计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(5):10−16 [doi: 10.15888/j.cnki.csa.006334]

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

协议无关的数据中心网络源路由机制研究①

常　坤, 赵　敏, 刘　磊, 田　野

(中国科学技术大学 计算机科学与技术学院, 合肥 230027)
通讯作者: 田　野, E-mail: yetian@ustc.edu.cn

摘　要: 传统数据中心网络已经不能满足当前大规模网络和云计算架构的需求, 传统数据中心网络的路由策略会导致转发单元的臃肿; 同时当规模增大时传统网络中的拓扑管理策略也不再适用. 协议无感知转发技术是软件定义网络中转发平面的一种创新技术. 本文结合源路由和协议无感知转发技术, 提出两种数据中心网络的关键技术: 首先, 设计一种协议无关的源路由机制, 从而简化转发单元; 其次, 提出一种主机和控制器间协作的拓扑管理算法, 从而减少探测包的冗余. 最后, 本文在数据中心网络中实现了以上技术, 实验结果表明本文提出的源路由机制可以有效降低转发单元的流表规模, 拓扑管理策略可以极大地减少探测包的冗余.

关键词: 软件定义网络; 数据中心网络; 协议无感知转发; 源路由

引用格式: 常坤,赵敏,刘磊,田野.协议无关的数据中心网络源路由机制研究.计算机系统应用,2018,27(5):10–16. http://www.c-s-a.org.cn/1003-3254/6334.html

Research on Protocol Oblivious Source Routing for Data Center Network

CHANG Kun, ZHAO Min, LIU Lei, TIAN Ye

(School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)

Abstract: Traditional Data Center Network (DCN) cannot satisfy large-scale networks and cloud-oriented infrastructure. Routing policies in traditional DCN complicate the forwarding elements and the topology management is inflexible in large-scale DCN. Protocol Oblivious Forwarding (POF) is a groundbreaking forwarding plane technology. In this study, we proposed two technologies of DCN. We designed extremely simple routing mechanism with POF. Forwarding elements of DCN can process packets easily using simple source routing protocol. Furthermore, we designed a cooperative topology management strategy to reduce redundancy of probe packets. We implemented our technologies in POF-based DCN and conduct some experiments. Results demonstrate that our routing mechanism can effectively simplify the forwarding elements and our topology management strategy can reduce the probe packets redundancy.

Key words: software-defined networking; data-center network; protocol oblivious forwarding; source routing

软件定义网络[1](Software-Defined Networking, SDN) 是一种新型的网络架构, SDN 将传统网络中耦合的控制平面和数据平面分离开来, 网络中的转发单元仅包含数据平面. 使用集中式的控制平面来为用户提供灵活可编程的接口, 上层应用使用这些接口对网络中的匹配转发、资源调度等进行可编程的集中式管理. OpenFlow[2,3]是斯坦福大学提出的一种 SDN 的实现技术. OpenFlow 交换机使用流表 (Flow Table) 进行匹配转发. 然而 OpenFlow 不能支持新协议, 虽然它的规范在不断的更新, 并通过增加字段的方式来支持更多的协议, 但是这种被动式的增长会导致协议的臃肿和数据平面硬件的不断重新设计.

① 基金项目: 国家自然科学基金(61672486); “新一代宽带无线移动通信网”国家科技重大专项子课题(2017ZX03001019-004); 安徽省自然科学基金(1608085MF126)
收稿时间: 2017-08-24; 修改时间: 2017-09-15; 采用时间: 2017-09-22; csa 在线出版时间: 2018-03-12

为了解决 OpenFlow 存在的问题, 华为提出了协议无感知转发[4–7](Protocol Oblivious Forwarding, POF) 技术. 在 POF 中, 数据平面通过{type, offset, length}的三元组形式标识协议字段, 不需要理解具体的协议格式; 同时使用统一的协议无关指令集进行匹配转发, 彻底做到数据平面的协议无感知. 因此 POF 可以方便的支持任意新协议而无需修改数据平面, 从而使得 SDN 的控制平面和数据平面彻底解耦.

数据中心网络[8,9]广泛应用于虚拟化、云计算等领域. 然而, 随着网络规模的扩张和需求的增加, 传统的路由机制使用的基于 IP 地址的最长前缀匹配的方法会造成转发单元的复杂和转发表的爆炸[10], 从而影响网络的性能; 同时随着网络规模的增大, 传统的拓扑学习策略导致大量探测包的冗余, 严重消耗了网络的带宽资源.

目前一种基于源路由的数据中心网络的实现是 Sourcey[11]. Sourcey 在数据中心网络中使用源路由进行路由转发, 同时提供基于主机的拓扑发现策略. 然而 Sourcey 提出的拓扑发现策略中, 所有主机都要探测全局的网络拓扑, 该策略会造成非常多的探测包冗余, 严重影响网络的性能.

针对以上问题和现状, 本文提出两个协议无关的数据中心网络关键技术. 首先, 利用 POF 支持任意协议的特性, 设计极其简单的源路由机制实现数据中心网络中的源路由, 从而极大地简化数据平面, 避免匹配转发的时间成为数据中心网络中的瓶颈; 其次, 设计一种主机和控制器协作的拓扑管理算法, 能够有效的减少探测包的冗余, 减轻控制器和每个主机的负载.

本文余下的内容安排为: 第 1 节介绍本文提出的源路由机制; 第 2 节介绍主机和控制器协作的拓扑管理策略; 第 3 节介绍实验内容; 第 4 节总结工作.

1 基于 POF 的数据中心网络路由机制

1.1 源路由

在源路由中, 主机通过在数据包头部添加路由信息来为数据包规划转发路径. 网络中的数据平面根据这些路由信息进行转发. 与传统的路由协议相比, 源路由大大简化了数据平面的复杂度.

当前已有的源路由策略包括传统的源路由和基于 OpenFlow 的源路由. 传统的源路由策略是基于 IPv4 的: 主机将由 IPv4 地址组成的路由信息添加在数据包 IPv4 协议层的 Options 中, 网络中的路由器根据其中某个 IPv4 地址选择下一跳进行转发. 传统源路由的缺点是必须与 IPv4 协议绑定在一起, IPv4 中的其余字段会造成额外的开销, 同时 Options 字段最长只有 40 字节, 因此不能用于较大规模的网络中.

当前有很多基于 OpenFlow 的源路由研究[12–15]. 其中一种常见的方法是是基于 MPLS[16]的源路由: 使用多个 MPLS 标签, 每个标签代表一跳路由, 网络中的数据平面根据某个 MPLS 标签进行转发. 这种方法的问题是数据平面支持的 MPLS 标签数目是有限的, 一般为 3~5 个, 因此不具有扩展性, 在大规模网络中很难使用这种方法. 另一种方法[17]是在 OpenFlow v1.3 及其之后的版本[18,19]中, 对字段进行匹配时可以使用任意 bit 长度的掩码, 因此使用 IPv6 等字段保存每一跳的信息. 然而, 这种方法同样包含很多不需要使用的字段, 增加了协议长度, 同时也面临扩展性的问题, 例如若网络中交换机端口数目最多为 256, 则 128 bit 的源 IP 字段最多只能支持 16 跳源路由.

1.2 基于 POF 的源路由机制

基于 OpenFlow 的源路由如此复杂的根本原因, 是 OpenFlow 不能够支持自定义的协议. 本文提出的基于 POF 的源路由策略, 充分利用 POF 协议无感知的特性, 定义新协议 POFSR 如图 1 所示, POFSR 包括标志字段 Flag 和一系列的 SRlabel 字段. Flag 为 0 时表示 POFSR 协议, 否则为其它数据包. 字段 SRlabel 是 8bit 长度的字段, 每个 SRlabel 包含一跳的路由信息, 这些 SRlabel 组成了完整的源路由信息.

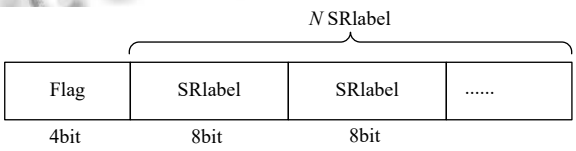

图 1 POFSR 协议格式

使用图 2 和图 3 来比较以上介绍的源路由方法. 图 2 是一个基于 POF 的数据中心网络, 由主机、POF 交换机和 POF 控制器三部分组成. 若图 2 中的 h1 要发送数据包给 h4, 则根据以上三种源路由协议, h1 需要组建的三种数据包如图 3 所示. 使用传统源路由时所需的数据包如图 3(a) 所示, 在 IPv4 的 Options 字段中包括了规划路径上的全部 5 个 IP 地址; 图 3(b) 表示使用 OpenFlow 的源路由时组建的数据包, h1 将所有路由端口写在 IPv6 协议的源 IP 地址的不同 bit 中, 转

发路径上的每一跳路由匹配该字段中的不同 bit 位; 而如图 1(c) 所示, 在基于 POF 的源路由中, h3 仅需组建 POFSR 协议, 将路径上每一跳的端口号分别放入 SRlabel 中, 而不需要任何多余的协议或字段. 路径上的每个 POF 交换机只需要读取一个 SRlabel 字段, 根据其中的端口字段的值进行匹配转发即可. 综上可知, 借助 POF 支持任意协议的特点, POFSR 协议做到了最简单的源路由.

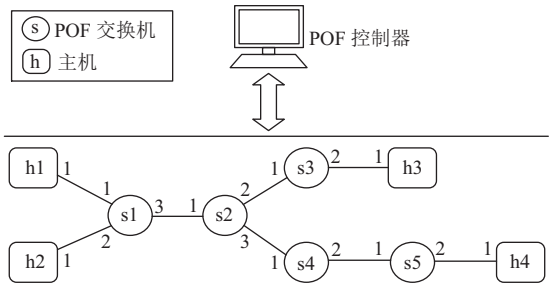

图 2 基于 POF 的数据中心网络

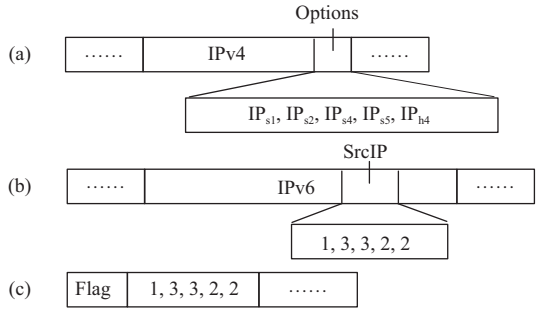

图 3 不同格式的源路由数据包

同时, 设计 SRProbe: 用于拓扑管理的探测数据包的协议格式. 如图 4 所示, 为了区分 POFSR 和 SRProbe, 规定当 Flag 字段为 0 时表示头部协议为 POFSR 的负载数据包 (简称负载包), 否则表示头部协议为 SRProbe 的探测数据包 (简称探测包). 图 4 中的 Hop 字段代表要探测的跳数, Own 字段代表发出该探测包的主机的编号. 最后是多个 SRProbe 标签, 该标签包括三个字段: 表示交换机 (或主机) 编号的 dpid, 代表数据包进入端口的 inp, 代表发出端口的 outp. 在 2.1 小节具体描述 POF 交换机如何处理探测包.

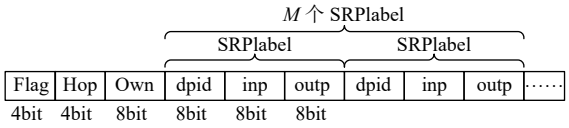

图 4 SRProbe 协议格式

2 拓扑管理

为了使用源路由, 每个主机都需要掌握全局的网络拓扑从而能够组建源路由的数据包. 本章详细描述主机和控制器协作的拓扑管理算法, 包括初始时的拓扑发现和运行中的拓扑监控两部分.

2.1 拓扑发现

拓扑发现的基本思想是: 初始时每个主机各自发送探测包进行拓扑发现, 通过 POF 交换机对探测包的过滤来减少冗余, 通过 POF 控制器来整合每个主机学习到的部分拓扑, 最后下发给所有主机. 在拓扑发现中每个角色的任务如下.

(1) 主机: 主机的拓扑发现基于广度优先搜索 (Breadth-First-Search, BFS) 的思想: 对于每个主机, 首先构造 Hop=1 的探测包来发现距自己一跳的拓扑, 然后构造 Hop=2 的探测包来发现距自己两跳的拓扑, 以此类推. 每个探测包在从主机出发后, 随着 Hop 的递减不断向前探索, 称这一阶段为前进阶段; 而当探测包的 Hop=0 或其它条件下, 探测包会沿着原路径返回直至主机, 我们称这一阶段为响应阶段, 这一阶段的探测包称为响应包.

算法1. HostTopo(h, Hop, LocalTopo[])

Input: host *h*, int *Hop*=0, topology list *LocalTopo*
Output: ToController

Steps:

```
1.  if has received SRProbe of other host then
2.      ToController(NULL);
3.  Hop = Hop + 1;
4.  pkt = construct_pkt(1, Hop, h.dpid);
5.  for i in h.ports do
6.      send(pkt);
7.  while time < LIMIT_TIME_H do
8.      if receive SRProbe packet pkt then
9.          add_topo(LocalTopo, pkt);
10. if received nothing or only boundary packet then
11.     ToController(LocalTopo);
12. Goto 3;
```

算法 1 是主机的拓扑发现算法, 首先判断是否响应过其它主机的探测包, 若是则将空拓扑上交给控制器并停止拓扑发现; 否则, 在第 4 行, 开始第一轮的拓扑发现. 第 7 行设置等待时间为 LIMIT_TIME_H, 该时间内主机监听所有端口, 若收到响应包, 解析该响应包并将其中的拓扑信息加入 LocalTopo.

主机可能收到的响应包分为两类: Flag=1 的响应

包和 Flag=2 的响应包, 其中前者是由于 Hop 自减到 0 后返回的正常响应包, 而后者是由于 POF 交换机的过滤而导致在 Hop 为 0 之前就返回的响应包, 称为边界响应包, 稍候将具体描述边界响应包. 如第 10 行所示, 如果在探测某一跳的拓扑时, 主机没有收到任何响应包, 或者只收到了边界响应包, 则停止拓扑发现, 将学到的拓扑信息上交给控制器.

(2) POF 交换机: 需要处理负载包和探测包两种数据包. 如算法 2 所示, 对于负载包, POF 交换机仅仅需要使用 SRForward 指令, 该指令是自定义的 POF 指令, 使用源路由的策略: 剥离第一个 SRlabel, 并根据该字段的值进行转发; 而对于探测包, POF 交换机首先判断其是否来自于自己第一次收到的探测包所属主机.

若是, 如第 7 行所示. 则接下来读取 Hop 的值: 若 Hop=0, 说明这是返回阶段的响应包, 使用 SRForward 处理; 若 Hop 为 1, 说明这是前进阶段的最后一跳, 如第 13~16 行所示, POF 交换机将自己的信息加入数据包并使用 SRForward 处理; 若 Hop 大于 1, 说明处于前进阶段, 则使用 SRFlood 指令: POF 交换机对于自己的每个端口, 在数据包里添加其相应的信息并转发出去.

若不是, 即该探测包所属的主机不是当前 POF 交换机第一次收到的探测包所属的主机. 为了避免探测包的冗余, 此时不应该继续向前转发: 如 18 行所示, POF 交换机通过 Own 字段判断是否是第一次收到该数据包所属的主机发来的探测包, 若是, 则将 Flag 置 2 后使用 SRForward 将探测包沿着来时的端口回送, 这样处理的目的是告诉源主机本 POF 交换机已经被别的主机探测过; 若不是, 则直接丢弃该探测包.

算法2. ProcessPkt(sw, pkt, inport)

Input: switch *sw*, packet *pkt*, ingoing port *inport*
Output: Forward or Drop
Steps:

```
1.  if pkt.Flag == 0 then
2.      SRForward(pkt);
3.  else
4.      if sw.Own == 0 then
5.          sw.Own = pkt.Own;
6.      if pkt.Own == sw.Own then
7.          if pkt.Hop == 0 then
8.              SRForward(pkt);
9.          else if pkt.Hop == 1 then
10.             pkt.Hop = 0;
11.             add_label(sw.dpid, inp, inp);
12.             double_label(pkt);
13.             SRForward(pkt);
14.         else
15.             SRFlood(sw, pkt, inp);
16.     else
17.         if see pkt.Own for the first time then
18.             pkt.Flag = 2;
19.             pkt.Hop = 0;
20.             add_label(sw.dpid, inp, inp);
21.             double_label(pkt);
22.             SRForward(pkt);
23.         else
24.             Drop(pkt);
```

Procedure SRForward(*pkt*)

```
1.  if pkt.Flag == 0 then
2.      port = extract(pkt.SRlabel);
3.  else
4.      port = extract(pkt.SRPlabel);
5.  forward(port);
```

Procedure SRForward(*pkt*)

```
1.  for i in sw.ports do
2.      if i != inport then
3.          add_label(sw, dpid, inport, i);
4.          forward(i);
```

(3) POF 控制器: 如算法 3 所示, POF 控制器在 LIMIT_TIME_C 时间内监听所有主机发来的局部拓扑, 并将这些拓扑合并为 GlobalTopo, 最后将 GlobalTopo 下发给所有主机.

算法3. ControllerTopo(h[], GlobalTopo[])

Input: host list *h*, global topology list *GlobalTopo*
Output: NULL
Steps:

```
1.  while time < LIMIT_TIME_C do
2.      if received LocalTopo[] then
3.          GlobalTopo.add(LocalTopo);
4.  for host in h[] do
5.      send(GlobalTopo);
```

2.2 拓扑发现举例

结合图 2 和图 5 来详细介绍 3.1 小节的拓扑发现算法. 假设在图 2 中, h1 首先开始拓扑发现, 于是 h1 组建如图 5(a) 所示的探测包并从所有端口发出, 只有 s1 会收到该探测包, 根据算法 2, s1 组建图 5(b) 所示的响应包并从端口 1 返回, h1 收到该响应包后解析, 从而学习到自己的端口 1 和 s1 的端口 1 之间存在一条链路.

接着, h1 组建图 5(c) 所示的 Hop=2 的探测包来探测距自己 2 跳的拓扑, s1 使用 SRFlood 组建图 5(d) 所示的两个探测包并从端口 2 和 3 洪泛. s2 处理该探测包并返回图 5(e) 所示的响应包. 在经过 s1 的转发后,

h1 最终收到图 5(f) 所示的响应包, 并解析出 s1 到 s2 的链路. 同时 h2 也收到 s1 转发来的探测包, 假设此时 h2 还没有拓扑发现, 于是与 s2 类似, h2 也回送响应包, 如图 5(g) 所示. 最终 h1 学习到 s1 和 h2 之间的链路.

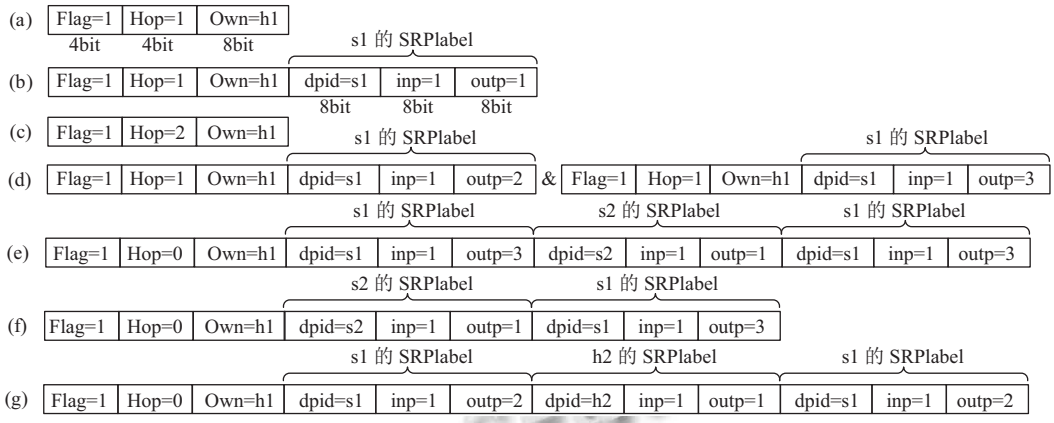

图 5 不同阶段的探测包和响应包

下面描述主机和控制器的协作. 假设在某一时刻, 主机 h1~h4 的拓扑发现状态为: h1 学习到了距自己 2 跳的拓扑, 即 s1, s2 和 h2; h2 由于响应了 h1 的探测包, 因此不进行拓扑发现; h3 学习了距自己 1 跳的拓扑, 即 s3, 而 h4 学习了距自己 2 跳的拓扑, 即 s5 和 s4. 根据算法 1 和算法 2, 所有主机在进行下一步的探测时, 都只能收到边界响应包, 这是因为所有的 POF 交换机都被探测过了, 对于其它主机的探测包, 所有交换机都只会响应边界响应包. 图 6 表示经过下一跳的探测后, 每个主机探测到的局部拓扑. 因此所有主机都会结束拓扑发现, 并将拓扑上交给 POF 控制器. 根据算法 3, POF 控制器将所有局部拓扑合并为全局的网络拓扑并下发给所有的主机.

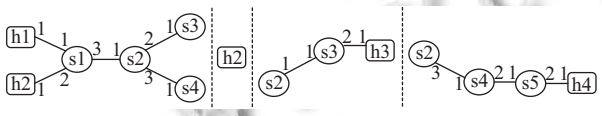

图 6 每个主机学习的局部拓扑

环路: 由于拓扑发现基于 BFS 并且逐跳的学习, 数据包在环路间循环到 Hop 为 0 时会原路返回, 因此并不会引起广播风暴. 另外, 通过为每个探测包分配一个编号即可避免探测包的重复, 交换机在收到编号相同的探测包且 Hop 不为 0 时就丢弃该数据包.

2.3 拓扑监控

拓扑监控的目的是维护最新的全局拓扑, 主要包括发现新链路和结点、探测失效的链路和结点. 与看重效率的拓扑发现相比, 拓扑监控需要的是时效性.

在拓扑发现中的协作方法仍然适用于拓扑监控: 由于每个主机在初始的拓扑发现时都学习了局部拓扑, 因此在拓扑监控时, 每个主机只负责自己的局部拓扑的监控, 通过控制器的协作来获取全局的拓扑变化.

每个主机间歇性的重复拓扑发现的步骤, 如果学习到的拓扑与上一次不一样, 则将变化的部分上交给 POF 控制器, 控制器将这些变化下发给所有主机, 这样所有主机就学习到了拓扑的变化. 例如图 7 是图 2 所示的拓扑发生了变化, 其中 s4 和 s5 之间的链路失效了, 并多出了结点 s6 和其与 s1 的链路. 则 s1 在拓扑监控中进行 2 跳的探测时就会探测到 s6 结点和 s1 到 s6 的链路, 而原先负责探测 s4 的主机 h4 在进行拓扑监控时就会探测到 s4 与 s5 链路的消失.

由于每个主机只需要负责部分拓扑的监控, 因此宏观上, 相当于所有主机同时进行全局网络拓扑的监控, 因此时延较低, 具有很好的时效性.

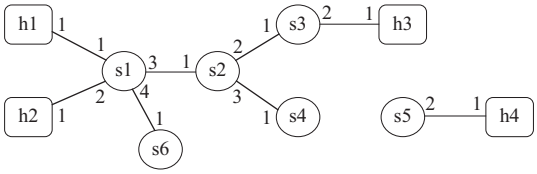

图 7 发生变化后的拓扑

3 实验

使用 Mininet[20]设计并部署基于 POF 和源路由的数据中心网络. 使用 POF 控制器[21]和基于华为的软件

版本 POF 交换机部署该数据中心网络中的控制器和交换机. 实验的主要目的是验证: (1) 本文设计的源路由机制可以有效减少转发平面中流表项的数目; (2) 拓扑发现算法可以有效减少探测包的冗余.

3.1 流表大小

本文提出的基于 POF 的源路由机制使用极简的源路由协议 POFSR, 转发平面只需要使用简单的源路由对数据包进行处理, 因此能够简化处理逻辑, 减小转发平面中流表的大小.

本实验中, 我们搭建 k=4(pods) 的 Fat-Tree[22]网络拓扑并在其中实现了 POFSR 协议. 为该拓扑中的主机编写脚本使它们能够为数据包添加 POFSR 头部. 为了验证目标, 使用每个主机与其它的所有主机进行通信, 统计该场景下 Fat-Tree 的不同层次中每个交换机流表项的数目, 并且与使用 OpenFlow v1.0 实现的成对单播 (pairwise unicast) 路由产生的流表项数目进行比较.

表 1 列出了对于 Fat-Tree 的不同层次, 基于 POF 的源路由机制和成对单播的流表项数目. 从表格中可以看出, 对于 Fat-Tree 的三层, 基于 POF 的源路由机制都能极大地节省流表项的数目. 这是因为其只需要匹配 SRlabel 字段, 而成对单播需要匹配源地址、目的地址等多个协议字段. 转发单元中处理逻辑的简化和流表项数目的减少可以有效的减少处理时间, 从而避免处理时间成为大规模数据中心网络的瓶颈.

表 1 流表项数目对比

	基于POF的源路由	成对单播
Core	4	84
Aggregation	4	48
Edge	4	58

3.2 探测包冗余

为了验证本文提出的拓扑发现策略可以有效减少探测包的冗余. 我们使用 Mininet 搭建不同规模的网络拓扑, 在这些拓扑中实现 3.1 小节的拓扑发现策略. 使用 POFOX 作为 POF 控制器并且编写程序来实现主机与控制器的消息交互. 计算不同规模的网络中的探测包数目, 并与同等条件下的 Sourcey[9]进行比较.

定义探测包的数目为: 前进阶段的探测包和返回阶段的响应包的总和. 其中前者是由主机和 POF 交换机 (使用 SRFlood) 产生的, 而后者是根据算法 2 产生的. 本实验中, 将主机的数量 m 设为 16, 网络的维度 d 设为 8. 图 8 和图 9 显示了探测包的数目随交换机的个数和最大端口数变化的趋势图. 从图中可以看出, 不论是随着交换机数目的增加, 还是随着最大端口个数的增加, 本文提出的拓扑发现方法产生的探测包都远远少于同等条件下 Sourcey 产生的探测包. 由此证明本文提出的拓扑发现策略可以极大地减少探测包的冗余, 节约网络的带宽资源.

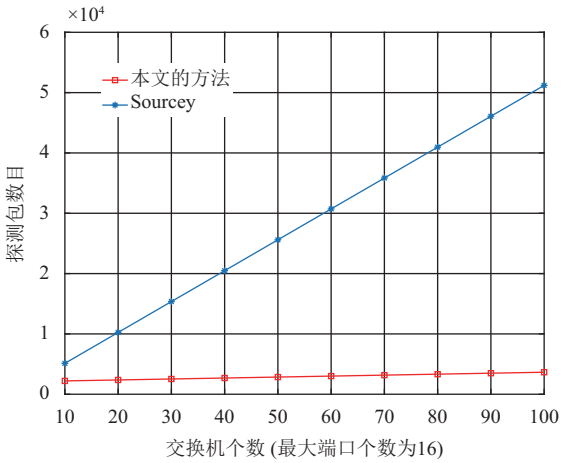

图 8 探测包随交换机个数变化对比图

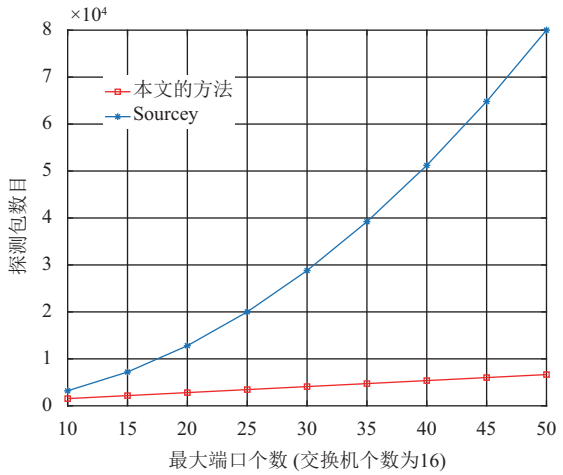

图 9 探测包随最大端口数变化对比图

4 结论与展望

本文通过对数据中心网络的现状进行分析, 结合协议无感知转发技术和源路由, 提出了两个数据中心网络的关键技术: 首先, 设计极其简单的源路由机制, 从而简化数据平面, 有效减少流表的规模; 其次, 提出一种协作的拓扑管理算法, 通过主机和控制器的协作和交换机的过滤来较少探测包的冗余, 提高拓扑发现的效率. 最后, 我们在 Mininet 中搭建数据中心网络并

实现了以上两点技术. 实验结果表明, 我们提出的关键技术可以有效地降低转发平面中流表的规模, 极大地减少探测包的冗余. 接下来的研究工作是完善拓扑管理算法. 此外, 基于协议无感知转发的数据中心网络将会是一个有价值的重要研究方向.

参考文献

1 Casado M, Freedman MJ, Pettit J, *et al*. Ethane: Taking control of the enterprise. ACM SIGCOMM Computer Communication Review, 2007, 37(4): 1–12. [doi: 10.1145/1282427]

2 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]

3 张朝昆, 崔勇, 唐澍祎, 等. 软件定义网络 (SDN) 研究进展. 软件学报, 2015, 26(1): 62–81. [doi: 10.13328/j.cnki.jos.004701]

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

5 Song HY, Gong J, Chen HF, *et al*. Unified POF programming for diversified SDN data plane. Eprint arXiv: 1405.0060, 2014.

6 Yu JZ, Wang XZ, Song J, *et al*. Forwarding programming in protocol-oblivious instruction set. Proceedings of the 22nd International Conference on Network Protocols. Raleigh, NC, USA. 2014. 577–582.

7 POF: Specification of Huawei's POF controller and switch. http://www.poforwarding.org/. [2017-09-20].

8 Niranjan Mysore R, Pamboris A, Farrington N, *et al*. Portland: A scalable fault-tolerant layer 2 data center network fabric. ACM SIGCOMM Computer Communication Review, 2009, 39(4): 39–50. [doi: 10.1145/1594977]

9 李丹, 陈贵海, 任丰原, 等. 数据中心网络的研究进展与趋势. 计算机学报, 2014, 37(2): 259–274.

10 Greenberg A, Hamilton J, Maltz DA, *et al*. The cost of a cloud: Research problems in data center networks. ACM SIGCOMM Computer Communication Review, 2009, 39(1): 68–73.

11 Jin X, Farrington N, Rexford J. Your data center switch is trying too hard. Proceedings of the Symposium on SDN Research. Santa Clara, CA, USA. 2016. 12.

12 Alizadeh M, Edsall T, Dharmapurikar S, *et al*. CONGA: Distributed congestion-aware load balancing for datacenters. ACM SIGCOMM Computer Communication Review, 2014, 44(4): 503–514. [doi: 10.1145/2740070]

13 Ramos RM, Martinello M, Rothenberg CE. SlickFlow: Resilient source routing in Data Center Networks unlocked by OpenFlow. Proceedings of the 38th Annual IEEE Conference on Local Computer Networks. Sydney, NSW, Australia. 2013. 606–613.

14 汪正康, 周鹏, 肖俊超, 等. 基于 SDN 的数据中心网络资源调度机制. 计算机系统应用, 2015, 24(8): 212–218.

15 Guo CX, Lu GH, Wang HJ, *et al*. SecondNet: A data center network virtualization architecture with bandwidth guarantees. Proceedings of the 6th International Conference. Philadelphia, PA, USA. 2010. 15.

16 Rosen E, Viswanathan A, Callon R. Multiprotocol label switching architecture. RFC No. 3031, 2001.

17 Jyothi SA, Dong M, Godfrey PB. Towards a flexible data center fabric with source routing. Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research. Santa Clara, CA, USA. 2015. 10.

18 OpenFlow switch protocol: Provides an open interface for controlling connectivity and flows within that connectivity in SDN. https://www.opennetworking.org/. [2017-09-20].

19 OpenFlow. OpenFlow switch specification version 1.5.1. http://t.cn/R0btma9. [2015-03-26].

20 Lantz B, Heller B, McKeown N. A network in a laptop: Rapid prototyping for software-defined networks. Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. Monterey, CA, USA. 2010. 19.

21 谈小冬, 邹山, 郭浩然, 等. 面向协议无感知转发技术的 SDN 试验床. 计算机系统应用, 2016, 25(4): 237–241.

22 Al-Fares M, Loukissas A, Vahdat A. A scalable, commodity data center network architecture. ACM SIGCOMM Computer Communication Review, 2008, 38(4): 63–74. [doi: 10.1145/1402946]