buffer_id)) THEN
    While (Retrieve(PiBuffer,Buffer_id)) DO
        Release(Pkt);
        Pipeline(Pkt,Actions)
    END WHILE
ENDIF
IF(Flow_mod) THEN
    Rules=Exact(Flow_mod);
    Insert(Flow_table,Rules);
ENDIF
```

5　实验结果与分析

在基于OpenFlow的数据中心应用场景中，OpenFlow 交换机既有物理硬件交换机的实现形态，也有软件交换机的实现形态. 为此，本文分别从软件和硬件两个实现角度对该缓存管理模型进行验证.

5.1　基于 OFSoftSwitch 软件交换机的软件实现

OFSoftSwitch① 软件交换机是由斯坦福大学、爱立信研究中心和 CPqD 公司共同维护的开源项目，支持 OpenFlow v1.3. 由于该软件交换机在用户态实现，因此相对 Open vSwitch 较易于修改，方便对新功能进行验证. 本文基于第 4 节提出的 PiBuffer 设计方案，在 OFSoftSwitch 软件交换机上实现了所提出的流缓存管理模型，替换原有的通用缓存方式.

实验所用的机器是一台 CPU 主频为 3.2 GHz 的 Intel Core i5、内存为 8GB 的台式机. 在该机器上构建如图 7 所示拓扑. 控制器采用模块化的 Floodlight，其上运行三层转发的应用. 在 VMware 虚拟机软件中运行 3 台虚拟机，分别命名为 VM1、VM2、VM3. 其中 VM1 内运行 OFSoftSwitch 软件交换机，通过 OpenFlow 通道与主机内的 Floodlight 控制器连接. VM2 和 VM3 分别作为源主机和目的主机，并处于不同的子网，只有通过控制器配置流规则，两者才可进行通信. 由于本文假设满足不等式(1)，仅需考虑第 1 跳交换机的缓存情况即可，所以该实验场景简化是合理的.

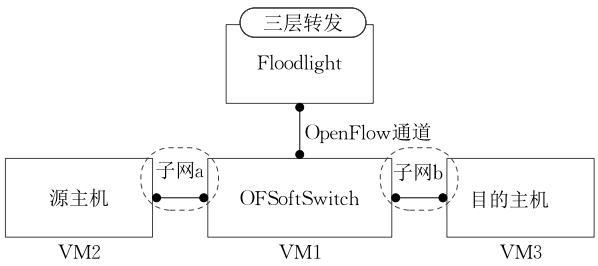


图 7　基于 OFSoftSwitch 的 PiBuffer 原型验证环境

首先本文对 PiBuffer 减少了交换机与控制器之间的通信开销进行验证. 使用 Iperf 工具进行测试，将 VM2 作为源主机，VM3 作为目的主机，测量在不同缓存模式下(无缓存、通用缓存、基于 PiBuffer 的缓存)Floodlight 控制器与 OFSoftSwitch 软件的负载情况. Iperf 的流量大小为 5 Mbps，持续时间为 60 s，测试结果如图 8 所示.

由于实验中 Floodlight 控制器并未及时发出 Flow_mod 消息下载规则，因此在 60 s 内，所有发送至软件交换机的报文都未命中规则. 接收带宽与发送带宽均是从交换机的角度而言，分别对应 Packet_out 消息和 Packet_in 消息. 从结果中可明显看到，基

① http://cpqd.github.com/ofsoftswitch13

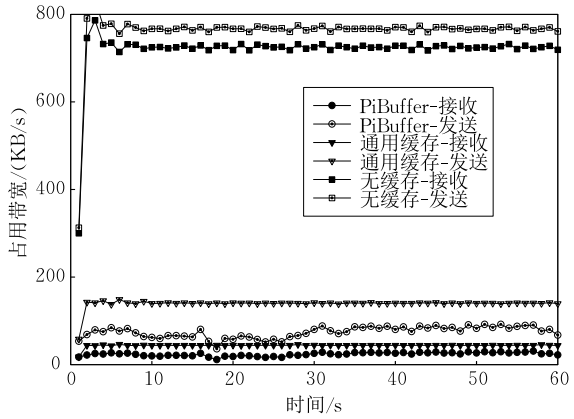


图 8 3 种缓存模式占用 OpenFlow 通道带宽的对比

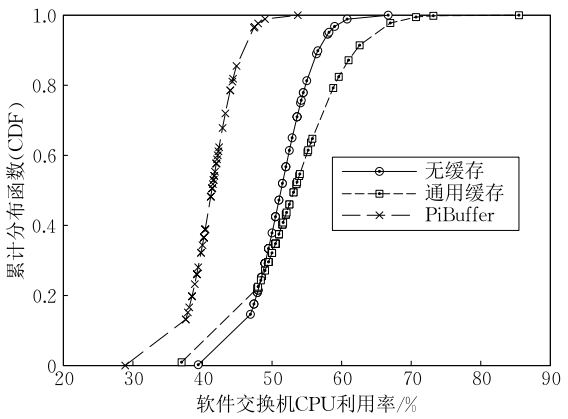


图 9 3 种缓存模式占用软件交换机 CPU 资源的对比

于 PiBuffer 缓存模式对 OpenFlow 通道带宽占用的最少,平均值分别约为 23 KB/s(接收)和 74 KB/s(发送). 基于通用缓存的次之,平均值分别约为 43 KB/s(接收)和 139 KB/s(发送). 无缓存的情况下,所有的报文都被封装成 Packet_in 消息送至控制器,并且也会被 Packet_out 至交换机,平均值分别约为 719 KB/s(接收)和 761 KB/s(发送). 其接收与发送的占用带宽基本与 Iperf 所发出的流量的带宽(5 Mbps)处于一个量级.

无缓存情况下总共占用带宽与通用缓存情况下总共占用带宽比值约为 1∶5,与第 3 节中理论分析一致. 在有缓存的情况下,Packet_out 消息只需携带 32 位的 Buffer_id,不需要携带报文相关信息,经过计算可知,仅为 Packet_in 消息的 1/3,这也符合实验中所观测到接受速率与发送速率的比值.

大量的 OpenFlow 控制消息还可造成控制器与交换机的过载. 对控制通路的带宽占用减少可降低交换机与控制器的 CPU 利用率. 图 9 反应了引入 PiBuffer 后对软件交换机占用 CPU 利用率的影响. 通过数据可知,采用 PiBuffer 模式的软件交换机要比原先更少地占用 CPU 资源. 这是由于 PiBuffer 缓存管理模型使得更少的报文被发送至控制器,同一条流的非首报文会被缓存在交换机中. 而封装产生 Packet_in 消息相比于缓存报文更消耗 CPU 资源. PiBuffer 更多地将未命中的报文缓存在交换机内部,减少了发往控制器的 Packet_in 消息,因此相比于通用缓存模式,PiBuffer 模式下交换机的 CPU 资源使用率减少约 20%. 可以预见,交换机引入 PiBuffer 会使控制器 CPU 资源占用较少.

所以,从以上实验结果可知,在交换机缓存资源足够的前提下,缓存报文不仅可以减少交换机与控制器之间的通信负载,同时可减少交换机与控制器本身的 CPU 处理负载.

其次,本文对 PiBuffer 解决通用缓存导致的流内乱序问题进行说明和验证. 源主机向目的主机连续发送携带顺序序号(1～100)的报文. 分析目的主机端接收到的报文序号,结果如图 10 所示.

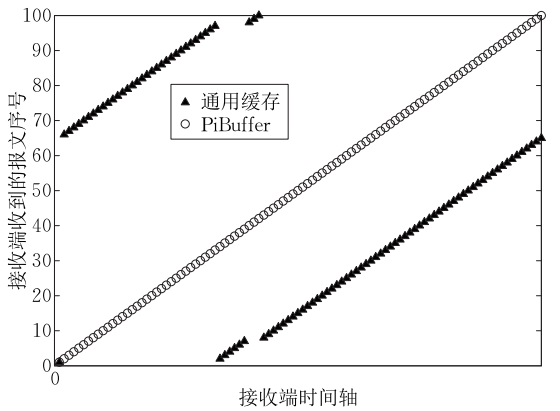


图 10 目的主机收到报文序号随时间的关系

从 2.1 节中论述可知,面向连接的 TCP 报文流在数据中心网络流特性的条件下,在到达 OpenFlow 交换机时,也可能存在无法命中流规则的情况. 从上图实验结果可知,通用缓存会带来较大的流内报文乱序问题. 接收到的首报文序号为 1,第 2 个报文为 64,依次至 97,而后又收到序号为 2 的报文,这会严重降低 TCP 流的传输带宽. 而对于应用了 PiBuffer 的交换机,则报文流不会产生报文乱序的情况.

5.2 基于 NetMagic 平台的硬件实现

本文基于 NetMagic 可编程硬件网络实验平台实现该缓存管理模型. NetMagic 平台的 FPGA 硬件逻辑可分为报文接收、发送、复制等基本通用逻辑和自定义开发逻辑,其中基本通用处理逻辑为自定义逻辑提供清晰的易用接口[19]. 缓存模型的逻辑实现如图 11 所示.

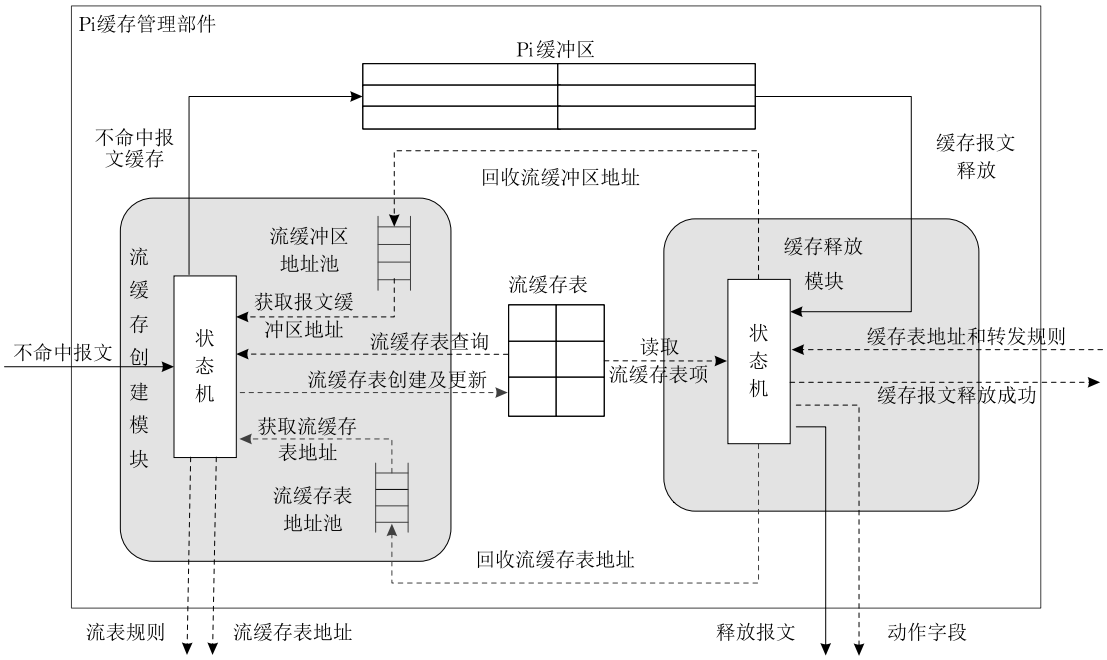


图 11　基于 NetMagic 的 PiBuffer 硬件实现逻辑

本实验目的是为了测量不同规则装载延迟下(即上文 $t_{S_1\to C}+t_C+t_{C\to S_1}$)Pi 缓存区的大小，即未命中报文的缓存使用情况. 搭建了如图 12 所示的 1 个原型系统. 图中 3 台 PC，1 台为控制器，其余两台为报文的发送方 A 和接收方 B；1 台基于 NetMagic 平台实现 PiBuffer 的 OpenFlow 交换机. 为简化实现但不影响实验结果，该交换机的查表过程为简化后的 OpenFlow1.0 标准.

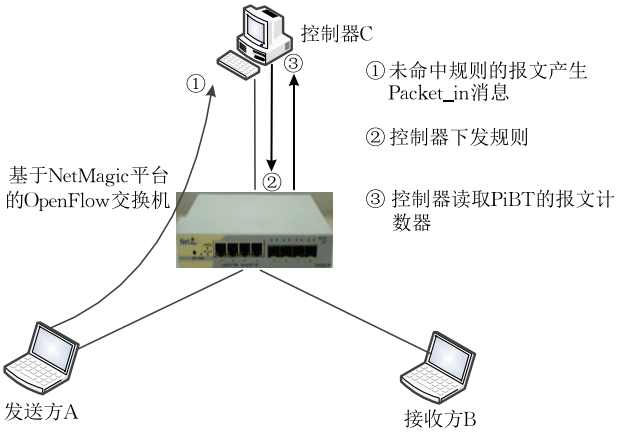


图 12　缓存开销验证环境

表 2 给出了实现该缓存模型所需的 FPGA 资源使用情况. 从资源的占用比例可知，实现流缓存管理功能所需要的 FPGA 资源并不多，硬件实现可行. 在缓存报文时按照 2 KB 大小为每个报文分配缓存空间.

表 2　流缓存管理功能 FPGA 资源占用情况

资源名称	FPGA 总资源	所有模块占用资源数量/%	缓存相关逻辑占用资源数量/%
查找表 LUT	36100	13528(37.5%)	627(1.7%)
寄存器	36100	15620(43.3%)	552(1.5%)
存储器位	2939904	1728640(58.8%)	74496(2.5%)

实验过程如下：首先由发送端 A 主机使用 VLC 软件流化一个视频流向接收端 B 主机发送，当视频流的首报文流经过 NetMagic 时，NetMagic 将为该流创建一个流缓存表项，并将该流的缓存信息封装成 Packet-in 发往控制器. 然后，当 Packet-in 到达控制器，通过设定控制器下发流表项的延时，来模拟远端控制器响应延时. 延时以 50 ms 为单位，设定 9 个延时间隔，最大延时设定为 500 ms. 在这个延时期间内，NetMagic 将缓存报文信息更新到流缓存表中. 另外，由于不同大小的报文的缓冲区大小相同，因此，测试会关心控制器延时期间的缓存报文个数，而缓存报文个数记录在该流的缓存表项中. 最后，当控制器下发的 *Flow-mod* 到达 NetMagic 时，由 NetMagic 读取在流缓存表项中记录的该流的流缓存报文个数字段，并将其封装成特殊报文发往主机 B，这样缓存报文的个数就可以通过主机 B 捕获的特殊报文获得.

图 13 测试结果显示，缓存报文大小随控制器延时的增加成线性增长，其中测试曲线的抖动是由于发送端软件流化视频流产生的速率抖动造成的.

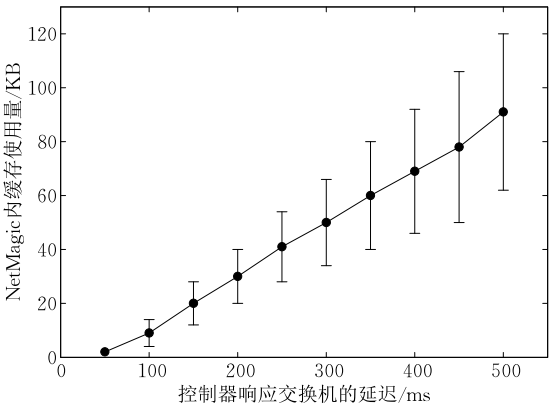


图 13 不同控制器延时下缓存使用情况

6 相关研究

避免集中的控制平面成为网络性能的瓶颈是SDN在数据中心中落地部署所要解决的重要问题之一.目前研究人员主要从两个方面解决该问题:一类是提升控制平面的处理能力;另一类是增加数据平面的功能,尽可能减少控制平面与数据平面的非重要信息的交互.

在提升控制平面的处理能力方面,SDN控制器的研发人员运用多线程并行技术、共享队列、批处理等提升单点控制器的报文处理能力[20].但由于单点控制器处理性能有上限,为此研究人员提出利用分布式技术提高控制平面处理能力. HyperFlow[7]通过部署多台分布式控制器来分摊与数据平面的交互开销,各控制器间通过消息订阅/发布方式获取全网的统一视图,从而实现逻辑一致性和集中控制. Onix[8]面向商业化应用,在更高层次上提出了控制平面扩展的分层架构,全局网络状态一致性通过每个控制器上的网络信息库来维护. ONOS[9]是近年来发展较为成熟的开源分布式SDN控制平台.以上研究均可与本文工作互补,提高控制平面的可扩展能力.

在OpenFlow数据平面功能增强方面,与本文工作最接近的是Kotani等人[10]提出的Packet_in消息过滤机制.同PiBuffer类似,该机制在发送Packet_in消息之前会记录报文头域的值.但不同的是,过滤机制记录的值用于过滤后续相同值的报文,降低高速报文对控制平面的影响.而PiBuffer并不丢弃这些报文,而是进行缓存,减少丢包带来的影响.此外,DIFANE[11]提出区分权威OpenFlow交换机和普通OpenFlow交换机,将OpenFlow控制平面工作部分卸载到权威OpenFlow交换机上,各普通OpenFlow交换机根据控制平面下发的分区规则与对应的权威OpenFlow交换机通信,再由权威OpenFlow交换机根据控制平面下发的权威规则与控制平面交互,从而减少控制平面与数据平面间的交互开销.另外,分区规则与权威规则只有在网络拓扑结构发生变化时,才被动地更新,可进一步减轻控制平面负载. DevoFlow[3]则在数据平面采用规则复制和局部操作的方式来减小与控制平面的交互,并使用触发、报告、采样等手段进一步减小数据平面的统计信息到控制平面的开销,从而提高SDN控制平面的可扩展性.

综上分析,增强控制平面的处理能力方案的优点是通用性强,对SDN网络体系结构影响较小,缺点是SDN网络内部仍充斥大量控制器与交换机间的消息,难以降低OpenFlow消息对网络资源占用的开销.而扩展数据平面功能的方案,则可以从根源上减少控制器和SDN网络的流量负载.本文所研究的OpenFlow交换机流缓存管理技术,属于数据平面增强的方案,实现了OpenFlow规范中指定的优化方法,可以在提高控制平面可扩展性的同时,实现与现有SDN网络兼容互通.

7 结论与下一步工作

为减少基于OpenFlow的数据中心中控制平面与数据平面的交互,提高控制平面的可扩展性,本文利用交换机中的存储资源,提出了一种流粒度的未命中报文缓存管理模型PiBuffer.原有的缓存模式缺乏对未命中报文的有效组织和管理,并没有充分利用缓存对减少流建立开销带来的好处.该模型通过PiBT表记录未命中报文的标识信息,区分同一条流的首报文和后续报文,使得同一条未命中报文流只产生一个Packet-in消息发送到控制器.而且使得流表下发时,先发送缓存的报文,从而实现流内报文保序.实验结果表明,PiBuffer与现有的缓存方式相比,降低了对OpenFlow通道的占用带宽和交换机的CPU开销,实现报文按序转发.

实验中发现PiBuffer占用较多的交换机缓存资源且影响了转发效率,下一步工作将对报文缓存大小进行评估并对缓存时的转发性能进行优化.

致 谢 网络与信息安全研究所662教研室的徐东来老师、熊兆中同学在代码调试过程中提供了指导和帮助,在此表示感谢!

参 考 文 献

[1] Deng Gang, Gong Zheng-Hu, Wang Hong, et al. Analysis and survey of resource management on modern data center networks. Journal on Communications, 2014, 35(2): 166-181(in Chinese)
(邓罡, 龚正虎, 王宏等. 现代数据中心网络资源管理技术分析与综述. 通信学报, 2014, 35(2): 166-181)

[2] AlFares M, Radhakrishnan S, Raghavan B, et al. Hedera: Dynamic flow scheduling for data center networks//Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI). San Jose, USA, 2010: 19-19

[3] Curtis A R, Mogul J C, Tourrilhes J, et al. Devoflow: Scaling flow management for high-performance networks//Proceedings of the SIGCOMM. Toronto, Canada, 2011: 254-265

[4] Stephens B, Cox A, Felter W, et al. PAST: Scalable ethernet for data centers//Proceedings of the ACM International Conference on Emerging Networking EXperiment and Technologies (CoNEXT). Nice, France, 2012: 49-60

[5] Koponen T, Amidon K, Balland P, et al. Network virtualization in multi-tenant datacenters//Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI). Seattle, USA, 2014: 203-214

[6] Banikazemi M, Olshefski D, Shaikh A, et al. Meridian: An SDN platform for cloud network services. IEEE Communications Magazine, 2013, 51(2): 120-127

[7] Tootoonchian A, Ganjali Y. HyperFlow: A distributed control plane for OpenFlow//Proceedings of the Internet Network Management Workshop on Research on Enterprise Networking (INM/WREN). San Jose, USA, 2010: 3-3

[8] 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 (OSDI). Vancouver, Canada, 2010: 1-12

[9] Berde P, G Matteo, H Jonathan, et al. ONOS: Towards an open, distributed SDN OS//Proceedings of the SIGCOMM Workshop on Hot Topics in SDN. Chicago, USA, 2014: 1-6

[10] Kotani D, Okabe Y. A packet-in message filtering mechanism for protection of control plane in OpenFlow networks//Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS). Los Angeles, USA, 2014: 29-40

[11] Yu M, Rexford J, Freedman M J, et al. Scalable flow-based networking with DIFANE//Proceedings of the SIGCOMM. New Delhi, India, 2010: 351-362

[12] Phemius K, Bouet M. OpenFlow: Why latency does matter//Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM). Ghent, Belgian, 2013: 680-683

[13] Pfaff B, Pettit J, Koponen T, et al. The design and implementation of open vSwitch//Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI). Oakland, USA, 2015

[14] Benson T, Akella A, Maltz D A. Network traffic characteristics of data centers in the wild//Proceedings of the ACM/USENIX Internet Measurement Conference(IMC). Melbourne, Australia, 2010: 267-280

[15] Lv Gao-Feng, Hu Xiao-Feng. TCP friendly design of router. Journal of National University of Defense Technology, 2005, 27(5): 25-30(in Chinese)
(吕高锋, 胡晓峰. TCP 友好的路由器设计. 国防科技大学学报, 2005, 27(5): 25-30)

[16] Canini M, Venzano D, Peresini P, et al. A NICE way to test OpenFlow applications//Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI). Berkeley, USA, 2012: 10-10

[17] Agarwal K, Dixon C, Rozner E, Carter J. Shadow MACs: Scalable label-switching for commodity ethernet//Proceedings of the SIGCOMM Workshop on Hot Topics in SDN. Chicago, USA, 2014: 157-162

[18] Lu Guo-Han, Guo Chuan-Xiong, Li Yu-Long, et al. ServerSwitch: A programmable and high performance platform for data center networks//Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI). Boston, USA, 2011: 2-2

[19] Li Tao, Sun Zhi-Gang, Chen Yi-Jiao, et al. A novel packet processing model for next-generation internet experimental platform——EasySwitch. Chinese Journal of Computers, 2011, 34(11): 2187-2196(in Chinese)
(李韬, 孙志刚, 陈一骄等. 面向下一代互联网实验平台的新型报文处理模型——EasySwitch. 计算机学报, 2011, 34(11): 2187-2196)

[20] Erickson D. The beacon OpenFlow controller//Proceedings of the SIGCOMM Workshop on Hot Topics in SDN. Hong Kong, China, 2013: 13-18

MAO Jian-Biao, born in 1988, Ph. D. candidate. His research interests include datacenter network, software defined networking.

BIAN Hong-Fei, born in 1987, M. S. His research interests include network switching and communication.

HAN Biao, born in 1986, Ph. D., assistant professor. His research interests include network routing and switching.

LI Tao, born in 1983, Ph. D., assistant professor. His research interests include network processor, routing and switching.

SUN Zhi-Gang, born in 1973, Ph. D., professor. His research interests include network architecture, routing and switching.

Background

Software Defined Networking (SDN) has been widely studied and deployed in Datacenter Network (DCN). Based on separating the control plane and data plane and centralized controlling, SDN tries to solve current networking problems in the DCN and provides a new way for the network innovation. SDN can simplify network and traffic management because it enables flow-level control over OpenFlow switches and provides global visibility of the flows in the DCN. However, such fine-grained flow control comes with excessive costs: it involves the controller too often both on flow setups and gathering statistics. Thus, the centralized controller is the bottleneck regarding to the network performance. Many researchers have sought to improve the scalability of the control plane. The solutions are mainly divided into two kinds. One is to increase packet processing capability of the control plane with the technique of distributed systems, multiple-threads parallel and batch processing. The other is to enhance the functionality of the data plane to reduce the exchanging of unimportant messages between the data plane and the control plane.

This work is supported by the National Natural Science Foundation of China under Grant No. 61202483 and the National High Technology Research and Development Program (863 Program) of China under Grant No. 2015AA016103. This project aims to make advances to the design of architecture of SDN and build the complete SDN network environment with independent intellectual property rights. Our work is a part of the design of high-performance data plane. OpenFlow Specification points out that if a switch has a packet buffer, a miss-matched packet can be buffered at the switch and sent to the controller with the abstract of the packet. But the current packet-granularity buffer in existing switches has deficient in buffer management, which leads many packets in the same flow are sent to the controller as Packet_in messages. It also causes packets out-ordering, which would seriously affects the quality of network transmission. In this paper, we enhance the dumb and fast data plane with well-managed flow-granularity buffers to reduce the communication load on OpenFlow channel and guarantee order-preserving packets forwarding. Our work makes a reference to the buffer design of high-performance OpenFlow switches.