计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(9):238−245 [doi: 10.15888/j.cnki.csa.005970]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

SDN 网络虚拟化中规则映射研究①

李　佟[1,2,3], 韩春静[1], 李　俊[2]

[1](中国科学院 信息工程研究所, 北京 100093)
[2](中国科学院 计算机网络信息中心, 北京 100190)
[3](中国科学院大学, 北京 100049)

摘　要: 软件定义网络(SDN)为网络虚拟化提供了新的解决方案, 通过网络虚拟化技术可以将一套基础设施虚拟化为多个逻辑网络从而满足不同的网络需求. 本文研究了 SDN 网络虚拟化时多个物理交换机虚拟为一个大虚拟交换机的过程中, 虚拟网络规则与物理网络规则的映射问题. 综合考虑链路负载、规则分布以及节点负载, 提出了三段式规则映射优化算法. 首先根据虚拟网络的规则请求生成组播源节点和目的节点集, 采用 MPH 算法生成规则映射树; 然后采用入节点最近原则, 将虚拟网络规则请求的指令序列部署到规则映射树中的中间节点和叶子节点中; 最后考虑节点负载, 对规则部署进行微调, 最终生成虚拟规则映射策略. 通过仿真实验, 与直接边缘节点部署相比, 平均降低了网络节点规则总数量 40% 以上.

关键词: 软件定义网络; 网络虚拟化; OpenFlow; 规则映射; 大交换机

引用格式: 李佟,韩春静,李俊.SDN 网络虚拟化中规则映射研究.计算机系统应用,2017,26(9):238–245. http://www.c-s-a.org.cn/1003-3254/5970.html

Research of Rule Mapping on Network Virtualization in SDN

LI Tong[1,2,3], HAN Chun-Jing[1], LI Jun[2]

[1](Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China)
[2](Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China)
[3](University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: Software Defined Network (SDN) provides a new solution for network virtualization, which can virtualize a set of infrastructures to multiple logical networks to meet different network requirements. This paper studies the mapping of virtual network rules to physical network rules when multiple physical switches are virtualized into one big switch in SDN network virtualization. Considering link load, rule distribution and node load, this paper proposes a three-stage rule mapping optimization algorithm. First, according to the rules of the virtual network, the multicast source node and a destination node set are located, and a rule mapping tree is generated by using the MPH algorithm. Then, the rules of the virtual network rules are deployed to multiple nodes of the physical network. Finally, with the consideration of the node load, the rule deployment is finely-tuned to generate the virtual rule mapping strategy. Simulation results show that compared with direct egress node deployment the average number of network node rules is reduced by more than 40%.

Key words: SDN; network virtualization; OpenFlow; rule mapping; big switch

近年来, 软件定义网络(SDN)在网络研究与应用领域取得了飞速的发展, 软件定义网络技术采用集中式控制, 将网络的控制平面与数据平面完全解耦, 从而使得网络从封闭走向了开放; 网络应用可以根据自己的需求定义访问规则、路由策略、甚至定义并实现创新型网络协议[1,2]. OpenFlow 作为软件定义网络技术中使

① 基金项目: 中国科学院先导专项(XDA06010306); 国家基金委青年基金项目(F020802)
收稿时间: 2016-12-30; 采用时间: 2017-01-26

用最为广泛的技术之一, 在负载均衡、虚拟机迁移、访问控制以及广域网流量调度等领域得到了实现和应用. 目前的各种开源控制器如 OpenDaylight、ONOS、Floodlight 等将网络设备单独管理, 网络应用需要了解网络拓扑结构与网络状态, 并需要将网络应用策略转化为底层设备的单条规则单独下发, 这种模型使得应用开发者需要维护底层网络设备与链路信息, 包括路径选择, 单个交换机上的流表与流表空间等, 增加了网络应用开发的复杂度.

对于 SDN 网络应用来说, 需要控制器提供良好的底层抽象, 并维护底层网络设备状态, 应用仅仅关心数据转发规则策略定义、网络负载以及路由策略等. 根据不同应用场景, 网络应用关心的高层策略可以分为如下两个方面:

1) 端点策略

端点规则将全网视为一个“大交换机”(Big Switch), 应用开发人员仅仅需要关注网络边缘端点策略的定义, 如入端口, 处理策略, 出端口等策略内容, 而不需要关注网络底层的拓扑结构, 网络内部的流量与策略不需要由上层应用处理, 由控制器完成网络内部的路由选择定义、链路负载均衡等内容.

2) 路由策略

路由策略关注了两个设备之间的路由与负载, 路由策略的定义通常由流量工程优化目标来生成, 可以根据最小时延、最大利用率等相关优化目标生成路由策略.

为了使一套网络基础设施能够满足不同的应用需求, 引入了网络虚拟化以及切片技术. 网络虚拟化的核心目标是通过虚拟化技术, 将一个物理网络分割为多个虚拟网络, 或者将多个物理网络设备的部分转发能力提取并抽象为一个逻辑虚拟网络, 不同的虚拟网络之间的服务质量能够得到不同级别的保障, 从而为上层的应用提供网络服务[3]. 网络虚拟化打破了网络物理设备和罗辑业务层之间的绑定关系, 并且能够将物理网络进行切片(slicing)[4,5], 满足更加复杂的业务需求. SDN 技术是网络虚拟化的一个新思路和新方法[3], 已经在全球网络创新试验环境(GENI)[4]以及数据中心[5]中开展了研究与应用.

本文就 SDN 网络虚拟化中多个物理设备虚拟成为一个虚拟设备, 也就是“大交换机”中的规则映射问题进行了分析, 研究了在大交换机中流表规则映射模型, 对 Openflow 中的指令(Instruction)和动作进行分类, 提出多个物理设备虚拟为一个虚拟大交换机场景下的规则映射算法.

1 相关研究工作

1.1 基于 SDN 的网络虚拟化中间件技术

基于中间件的网络虚拟化方案是在 SDN 设备和控制器之间添加一个虚拟化层, 从物理 SDN 设备来看, 网络虚拟化中间件就是它的控制器; 从虚拟网络的控制器角度来看, 控制器所面对的就是一系列虚拟的交换机以及他们之间的虚拟链路. 在目前基于 OpenFlow 的虚拟化领域中, FlowVisor[7]、OpenVirteX[8]、CoVisor[9]、FlowN[10]等采用了这种思想实现了网络虚拟化.

1.2 SDN 规则映射以及优化技术

文献[11]关注于大交换机的网络抽象和大交换机的流表规则映射问题, 这种网络抽象将整个网络视为一个交换机从而实现网络的扁平化管理, 从应用的角度来看, 整个网络是一个交换设备, 简化了移动性管理、网络访问策略管理等工作. 该文献提出了大交换机中端点策略的分配问题, 采用路径探测的方式将端点策略在路径上进行分配来减少网络边缘节点的流表数量.

文献[12]提出的 Palette 分布式框架关注于分布式流表, 将端点策略表拆分为多个子表并将其分布在多个交换机上, 转发保证每个数据包能够在子表中经过一次, 这样就等价于在边缘节点上处理了流表规则中的所有动作. 由于 Palette 采用了最短路径优先的策略实现流表的分布, 这可能导致某个节点中流表规则聚集从而造成负载过重.

2 本文研究内容

本文针对网络虚拟化中的大交换机虚拟化模式, 对文献[11]与文献[12]中的规则映射进行了扩展, 充分考虑了 Openflow 中流表匹配规则、不同指令与动作类型以及流表中多出端口等情况. 提出了三阶段虚拟流表映射算法 MPH-DRP(Minimum Path Heuristic Dynamic Rule Placement), 并进行了原型实现、仿真与验证.

3 SDN 网络虚拟化中的规则映射模型

本文就 SDN 网络中的规则映射模型定义如下:

拓扑: 拓扑是指 SDN 基础设施网络, 可以用二元组 G(V, E)来定义, 其中 V 是交换机节点集合, E 为交换机节点间链路集合, 可以定义节点容量函数: Capacity(n):V->R^+其中, R^+为非负实数集; 链路容量函数 Cost(e):E->R^+其中, R^+为非负实数集.

交换机: 交换机可以定义为一个网络上的节点, 每个交换机包含一系列的规则列表[r_1, r_2, ……, r_k], 其中 r_i 为交换机上的规则.

规则: 规则 r 可定义为 Match->Instructions. 其中, Match 为匹配, Instructions 为指令序列. 一个数据包 p 与 Match 匹配后按顺序执行指令序列中的序列.

匹配: 匹配(Match)是一个匹配列表, 可以包括入端口匹配以及数据包字段匹配两个部分. 形式化表示为 Match:{ingressPort=h1, matchField=(m_1, m_2…m_n)}. 其中 ingressPort 定义为某个交换机上的一个入端口, ingressPort 所在的交换机定义为 ingressNode, matchField 为需要匹配的字段集合.

指令: 指令(Instruction)是数据包匹配后所操作的指令, 通常, 指令以指令序列的形式来执行, 可以形式化表示为 Instructions:{($instructionSet_1$, Output=$Output_1$), ($instructionSet_2$, Output=$Output_2$)…. ($instructionSet_n$, Output=$Output_n$)}, 该指令序列由 Output 动作进行拆分, 表示一个数据包在经过指令操作后在多个端口转发. 我们将 Output 端口定义为出端口(egressPort), 出端口所在的交换机定义为出节点(egressNode).

不失一般性, 本文中将 OpenFlow 中的 ApplyActions 中的系列动作统一看作特殊的指令来进行处理. 考虑到指令可能对数据包的修改, 这里将指令分为两类: 数据包修改指令 $Instruction_W$ 与数据包只读指令 $Instruction_R$.

$Instruction_w$ 是指对数据包的数据字段进行修改的指令与动作, 以 Openflow1.3 为例, 包括 AddFiled 指令、RemoveFiled 指令以及 SetField 指令等相关指令.

$Instruction_R$ 是指对数据包内容没有进行修改的指令与动作. 如 ApplyMeter、ApplyCounter、GotoTable 等相关指令.

规则映射: 一个规则映射是将应用对虚拟大交换机的规则请求映射成为底层物理交换机上的规则的过程:

r_v-> R_{phy}. 其中 R_{phy} 是物理交换机上的规则集合 R_{phy}={r_{phy}| r_{phy} 是物理交换机上的一条流表规则}.

映射模型: 给定物理网络拓扑 G 和一个应用请求 r_v, 给出物理交换机上的规则集合 R_{phy}, 使其对数据包 p 的操作结果等价, 即 $r_v(p)=R_{phy}(p)$.

规则映射树: 在进行规则映射时, 一条虚拟规则最终映射为物理规则时, 需要参与转发的节点和链路构成的转发树, 成为规则映射树 T.

映射树路径代价: 给定网络 G(V, E)和规则请求 r_v, 生成一个包含 ingress 端口所在节点 s 以及 egress 端口所在节点集合 D 的树 T, 该树 T 的路径代价可以分为:

路径代价:

$$Cost_{Edge}(T)=\sum_{v\in D}\sum_{e\in P(s,v)}Cost(e)$$

节点代价:

$$Cost_{Node}(T)=\sum_{v\in D}|r_v| \quad (1)$$

最优化目标: 在给定网络容量条件下, 采用最小的代价将对大交换机的规则请求 r_v 映射到物理拓扑中, 给出 R_{phy}.

4 基于 MPH-DRP 的 SDN 规则映射

根据第三章的定义, 将一个大交换机的规则请求 r_v 映射到物理网络中求出 R_{phy} 是一个 NP 难的问题. 这是因为, 在将请求 r_v 映射到物理拓扑的过程中, 如果在 r_v 的指令序列中 Output 动作前不包括任何操作指令, 则该问题退化成为一个组播树生成最优化的问题, 而组播树生成最优化问题为一个 NP 难问题[13], 因此, 大交换机规则映射问题也是一个 NP 难问题.

将一个虚拟规则映射到物理拓扑中, 我们可以根据第三章的模型定义将大交换机的一个虚拟规则转化为一系列物理交换机上的物理规则, 这些物理规则对数据包 p 的作用结果与虚拟规则作用的结果一致, 包括匹配(Match), 指令(Instruction)序列. 其中虚拟规则的匹配($Match_v$)被分散在多个物理节点上生成物理匹配规则($Match_{phy}$), 虚拟规则的指令(Instruction)序列, 被分散在多个物理节点上, 这些物理节点上的规则作用于数据包后, 与虚拟规则作用等价.

为了优化规则映射树生成, 本文采用三阶段的方法对大交换机的流表规则请求映射进行优化. 首先, 确定 Output 指令所在的 egressNode 集合作为组播的接收节点, 将 ingressPort 所在的 ingressNode 作为组播源,

采用 MPH 算法生成规则映射树; 第二, 对指令序列中的非 Output 指令进行处理, 采用入端口最近规则部署(IngressPort Nearest Rule Placement)原则, 根据指令前缀递归遍历转发树, 在规则映射树节点中分配非 Output 指令所在位置. 最后, 遍历规则映射树对节点规则部署进行微调, 如果步骤二中某些规则所在的节点的儿孙节点只有一个儿子节点, 则该规则可以放置在当前节点或者其某个儿孙节点上, 从而避免规则映射树退化成为链表时规则聚集在入口节点, 实现节点的负载均衡.

阶段一: 基于 MPH 的规则映射树生成

提取出虚拟拓扑指令 r_v 中的 Output 动作, 确定 Output 动作所在交换机, 生成规则映射树, 该规则映射树生成问题等价于组播树生成问题.

组播树生成优化问题已经有了比较深入的研究, 比较典型的算法有 MPH[13]算法、KMB[14]算法、ADH[15]算法等等, 其中时间复杂度分别为 $O(mn^2)$, $O(n^3)$以及 $O(mn^2+e)$. MPH(Minimum Path Cost Heuristic)是大家讨论得较多的一种方法 , 同时, 也是一个非常优秀的求解组播树的算法. 在文献[13]中, S Ramanathan 把 MPH 作为比较组播树算法的一种标准算法. 几种算法的优化目标都是组播树的总费用, 同时, 优于组播树生成算法是 NP 难问题, 几种算法都是基于启发式来生成组播树的近似解. 基于以上考虑本文采用 MPH 算法作为规则映射树生成算法.

转发组播树生成过程如下:

1) 假设 $T=<V_t, E_t, C_t>$为部分组播树, 初始化为只包含 ingressPort 所在的交换机 s, 此时 $V_t=\{s\}$, $E_t=\Phi$, $C_t=\Phi$.

2) 在余下的 egressPort 所在的端节点 $D\backslash V_t$ 中搜索出到当前最小生成树 T 的路径最短的端节点 u(若有多个符合条件则任取一个即可), 将从 u 到 T 的最短路径并入生成树 T(端节点 u 及路径上的所有节点并入 V_t, 该路径上的边并入 E_t, 对应边的费用并入 C_t).

3) 重复步骤 2)直到 D 中的所有节点都包含到 V_t 中, 产生的 T 为近似最小成本组播树.

阶段二: 入端口最近规则部署

该阶段将指令序列中非 Output 指令在阶段一中生成的映射树中部署, 这里我们采用指令前缀递归遍历规则映射树的方法完成指令放置, 尽量将指令部署在距离源端口距离近的节点, 从而减少指令在 egressPort 的聚集并减少整个网络中规则的总数量, 使得物理节点上的规则数最少.

根据前文所述, 非 Output 指令包括两种类型, $instruction_W$ 与 $Instruction_R$ 两种类型. 由于 $Instruction_W$ 会对数据包进行修改, 这种操作将导致对匹配内容的修改, 不失一般性, 本文中对修改后的数据包匹配采用将原 Match 形式化变换为 Match', 后续节点的匹配以 Match'为基准进行数据包匹配, Match'的生成可以由原 Match 以及 $instruction_W$ 共同生成, 本文关注于指令放置, 对 Match'生成不进行深入讨论.

入端口最近规则部署主要步骤如下:

1) 根据 Output 指令, 对 r_v 中的指令进行汇聚, 决定每个指令对应的 egressPort 列表, 放置于 insEgressPortsmap 数据结构中.

2) 将所有非 Output 指令放置于 instructions 列表数据结构中, 指令部署指针放置于 instructions 列表表头, 并将遍历指针至于规则映射树的根节点;

3) 从根节点开始, 采用递归的方法放置 instructions 中的各个指令. 当该指令出现在当前遍历节点下所有 egressPort 上时, 将该指令放置于该节点上, 指令部署指针指向下一个指令, 并重复 3)遍历当前指针的所有儿子节点; 否则重复步骤 3)遍历当前指针的所有儿子节点.

算法伪代码描述如下:

```
//所有非 output 指令放入 instruction 列表
instructions=allInstruction\outputInstruction;
deployInsOnNode(curNode, insIndex){
//所有指令部署完成, 退出递归
if(insIndex== instructions.size())
   return;
ins= instructions.get(insIndex);
//当前 instruction 作用于该节点所有 egressNode 上
if(onAllEgressNode(curNode, ins)){
   curNode.putInstruction(ins);
   index++;
   deployInsOnNode(curNode, insIndex);
   }else{
   for(childNode: node.getChildren()){
      deployInsOnNode(childNode, index);
   }
}
}
```

阶段三: 规则映射树上指令部署优化

该阶段基于阶段二的指令部署结果, 根据规则映射树结构及其节点负载情况, 深度优先遍历规则映射树, 进一步优化节点规则部署. 对规则映射树生成过程中退化为链表数据结构的节点上的规则进行进一步优化.

指令部署优化阶段伪代码描述如下:

```
//规则备选集以及节点备选集
ruleCandidates=new List();
nodeCandidates= new List();
optimizeRule(curNode){
  if(curNode is egressNode){
    //将规则备选集的规则部署在节点备选集上
    putRuleOnNodes(curRules, candidateNodes);
    ruleCandidates.removeAll();
    nodeCandidiates.removeAll();
    return;
  }
  //将当前节点加入备选节点
  nodeCandidites.add(curNode);
  If(curNode.childSize== 1){
    //该节点只有一个儿子节点, 加入备选集
    ruleCandidate.add(curNode.rules);
    optimizeRule(curNode.child);
  }else{
    //该节点有多个儿子节点
    //将规则备选集的规则部署在节点备选集上
    putRuleOnNodes(curRules, candidateNodes);
    ruleCandidates.removeAll();
    nodeCandidiates.removeAll();
    //递归遍历该节点的儿子节点
  for(childNode of curNode){
    optimizeRule(childNode);
    }
  }
}
```

以如图 1 的基础设施网络拓扑为例说明指令的放置过程, 其中, 节点以大写字母 A、B、C、D 等进行命名, 节点括号中数字表示当前该节点的规则容量.

对于规则 r_v= Match->Instructions;

其中 Match={ingressport=(a, match)|a 为节点 A 上的一个 ingress 端口}

Instructions={Ins_1, Output=e, Ins_2, Output=g, ins_3, egerss=Output, ins_4, Output=b|e, g, h, b 分别为 E, G, H, B 节点上的 egress 端口}.

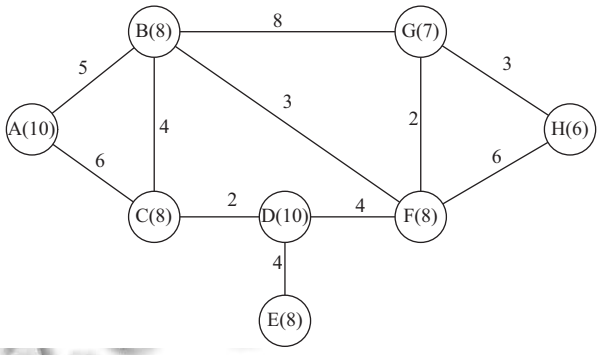

图 1 基础设施网络拓扑图

数据包 p 经过规则 r_v 处理后, 根据 Instructions 指定的转发出口, 该数据包需要分别在在 e、g、h、b 四个端口转发出 SDN 网络. 首先, 对于该拓扑结构执行步骤一, 基于 MPH 算法, 以 A 为入节点, E、G、H、B 出节点, 生成规则映射树, 生成的规则映射树如图 2 所示.

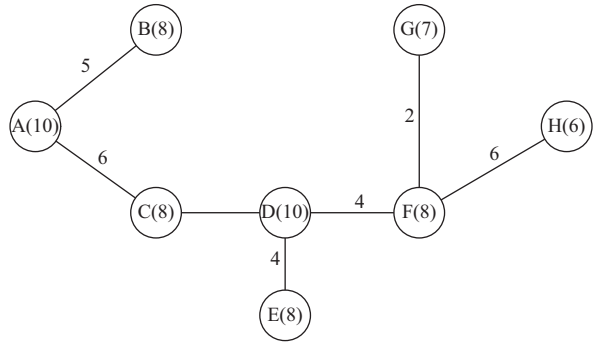

图 2 基于 MPH 的规则映射树

生成规则映射树后, 转发路径与出端口需要执行指令如表 1 所示.

表 1 边缘端口路径与指令表

边缘端口	路径	执行指令
e	A C D E	Ins1
g	A C D F G	Ins1, Ins2
h	A C D F H	Ins1, Ins2, Ins3
b	A C B	Ins1, Ins2, Ins3, Ins4

执行步骤二, 经过入端口最近规则部署算法处理后, 指令被分配于规则映射树中, 放置情况如表 2 所示.

表 2 MPH-INRP 算法指令部署表

网络节点	放置指令
A	Ins1
C	Ins2
H	Ins3
B	Ins2, Ins3, Ins4

对于当前指令放置执行步骤三. 可以看出经过步骤二后, 节点 C 只有一个儿子节点, 因此指令 Ins2 可以部署在 C 节点或者 D 节点上, 而 D 节点容量较大, 则 Ins2 将被优化部署在节点 D 上. 执行步骤三后指令部署如表 3 所示.

表 3 MPH-DRP 算法指令部署表

网络节点	放置指令
A	Ins1
D	Ins2
H	Ins3
B	Ins2, Ins3, Ins4

5 算法评价与实验分析

本问题提出的 SDN 规则映射分为三个独立的步骤, 映射树生成步骤、指令放置步骤以及指令优化步骤. 对映射树生成步骤, 是典型的 MPH 组播树生成的算法, 设基础拓扑的节点个数为 n, egressPoint 个数为 m, 根据文献[13]其算法时间复杂度为: $O(m^2n+e)$.

对于步骤二, 设映射树中节点数量为 m, 指令序列中除去 Output 指令的指令个数为 k, 那么, 对于指令序列的遍历需要执行 k 次, 对于转发组播树的节点遍历需要执行 m 次, 因此, 最坏情况下, 需要对每个节点进行 k 次遍历, 复杂度为 $m*k$, 同时在循环中需要判断 Instruction 是否作用于每个端点, 需要执行 m 次循环, 因此步骤二最坏情况下的时间复杂度为 $O(m^2k)$.

对于步骤三, 需要对映射树进行深度优先遍历, 在最坏的情况下, 映射树退化为一条 n 个节点链表, k 个非 Output 指令在步骤二中部署在入口节点. 需要将 k 个非 Output 指令部署在 n 个节点的链表上, 这里我们将 k 个非 Output 指令部署于负载最轻的节点, 复杂度最坏情况下为 $O(n)$.

因此整个算法的时间复杂度为 $O(m^2n+e+m^2k+n)$, 即 $O(m^2(n+k)+e+n)$.

由于基于 MPH 算法的组播树生成问题已经有比较深入的研究, 本文对映射树生成不进行深入探讨. 在实验中, 我们实现了 MPH-DRP 两端式规则放置算法, 并实现了文献[16,17]中提出的将规则直接部署于 egerssNode 算法, 为了便于比较, 我们称之为 MPH-ERP(Minimum Path Heuristic Egress Rule Placement), 实验中将 MPH-ERP 与 MPH-DRP 进行了如下两种实验比较:

实验一. 采用 6*6 网格形的基础设施拓扑结构, 经过大交换机虚拟化之后, 在拓扑结构中选择随机选择一个入端口, 并在基础设施中随机选择 4 个出端口作为 egressPort 上进行数据包转发, 并定义 instruction 序列为: Instructions={Ins_1, Output=p_1, Ins_2, Output=p_2, ins_3, Output=p_2, ins_4, Output=$p_4|p_i$ 分别为大交换机的 egress 端口}.

在虚拟网络中部署 101 条虚拟指令, 每部署一条指令后, 每次记录所有节点上分配的规则总数量以及基础设施拓扑中规则最多节点中部署的规则数量, 并进行比较, 仿真实验结果如图 3、图 4 所示.

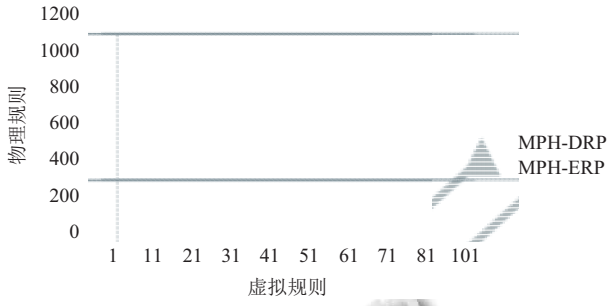

图 3 随机端口选取部署基础设施规则总数量

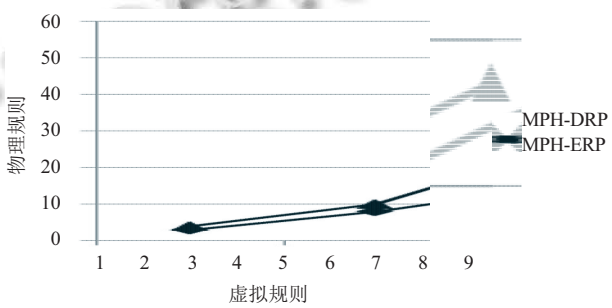

图 4 随机端口选取部署负载最重节点规则数量

在随机部署的条件下, MPH-DRP 算法的总规则数量平均为 MPH-ERP 部署的 43.45%, 在基础设施中负载最大节点规则数量方面, MPH-DRP 算法的最大负载节点的负载平均为 MPH-ERP 部署的 60.07%. 性能有显著提升.

实验二. 与实验一相同, 采用 6*6 网格形的基础设施拓扑. 经过大交换机虚拟化之后, 隐藏了所有中间节

点, 在拓扑结构中选择随机选择边缘节点上的端口作为入端口, 并在基础设施 6*6 个节点中随机选择 4 个边缘节点的端口作为 egressPort 进行数据包转发, 定义 instruction 序列为: Instructions={Ins1, Output=p1, Ins2, Output=p2, ins3, Output=p2, ins4, Output=p4|pi 分别为大交换机的 egress 端口}.

在虚拟网络中部署 101 条虚拟指令, 每部署一条指令后, 记录所有节点上分配的物理规则总数量以及基础设施拓扑中物理规则最多节点的规则数量, 实验结果如图 5、图 6 所示.

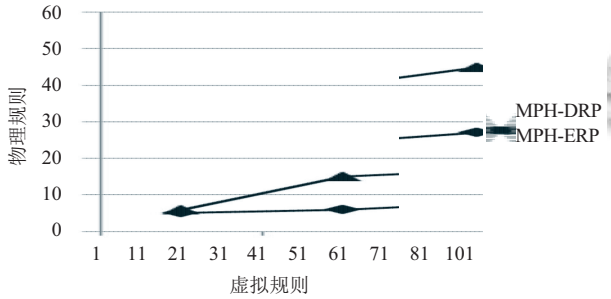

图 5 随机边缘端口选取部署基础设施规则总数量

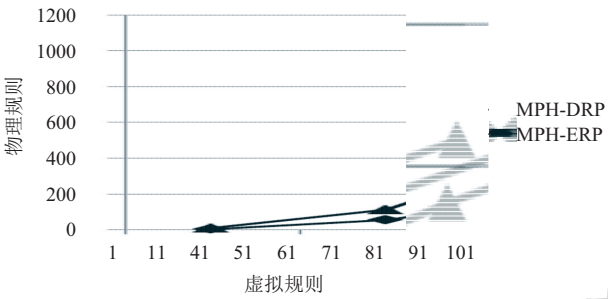

图 6 随机边缘端口选取部署负载最重节点规则数量

在随机边缘端口部署的条件下, MPH-DRP 算法的总规则数量平均为 MPH-ERP 部署的 44.69%, 对于基础设施中负载最大节点规则数量方面, MPH-DRP 算法的最大负载节点的负载平均为 MPH-ERP 部署的 56.61%. 性能有显著提升.

6 结语与进一步研究工作

本文针对网络虚拟化中大交换机流表映射问题, 提出了 MPH-DRP 算法, 该算法采用两阶段方案, 通过组播树生成算法构造转发树, 并采用入端口最近优先的规则部署策略, 优化了大交换机虚拟化中一个端口进入, 多端口转发并执行相关指令的问题. 该算法与边缘规则部署相比, 性能有显著的提升.

本文中提出的 MPH-DRP 算法在出端口仅为一个时, 退化为文献[11,12]中的规则部署问题, 本文中将规则部署与负载最轻的节点, 还没有进一步优化在链表上的规则部署, 因此算法还有优化空间, 提升基础设施容量.

参考文献

1 McKeown N, Anderson T, Balakrishnan H, *et al*. Openflow: Enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 2008, 38(2): 69–74. [doi: 10.1145/1355734]

2 Song HY. Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane. Proc. of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking. Hong Kong, China. 2013. 127–132.

3 Kabir Chowdhury NM, Boutaba R. A survey of network virtualization. Technology Report CS-2008-25. Waterloo: University of Waterloo, 2008.

4 Berman M, Chase JS, Landweber L, *et al*. GENI: A federated testbed for innovative network experiments. Computer Network, 2014, 61: 5–23. [doi: 10.1016/j.bjp.2013.12.037]

5 Sherwood R, Chan M, Covington A, *et al*. Carving research slices out of your production networks with OpenFlow. ACM SIGCOMM Computer Communication Review, 2010, 40(1): 129–130. [doi: 10.1145/1672308]

6 Bari MF, Boutaba R, Esteves R, *et al*. Data center network virtualization: A survey. IEEE Communications Surveys & Tutorials, 2013, 15(2): 909–928.

7 Sherwood R, Gibb G, Yap KK, *et al*. FlowVisor: A network virtualization layer. Stanford: Deutsche Telekom Inc. R&D Lab, 2009.

8 Al-Shabibi A, De Leenheer M, Gerola M, *et al*. OpenVirteX: Make your virtual SDNs programmable. Proc. of the Third Workshop on Hot Topics in Software Defined Networking. Chicago, Illinois, USA. 2014. 25–30.

9 Jin X, Gossels J, Rexford J, *et al*. CoVisor: A compositional hypervisor for software-defined networks. Proc. of the 12th USENIX Conference on Networked Systems Design and Implementation. Oakland, CA, USA. 2015. 87–101.

10 Drutskoy DA. Software-defined network virtualization with flown[MSc. thesis]. Princeton: Princeton University, 2012.

11 Kang NX, Liu ZM, Rexford J, *et al*. Optimizing the “One big switch” abstraction in software-defined networks. Proc. of

the Ninth ACM Conference on Emerging Networking Experiments and Technologies. Santa Barbara, California, USA. 2013. 13–24.

12 Kanizo Y, Hay D, Keslassy I. Palette: Distributing tables in software-defined networks. Proc. IEEE INFOCOM. Turin, Italy. 2013. 545–549.

13 Ramanathan S. Multicast tree generation in networks with asymmetric links. IEEE/ACM Trans. on Networking, 1996, 4(4): 558–568. [doi: 10.1109/90.532865]

14 Shaikh A, Shin K. Destination-driven routing for low-cost multicast. IEEE Journal on Selected Areas in Communications, 2006, 15(3): 373–381.

15 Waxman BM. Routing of multipoint connections. IEEE Journal on Selected Areas in Communications, 1988, 6(9): 1617–1622. [doi: 10.1109/49.12889]

16 Casado M, Freedman MJ, Pettit J, *et al*. Rethinking enterprise network control. IEEE/ACM Trans. on Networking, 2009, 17(4): 1270–1283. [doi: 10.1109/TNET.2009.2026415]

17 Ioannidis S, Keromytis AD, Bellovin SM, *et al*. Implementing a distributed firewall. Proc. of the 7th ACM Conference on Computer and Communications Security. Athens, Greece. 2000. 190–199.