软件定义网络基于分段路由的多路径调度算法 *

李 艺†，唐 宏，马枢清

(重庆邮电大学 移动通信技术重庆市重点实验室，重庆 400065)

摘 要：针对当前 SDN 网络在应对大量数据流时造成的流表利用率低，转发响应较慢以及当前网络调度算法容易造成网络局部拥塞和负载不均衡等问题，提出一种基于分段路由的多路径调度算法 SRMF。首先，SDN 控制器根据网络拓扑连接情况下发初始流表；然后，综合考虑网络链路剩余带宽、丢包率和数据流估测带宽需求进行路径权重计算；最后，根据路径权重选择最优路径并构造分段流表下发到边缘交换机。实验结果表明分段路由转发技术在多种网络拓扑下较一般转发技术在流表项开销方面有明显优势，SRMF 算法与 Hedera、ECMP 相比，在业务流端到端时延、端到端时延抖动、网络吞吐率、丢包率等方面有一定的优势。

关键词：软件定义网络；分段路由；负载均衡；估测带宽

中图分类号：TP393 doi: 10.19734/j.issn.1001-3695.2020.07.0183

Multi-path scheduling algorithm based on segment routing in software defined network

Li Yi†, Tang Hong, Ma Shuqing

(*Chongqing Key Laboratory of Mobile Communications Technology*, *Chongqing University of Post & Communications*, *Chongqing* 400065, *China*)

Abstract: In view of the low flow entry utilization rate caused by the current SDN network when dealing with a large number of flows, the slow forwarding response and the current network scheduling algorithms may cause local network congestion and load imbalance, a multi-path scheduling based on segment routing is proposed. Algorithm SRMF. First, the SDN controller sends the initial flow entries according to the network topology links information; then, comprehensively considers the remaining bandwidth of the network links, the packet loss rate and the estimated bandwidth requirements of the flow to calculate the path weight; finally, selects the optimal path according to the path weight and install the segment flow entries to the edge switches. Experimental results show that the segment routing technology has obvious advantages in terms of flow entries overhead compared with general forwarding under multiple network topologies. Compared with Hedera and ECMP, the SRMF algorithm has certain advantages in terms of end-to-end delay and delay jitter, network throughput, and packet loss rate.

Key words: software defined network; segment routing; load balance; demand estimation

0 引言

随着云计算，大数据蓬勃发展，一些新兴业务对网络性能提出了更高的要求，云服务提供商正面对传统 IP 网络“资源利用率低，高能耗，自动化程度低”的现状[1]。针对这些问题，数据中心网络拓扑设计得到了大量的研究，一些新型的网络拓扑设计相继提出，用以提高网络的对分带宽，如 Fat-Tree[2]、Bcube[3]、VL2[4]等，同时借助静态的路由算法，如等价多路径(equal cost mutipath routing, ECMP)[5]将数据散列到多条等价路径从而提高网络对分带宽。

研究表明，数据中心网络数据呈现长尾分布，即少数大型数据流占据了多数带宽。数据中心网络中超过 80%的数据流持续时间不超过 10 秒，且 50%的字节数集中在持续时间不超过 25 秒的数据流[6]。文献[7]指出，通过研究 10 个数据中心发现，超过 80%的数据流字节数少于 10KB，90%的数据流小于 10MB。ECMP 确实可以有效调度大量老鼠流，但是容易将多条大流调度到同一路径导致拥塞发生，使得数据中心带宽资源无法充分利用。

软件定义网络(software defined network，SDN)的出现使数据中心网络动态路由调度成为可能，在软件定义网络架构中，交换机的控制能力被转移到 SDN 控制器，仅保留转发能力，控制器可以实时获取网络负载信息并根据实时网络状态和流信息制定细粒度的动态转发策略[8]。然而数据中心网络中数据是海量的，针对每条数据流的调度算法不现实，针对以上特点，研究者针对大小流的区分以及区分大小流动态调度策略做了大量的研究，事实证明，在 SDN 网络架构下合理的区分大小流并针对性调度可以有效提高数据中心网络对分带宽，降低拥塞发生概率。

仅仅关注路由调度算法带来的性能提升是有限的，如在大数据量场景下，控制器负载加重，会造成流表更新不及时从而导致数据流无法及时有效地调度，控制器负载主要体现在以下几个方面：控制器实时获取网络信息的开销；控制器计算路由的开销；流表下发的开销。前两点可以通过设计高效算法，调整监控周期来降低开销，而后者可以通过改进转发技术来降低开销。针对这个现象，分段路由(segment routing，SR)作为一种用于简化网络控制平面的新型技术方案被提出[9]。Internet 工程任务组(Internet Engineerting Task Force，IETF)于 2018 年 7 月确定了分段路由建议标准 RFC8402[10]。移除了 MPLS 在控制平面上复杂的 LDP(label distribution protocol)和 RSVP-TE(resource reservation protocol-traffic engineering)协议，由 SDN 控制器集中进行标签分配[11]。当前，随着 Ipv6 和 Segment Routing 结合，演进出了 SRv6 技术，作为新一代

收稿日期：2020-07-22；**修回日期：**2020-09-09 **基金项目：**长江学者和创新团队发展计划(IRT_16R72)

作者简介：李艺(1995-)，男(通信作者)，山东滨州人，硕士，主要研究方向为软件定义网络(lee_work2020@163.com)；唐宏(1967-)，男，四川南充人，教授，博士，主要研究方向为主要研究方向为计算机网络、移动通信；马枢清(1995-)，女(土家族)，重庆市人，硕士，主要研究方向为软件定义网络.

IP 网络承载协议，SRv6 统一了复杂的多种网络协议，是构建 IP 网络自动化的基础[12]。

1　相关工作

在 SDN 网络架构下，许多研究者采取区分老鼠流和大象流的路由策略。

文献[13]首先将所有数据流通过 ECMP 等价多路径转发，SDN 控制器实时轮询交换机获得网络链路信息和流信息，将数据流速率超过链路带宽 10%的定义为大流，并针对大流进行理想传输带宽估计，根据大流带宽需求估计值采用 Global First Fit(全局首次适应，GFF)算法找到第一条满足带宽需求的路径转发，但是频繁的流重路由会造成数据包乱序现象，且加重的控制器负担，。

文献[14]的实现基于数据中心管理员对数据中心内部使用的操作系统拥有自主控制的权利，在服务器操作系统内核实现一种在 TCP 缓存中标记大流的机制，当检测到 TCP 缓存数据流超过 100KB，则将分组的 TOS 字段改为 000011XX 从而标记为大流，边缘交换机匹配大流后将首包发送到控制器然后找到一条剩余带宽最大路径转发。终端侧检测大流提高了实时性且降低了控制器负载，但容易造成误判。

文献[15]提出针对小流采用预先安装流表的方式，通过动态加权多路径转发，针对大流采用一种 blocking Island 算法缩小搜索空间。该算法降低了大流转发的响应时间，提高了网络带宽利用率。

文献[16]提出将深度增强学习机制应用于 SDN 选路过程，用以改进 Q-learning 造成的存储资源浪费。所提算法在降低时延，提高网络吞吐量同时，实现了连续时间上的黑盒优化，与传统路由算法相比表现出更好的性能和稳定性。

基于分段路由转发技术的调度策略，可以实现快速的业务流重调度，降低流表资源开销。文献[17]分别针对单链路失效和多链路失效提出 NHSR-FRR 和 FDSR-FRR 快速重路由技术，前者检测到链路失效后从链路前一跳到下一跳间找到一条有效拼接路径，根据 SR 技术，后续数据在前一跳压入新拼接标签栈，然后由新拼接路径转发；后者在检测到链路失效后从链路前一跳计算到目的地址的新拼接路径，同样后续数据在前一跳压入新拼接路径的标签栈进行转发。两种算法是 SR 在 FRR 下的尝试，证明了其可行性，提高了重路由的实时性。

2　基于分段路由的多路径调度算法

本章首先介绍一种基于本地端口的分段路由技术，然后阐述本文所提多路径调度算法，包括大象流老鼠流的判别机制，网络状态信息检测，数据流带宽需求估计，路由算法设计，以及流表构造与下发。

2.1　分段路由工作机制

分段路由是 MPLS 技术和 SDN 结合的产物，其工作机制如下：

在边缘交换机将路由压入标签栈到数据包，通过逐节点的匹配栈顶标签和标签出栈最终将数据包送达目的交换机。中间节点不必为所有可能经过它们的流维持状态信息，所有的状态信息以标签栈的形式存在于数据包之中，而且流表更新只发生在边缘节点。

a)网络初始化阶段，SDN 控制器通过 PAKCET_OUT 下发 LLDP 包给所连交换机，交换机收到 LLDP 包发送给相连交换机，相连交换机收到 LLDP 包触发 PACKET_IN 到控制器，控制器获取 PACKET_IN 信息并解析 LLDP 进而获得网络连接信息，最终抽象为有向图数据结构 G(V,E)，其中 V 是网络中所有节点，E 为所有链路。

b)如表 1 所示，控制器根据有向图邻接关系，为中间节点构建初始标签匹配流表。由于控制器针对网络拓扑图集中计算出的路径是一条表明了前后节点连接关系序列，所以可以根据连接关系得到对应链路，如 R1 与 R2 的链路为 $Link_{R1,R2}$，若 R1 又通过 eth1 连接 R2，则在 R1 上 $Link_{R1,R2}$ 可以表示为 eth1，所以一条路径可以表示为一串本地端口序列，如表 1 和图 1 所示，在节点 R1 上，R1-R2 链路在 R1 上可由本地端口 eth2 表示，同样，R2-R4 在 R2 上可由 eth1 表示。

表 1　网络邻接矩阵

Tab. 1　Network adjacency matrix

网络节点	R1	R2	R3	R4	R5	R6
R1	null	eth2	eth1	null	null	null
R2	eth2	eth2	eth2	eth1	null	null
R3	eth3	eth2	null	null	eth1	null
R4	null	eth4	null	null	eth2	eth1
R5	null	null	null	eth2	null	eth1
R6	null	null	null	eth2	eth3	null

c)新数据到达边缘交换机时，由于没有匹配流表，触发 PACKET_IN 进入控制器，控制器为数据流计算一条路径，进而转换为由本地端口序列组成的标签栈，并下发到边缘节点压栈流表，这个流表匹配当前数据的五元组信息，动作为压入标签并由第一个出端口发出。数据流在中间各节点依次匹配默认流表，直到恢复为 IP 数据包到达目的交换机，完成转发，如图 1 所示。

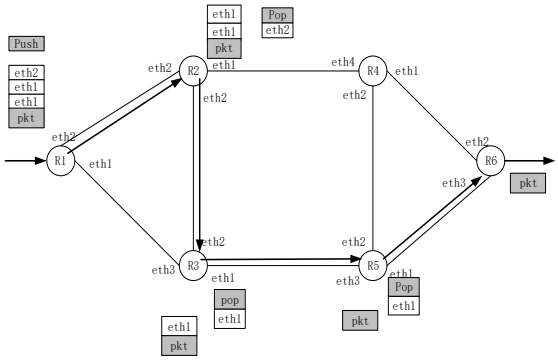

图 1　分段路由转发过程

Fig. 1　Segment routing and forwarding process

2.2　系统架构

基于分段路由的多路径调度系统分为三大模块：网络探测模块，实时监控模块和流表安装模块，其中每个模块又包含多个功能模块，具体如图 2 所示。

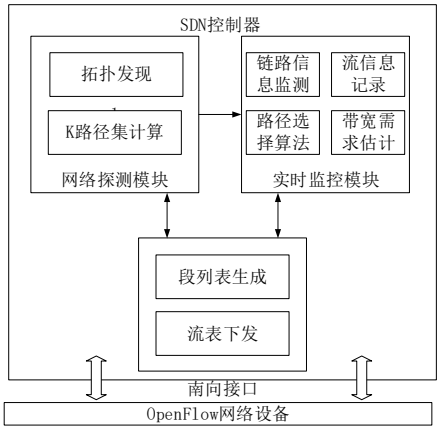

图 2　系统架构

Fig. 2　System architecture

2.2.1 网络探测模块设计：

网络探测以一定的周期 T 执行，它主要来获取网络拓扑，

继而获得各节点对的K条最短路径并保存供实时路由选择直接调用，具体功能实现如下：

a) 控制器通过 OFPT_HELLO 与交换机建立连接，并定期发送 OFPT_ECHO 与交换机保持通信。控制器通过 OFPT_FEATURES_REQUEST 请求交换机信息，交换机通过 OPFT_FEATURES_REPLY 回复给控制器相关信息。

b) 控制器获取网络拓扑并下发默认流表，如 2.1 中所述。

c) SDN 控制器根据有向图 G(V,E)以跳数为权重采用 Dijkstra 算法计算各边缘节点对的 K 条最短路径并保存，当网络拓扑发生变化时重新计算。

d) SDN 控制器根据有向图 G(V,E)，为每个中间节点依据邻接关系下发标签转发默认流表，当网络拓扑发生变化时更新。

2.2.2 实时监控模块设计：

该模块主要进行实时业务处理，如实时链路信息监测，数据包解析，流带宽需求估计以及路径的计算。

a) 链路信息监测通过控制器以 t 为周期发送 OFP_PORT_STATS_REQUEST 给交换机，交换机回复 OFP_PORT_STATS_REPLY 消息给控制器，控制器解析消息体可以获得端口信息包括 rx_packets，tx_packets，rx_bytes，tx_bytes，rx_dropped，tx_dropped 以及 duration_sec，duration_nsec 等信息。其中 rx 表示接收，tx 表示传输，duration_sec 和 duration_nsec 表示时间戳。根据这些信息可以进一步计算端口实时传输速率，丢包率等。

链路 a 速率计算公式如下：

$$speed_a(t)=\frac{tx^a_{bytes}(t)-tx^a_{bytes}(t-\Delta t)}{\Delta t}\times 8 \tag{1}$$

其中 $tx^a_{bytes}(t)$ 和 $tx^a_{bytes}(t\text{-}\Delta t)$ 分别表示 t 时刻和 $t-\Delta t$ 记录的传输字节数。

链路 a 丢包率：

$$loss_a(t)=\frac{tx^a_{dropped}(t)-tx^a_{dropped}(t\text{-}\Delta t)}{tx^a_{packet}(t)-tx^a_{packet}(t-\Delta t)} \tag{2}$$

其中 $tx^a_{dropped}(t)$ 和 $tx^a_{dropped}(t-\Delta t)$ 分别表示 t 时刻和 $t-\Delta t$ 记录的丢包数。$tx^a_{packet}(t)$ 和 $tx^a_{packet}(t-\Delta t)$ 分别表示 t 时刻和 $t-\Delta t$ 记录的传输数据包数目。

进而计算链路 a 可用带宽：

$$B_a(t)=B-speed_a(t) \tag{3}$$

其中 B 为端口最大可用带宽，由于单条路径的可用带宽取决于带宽瓶颈的链路带宽，倘若路径 p 上有 a 条链路，则路径 p 的可用带宽：

$$B_p=Min(B_1,B_2,...B_a) \tag{4}$$

路径 p 的丢包率计算公式如下：

$$loss_p=1-\prod_i(1-loss_i) \tag{5}$$

b)带宽需求估计模块：单纯通过当前时刻采集到的的业务流速率难以反映其带宽需求，所以要为业务流的带宽需求进行估测。关于流带宽估计已经得到了研究，Hedera[13]认为在 Nonblocking 网络环境下，流速率仅仅取决于发送和接收端网卡带宽限制，由于 TCP 的 AIMD 机制，多条竞争流会收敛于均分带宽的动态平衡状态。然而 Hedera 的带宽需求估计机制需要建立在业务流已经在网络中传输的情况下，针对新业务流到达时无法适用。DPSOFS[18]中也提到了带宽需求估计，该算法是对 Hedera 所提带宽估计算法的改进，它通过维护流量信息矩阵来计算和调整均分网卡带宽后业务流的估测需求，依然只适用于在业务流已经在网络中传输的情况。本文根据 Hedera 和 DPSOFS 的竞争流均分带宽思想，提出一种快速流带宽估计算法，该算法在首包触发 PACKET_IN 后就能做到实时估测。算法借助字典数据结构存储流信息，利用 OFPT_PACKET_IN 和 OFPT_FLOW_REMOVE 消息来调整流量矩阵，其中字典底层采用哈希表数据结构，所以存储和读取时间复杂度接近 $O(1)$。算法流程如下：

a) 当首保到达控制器时，控制器解析数据包，获得数据包的源 ip 地址(src)和目的 ip 地址(dst),更新流量矩阵相关项业务流数目加 1。

b) 若 OFPT_FLOW_REMOVE 被触发，通过获取流删除消息，解析被删除流的 src 和 dst，更新流量矩阵相关项业务流数目减 1。

c) 若源终端的发送带宽限值为 S_{src}，发送业务流数目为 N_{src}，目的终端的接收带宽限制为 D_{dst}，接收业务流数目为 N_{dst}，根据公式获得带宽估计需求：

$$flow(src,dst)_{demand}=Min\left(\frac{S_{src}}{N_{src}},\frac{D_{dst}}{N_{dst}}\right) \tag{6}$$

即，网卡带宽限值与业务流数目之比表示 TCP 收敛于均分带宽状态时每条业务流所分配带宽，当发送均值不同于接收均值时，取两者较小值作为业务流估测带宽。

3　多路径调度算法设计

由于网络中绝大多数数据流都是存在时间短，数据量小的老鼠流，只有少量数据流是存在时间长，数据量大的大象流，所以一般将老鼠流作为背景流不经过控制器调度。所提算法只针对大象流的调度，并设置小流参数来预留带宽保证小流的传输质量。

大小流区分主要有两种方式：基于终端侧检测和基于边缘交换机检测。其中基于终端侧检测的典型算法是 Mahout 和 Nimble，它们都是在数据中心网络管理员拥有终端设备管理权的背景下，通过修改终端 OS 来监控发送数据缓冲区，标记超过阈值的大流。基于终端侧检测大象流无须控制器参与，对控制器性能开销较小。而基于边缘交换机检测是通过控制器轮询边缘交换机上的流表并计算这些流的传输速率，将超过一定阈值的流定义为大流，这种方式加重了控制器负担且需要安装额外默认流表记载流信息。本文选择 Mahout 所提算法进行大小流区分。

所提算法的基本思想是在数据中心网络下，首包触发 PACKET_IN 时，从最短路径集中选择一条最优路径转发。对于最优路径的选择，仅考虑单一路径开销无法满足多目标优化需求，虽然业务流吞吐率的决定性因素是路径可用带宽，但是丢包率也起着关键作用，路径的丢包率反映了链路质量，节点处理能力和缓冲区队列是否已满，丢包会引起 TCP 重传从而降低 TCP 流速率，同时决定了小流的服务质量。同时，根据 Rfrag[19]指出的网络带宽“碎片化”会造成带宽资源浪费从而影响网络性能，本文从提高带宽利用率出发提出路径可用带宽波动值来评估路径性能。

3.1　SRMF 算法设计

新到来的数据流通过 PACKET_IN 将数据包头信息传递给控制器，控制器解析并获得该数据流的源 ip 地址，目的 ip 地址，经由带宽需求估计后获得其理想传输带宽需求 B_{demand}，然后从相应的 K 条最短路径中找到满足带宽需求的路径集 paths，最后从路径集 paths 中选择最优路径转发：

首先根据式(7)：

$$r(p)=\alpha\times B_p-B_{demand} \tag{7}$$

B_p 为路径 p 可用带宽，$\alpha\in[0.6,0.9]$，即要为小流预留一部分带宽，同时为了避免突发流量造成路径拥塞而提供的一个缓冲空间，当 $r(p)\geq 0$，将路径 p 加入路径集 paths，倘若 paths 为空，随机选择一条路径直接转发。

带宽满足需求无法充分说明一条路径性能，缓冲区队列满和链路故障带来的丢包对数据的吞吐率有显著影响。从这一点出发建立选择函数 c^1_p

$$c_p^1 = \frac{loss_{\max} - loss_p}{loss_{\max} - loss_{\min}} \tag{8}$$

$loss_{\max}$ 表示路径集中最大丢包率，$loss_{\min}$ 表示路径集中最小丢包率，c_p^1 越大越好。

根据路径带宽碎片化现象[19]，由于一条路径的剩余带宽仅取决于最小链路带宽，当路径上各链路带宽波动较大情况下，可能造成大带宽链路带宽资源浪费。据此建立路径带宽波动函数 c_p^2：

$$b_p = \frac{\sum_{i=1}^{a} B_i}{a} \tag{9}$$

$$d_p = \sqrt{\sum_{i=1}^{a} (B_i - b_p)^2} \tag{10}$$

$$c_p^2 = \frac{d_{\max} - d_p}{d_{\max} - d_{\min}} \tag{11}$$

B_i 为路径第 i 条链路的可用带宽，b_p 为路径 p 各条链路的平均可用带宽，d_p 表示路径链路带宽标准差，代表了路径带宽波动情况，波动越小性能越好，$d_{\max}$ 表示路径集 paths 中最大波动值，$d_{\min}$ 表示最小波动值，有式(11)，c_p^2 越大越好。

综合上述，为路径 p 计算综合权重 $weight_p$：

$$weight_p = \mu \times c_p^1 + (1-\mu) \times c_p^2 \tag{12}$$

μ 为权重因子，最终，为数据流选择权重值最大的路径直接转发。伪代码如下所示

输入：路径集合ShortestPaths , 网络拓扑G(V,E) ,
　　源地址src , 目的地址dst;
输出：最优路径bestPath;
1 for p in ShortestPahts(src,dst) do
2 　if $\alpha \times B_p > B_{demand}$
3 　　then paths.append(p)
4 　　if $loss_{\max} < loss_p$
5 　　then $loss_{\max} \leftarrow loss_p$
6 　　if $loss_{\min} > loss_p$
7 　　then $loss_{\min} \leftarrow loss_p$
8 　　if $d_{\max} < d_p$
9 　　then $d_{\max} \leftarrow d_p$
10 　　if $d_{\min} > d_p$
11 　　then $d_{\min} \leftarrow d_p$
12 if paths is Empty
13 　then bestPath ← randomPath
14 for p in paths do
15 　$c1_p^1 \leftarrow (loss_{\max} - loss_p)/(loss_{\max} - loss_p)$
16 　$c2_p^2 \leftarrow (d_{\max} - d_p)/(d_{\max} - d_{\min})$
17 　$weight_p \leftarrow \mu \times c1_p^1 + (1-\mu) \times c2_p^2$
18 　if $weight_{\max} < weight_p$ then
19 　　$weight_{\max} \leftarrow weight_{\partial}$
20 　　bestPath ← p
21 return bestPath

算法首先从网络探测模块的最短路径集中获得 ShortestPahts(src,dst) 即所有 src 到 dst 的路径集。

步骤 1 到 3：将路径集中路径剩余带宽大于业务流带宽需求的加入 paths 集合；

步骤 4 到 11：获取所有路径中的最大丢包率 $loss_{\max}$，最小丢包率 $loss_{\min}$，最大路径链路带宽标准差 $d_{\max}$ 以及最小路径链路带宽标准差 $d_{\min}$

步骤 12 到 13：当没有满足路径剩余带宽大于业务流带宽需求的路径时，即 paths 集合为空，则选择任意路径直接转发。

步骤 14 到 21：否则遍历 paths 中各条路径，依次计算出选择函数 c_p^1 和选择函数 c_p^2，最终计算路径综合权重 $weight_p$，同时依次比较选择出最大综合权重值 $weight_{\max}$ 以及对应的路径 bestPath，即最优路径，返回该路径。

3.2 流表的构造与安装

控制器为数据流选择路径后，将路径信息根据网络拓扑图连接关系转换为本地端口标签信息，并构造流表项，匹配域为数据的五元组信息，动作列表为先将表征路径的一系列端口标签压入 MPLS 标签栈，然后通过第一个端口发出。

事实上流表项安装成功后，数据转发过程会出现又一次的 PACKET_IN。如在传输 TCP 或 ICMP 等数据时，由于发送方和接收方需要持续通信，如 TCP 的接收方需要不断发 ack 确认报文，于是首个 seq 和首个 ack 数据包会先后触发两次 PACKET_IN 且控制器计算两次路由，虽然这样保证了正向和反向流量通过不同最优路径传输，但两次路由计算时间间隔极小，通常小于控制器更新网络链路信息周期，故本文认为只需要计算一次正向路径，后向流量转发路径为正向路径的反转。在流表项安装过程中，首先安装后向流表，然后再安装正向流表，这是由于倘若先安装正向流表，在后向流表还未安装成功这段时间内，正向数据匹配正向流表后立刻发送到目的端，目的端发送反向数据依然触发 PACKET_IN。采用双向流表项安装机制，只需要触发 1 次 PACKET_IN 动作，计算 1 次路由，缩短了数据转发响应时间和控制器用于路径计算的开销。步骤如下：

a) SDN 控制器为数据流计算出路径。

b) 首先将路径反转，构造流表项下发。

c) 重复步骤 b)，即将反转后的路径恢复为正向路径，构造流表项下发。

4 实验仿真与分析

4.1 仿真平台搭建

本文采用轻量级仿真平台 mininet 搭建 Fat-Tree(k=4)，Fat-Tree(k=8)规则数据中心拓扑，以及非规则的 COST266 网络拓扑，各条链路带宽设为 100Mbps。控制器选择基于 Python 的开源 Ryu 控制器，实验平台是 Ubuntu16.0.4，Python 版本 2.7，流量发生软件 Iperf，网络吞吐量记录软件 Bwmng。

4.2 实验设置

实验采用两种通信模式：随机 random 方式和交错方式 Staggered(pEdge,pPod)，其中 pEdge 是同一接入层交换机下业务流占比，pPod 为同一 pod 内不同接入交换机之间业务流占比，不同 pod 间业务流占比为 1-pPod-pEdge。如 stag_0.2_0.3 表示接入层交换机下业务流占比 0.2，同一 pod 不同接入交换机业务流占比 0.3，不同 pod 间流量占比 0.5，random 表示随机产生业务流。

实验中根据通信模式利用 Iperf 产生 TCP 数据流模拟大象流，利用 ping 产生数据流模拟老鼠流，TCP 流和 ping 同时产生，每条 ping 数据流的传输信息写入文件记录，同时在经过一定时间待数据传输趋于稳定后开启 Bwm-ng 线程记录网络参数 60 秒并写入文件。

4.3 算法参数设置

对于第 4 章提到的小流参数 α 和选择参数 μ，设置进行正交实验，首先固定 μ =0.5，按照 random 流量模型产生数据流，计算网络平均吞吐率，小流端到端时延，小流时延抖动，小流丢包率来评估 ∝ 取不同值时网络性能，可以得到当 ∝ 取 0.7 时，网络平均吞吐率最高，且小流端到端时延，小流端到端时延抖动都最低，虽然小流丢包率略高但是总体性能最佳，故 α 取 0.7。结果见表 2。

表 2　α 取不同值时算法性能对比

Tab. 2　Different α algorithm performance comparison

α	平均吞吐率/Mbps	小流时延/ms	小流时延抖动/ms	小流丢包率/‰
0.6	1117	11.37	12.42	0.401
0.7	1127	10.46	10.54	0.414
0.8	1087	11.82	15.13	0.401
0.9	1092	12.43	13.87	0.426

同样，固定 α=0.7，评估 μ 取不同值时网络性能，如表 3 所示，当 μ 取 0.2 时，虽然吞吐率和丢包率表现不是最好的，

但是差距很小，不过小流端到端时延和端到端时延抖动都明显优于其他取值时，故本文将 μ 设为 0.2。

综上所述，所提算法设置为 $\propto$ =0.7，μ =0.2。

表 3　μ 取不同值时算法性能对比

Tab. 3　Different μ algorithm performance comparison

参数 μ	平均吞吐率/Mbps	小流时延/ms	小流时延抖动/ms	小流丢包率(‰)
0.1	1122	10.68	11.60	0.399
0.2	1099	6.27	5.73	0.404
0.3	1069	7.44	9.19	0.400
0.4	1090	8.20	8.45	0.386
0.5	1078	10.14	13.09	0.385
0.6	1082	7.44	7.29	0.400
0.7	1076	9.11	10.37	0.411
0.8	1082	9.60	12.15	0.392
0.9	1082	9.32	11.60	0.406

4.4　仿真结果

所提算法采用所提分段转发技术称为 SRMF，采用默认转发技术称为 NMF，为了验证分段转发技术的可行性和性能，实验将所提 NMF，SRMF 与 Hedera，ECMP 算法做对比，NMF 与 SRMF 对比可以证明相同路由算法下，分段转发技术和一般转发技术作对比，同时 NMF 和 Hedera 以及 ECMP 对比可以获得相同转发技术下路由算法性能差异。

实验分别采取 stag_0.1_0.2，stag_0.2_0.3，stag_0.4_0.3，stag_0.5_0.3，stag_0.6_0.2，stag_0.7_0.2，stag_0.8_0.1(下图中采用如 0.8_0.1 表示 stag_0.8_0.1)和 random 通信模式产生 TCP 数据流，每台主机发送 4 条，同时采取 random 通信模式发送 100 条 ping 数据流模拟小流。在 Fat-Tree 拓扑下，平均吞吐率，标准化平均吞吐率，小流端到端时延，小流端到端时延抖动，小流丢包率如图 3~7 示。其中标准化吞吐率指平均吞吐率与网络最大吞吐率之比。

由图 4 可以看出，所提算法在平均吞吐率方面，NMF 与 Hedera 和 ECMP 相比，在 random 和 stag_0.1_0.2 通信模式下略低于 Hedera 但优于 ECMP，在其他通信模式下 NMF 均领先 Hedera 和 ECMP。由图 5 小流端到端时延可以看出，NMF 在 random 模式下相比 Hedera 和 ECMP 时延较高，但是在其他通信模式下性能表现优异，由图 7 所示丢包率方面，NMF 优于 Hedera 和 ECMP。由图 6 所示 NMF 在时延抖动方面表现一般。

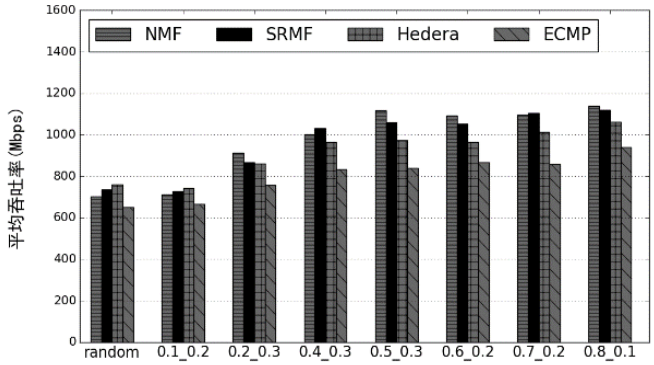

图 3　平均吞吐率

Fig. 3　Average throughput

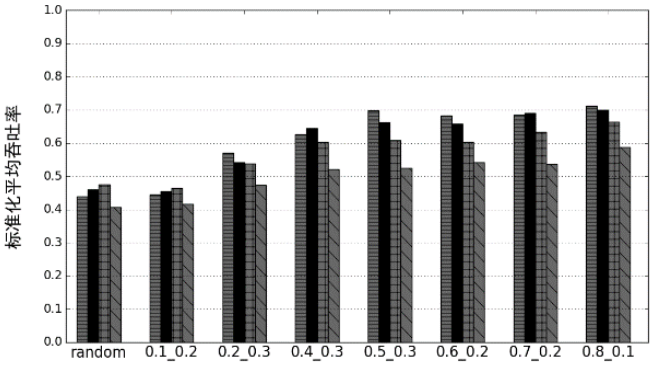

图 4　标准化平均吞吐率

Fig. 4　average normalized throughput

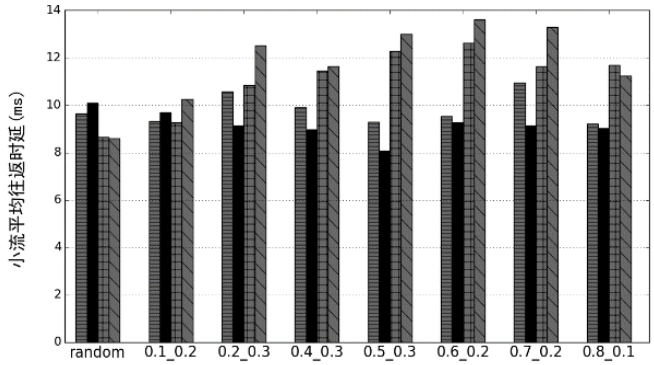

图 5　小流平均端到端延迟

Fig. 5　Mice flow average end to end delay

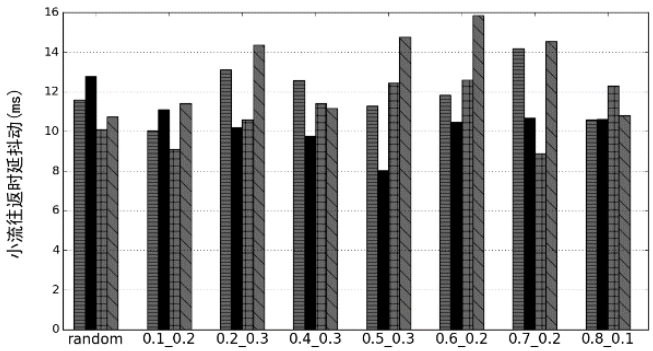

图 6　小流端到端时延抖动

Fig. 6　mice flow end to end delay jitter

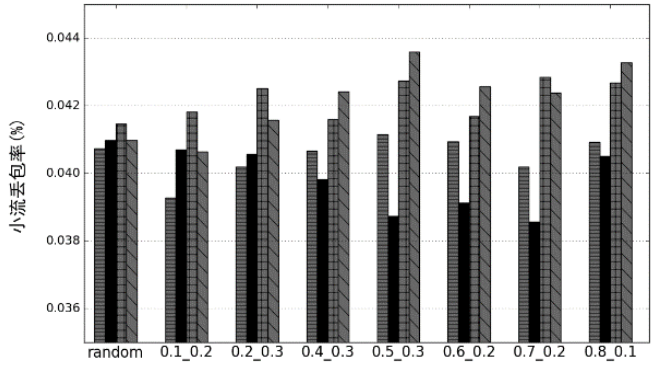

图 7　小流丢包率

Fig. 7　Mice flow packets loss

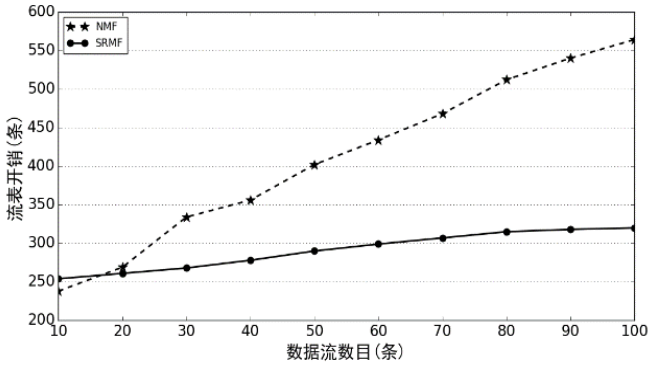

图 8　最大流表项开销

Fig. 8　maximum flow entries cost

SRMF 和 NMF 作对比，在平均吞吐率方面表现几乎一致，在时延，时延抖动，丢包方面 SRMF 表现的更好，这是由于分段转发机制中间节点不必维护大量流表，降低了负载，提高了转发效率。最后在流表开销方面，如图 9 所示，本文按照 random 通信模式发送不同数目的数据流，记录 NMF 和 SRMF 的流表开销。由于 SRMF 需要安装更多的默认流表，在数据流量较少的情况流表开销比 NMF 大，而随着数据流数目增大，SRMF 优势越来越大。这是因为 SRMF 仅仅新增源节点流表，中间节点无须更新。

以同样的方式测量 Fattree(K=8)和 COST266 的网络性能，分别以八种通信模式随机产生 16 个节点发送数据流并计算八种通信模式的平均吞吐率，拓扑参数如表 5 所示，平均标准吞吐率如图 9 所示。

表 5　三种实验拓扑参数

Tab. 5　Three experimental topological parameters

拓扑名称	节点数	链路数	主机数
Fattree(k=4)	20	32	16
Fattree(k=8)	80	256	128
COST266	28	41	21

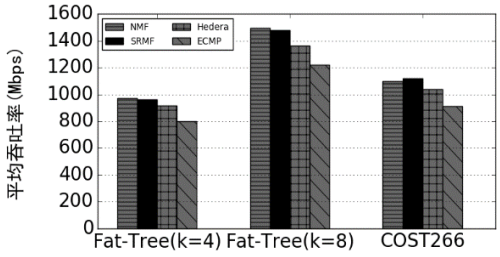

图 9　不同拓扑下平均吞吐率

Fig. 9　Average throughput under different topologies

在 Fat-Tree(K=4),Fat-Tree(k=8) 以及不规则拓扑 COST266 下进行吞吐率对比，随着拓扑规模增大，冗余链路增多，各算法的吞吐率都有一定程度的提升。

综上所述，相比采用默认转发方式的 NMF 算法，SRMF 大大节约了流表项开销的同时,综合性能有一定提升，相比 Hedera 和 ECMP，SRMF 和 NMF 综合性能更好，同时在规则拓扑如 Fat-Tree 和不规则拓扑下均有其优势。

4.5　复杂度分析

针对单条业务流路由选择的时间复杂度分析，无论是所提算法还是 Hedera 算法都需要提前计算 K 条最短路径，K 条最短路径集采用 Dijkstra 计算，倘若网络节点数目为 N，则时间复杂度为 $O(KN^2)$。

由于 ECMP 没有控制器参与，主要原理是根据数据包头的五元组信息哈希散列，所以其时间复杂度为 $O(1)$。

无论 Hedera 还是 SRMF 都需要获得各条路径的剩余带宽，所以若路径包含 L 条链路，则 Hedera 获取路径信息的时间复杂度为 $O(L)$，由于 SRMF 获取的路径信息既包括带宽，也包括丢包率和带宽波动值，经过分析这一过程的时间复杂度为 $O(L^2)$。

Hedera 算法需要对业务流重路由，一旦从最短路径集中找到一条满足带宽需求的路径立刻转发，最好时间复杂度为 $O(L)$，即第一条路径满足带宽需求，最坏的时间复杂度为 $O(KL)$，即最后一条路径满足带宽需求，故 Hedera 的时间复杂度为 $O(KL)$。

所提 SRMF 算法首先选择剩余带宽大于带宽需求的路径，假设有 K1 条，然后从 K1 条路径中找到最优路径转发，倘若 K1=0，则随机选择一条路径直接转发，所以时间复杂度为 $O(KL^2+K1)$

为了更加直观的对比算法的运行时间，从两个角度来进行测量。

a)首包往返延迟，首包往返延迟 *delay* 如式(11)所示。

$$delay = T_{compute} + T_{install} + T_{out} + T_d \tag{11}$$

其中 $T_{compute}$ 是路由计算时间，$T_{install}$ 是流表安装时间，T_{out} 是首包从控制器发出的时间，T_d 是两倍的路径时延。其中路径不发生拥塞情况下 T_d 很小且各路径差距不大，T_{out} 时间较小可以忽略，真正影响时延的是 $T_{compute}$ 和 $T_{install}$。发送单个 icmp 数据包来测量，为了测试算法性能，分别同时发送 1，16，32，64，128 个 icmp 数据包，分别对应触发相应次数的路由计算和流表安装，并取最大时延作为依据，最大时延发生在控制器负载最大的时刻。

如图 10 所示，可以看到，所提算法时间复杂度高于 Hedera 和 ECMP，由于 ECMP 没有控制器参与，所以不存在路由计算和流表安装过程，SRMF 流表安装数量小于 NMF，故 SRMF 的运行时间小于 NMF。Hedera 的 GFF 算法只需要一轮遍历找到路径并跳出，所以单从算法角度其运行时间低于 NMF。实际上，Hedera 在整个数据传输过程中需要持续重路由，实际的计算负载更高。

b)数据传输完成时间代表传输一定数据量的数据算法的运行时间。设置多组数据，每组由若干数据流组成，每条数据流字节数一定，总数据量越大，数据流数目越多，测量当数据传输完毕时的传输时间，如图 11 所示，即使所提算法时间复杂度略高，但是由于能够对数据流进行有效调度，所以传输完成时间最低，ECMP 即使无须路由计算，由于数据未经合理调度，其数据传输完成时间最长。

综上所述，即使所提算法流计算时间复杂度较高，但是由于可以更好的调度数据流，所以在总体耗时方面表现较好。

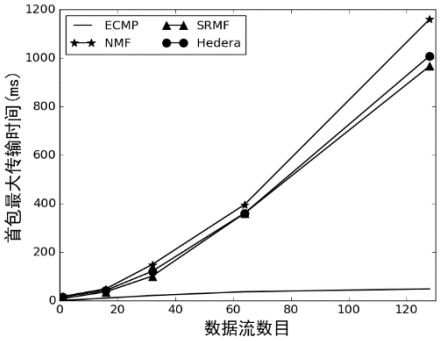

图 10　首包时延对比

Fig. 10　Comparison of first packet delay

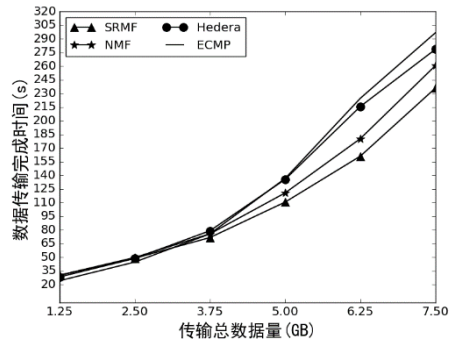

图 11　算法运行时间对比

Fig. 11　Comparison of algorithm running time

5　结束语

针对传统的路由算法容易造成大象流冲突的缺陷，以及一般的软件定义网络架构下的转发方式流表项利用率低，控制器下发大量流表造成流表更新不及时的问题。本文首先提

出一种基于本地端口的分段转发机制，然后提出一种基于分段路由的多路径调度算法—SRMF。SRMF 综合路径可用带宽，业务流带宽需求估计值，丢包率，路径可用带宽波动函数构建路径权重，在 K 条最短路径中选择最优路径转发，同时本文优化了流表的安装过程，提出一种双向流表安装机制，仿真结果表明，SRMF 在降低了平均流表项开销的同时，提高了网络性能。

在以后的工作中，将尝试在其他网络拓扑如 Bcube，Dcell 以及大型拓扑中的部署，同时继续优化算法的复杂度，进一步降低控制器负载，提高数据转发响应速度。

参考文献：

[1] Gao Y, Wei H. Profit-aware workload management for geo-distributed data centers [C]// 2017 18th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCA-T) . IEEE Computer Society, 2017.

[2] AlFares M, Loukissas A, Vahdat A. A scalable, commodity data center network architecture [J]. Acm Sigcomm Computer Communication Review, 2008, 38 (4): 63-74.

[3] Guo, Chuanxiong Lu, Guohan Li, *et al.* BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers. Computer Communication Review, 2009, 39: 63-74.

[4] Greenberg, Albert H, James J, *et al.* VL2: A scalable and flexible data center network. ACM SIG-COMM Computer Communication Review, 2011, 39: 51-62.

[5] Christian E, Hopps. Analysis of an Equal Cost Multi Path Algorithm [J]. Rfc, 2000, 1-09 (1): S265.

[6] Kandula S, Sengupta S, Greenberg A, *et al.* The nature of data center traffic: measurements & analysis [C]// Proceedings of the 9th ACM SIGCOM-M conference on Internet measurement. ACM, 2009: 202-208.

[7] Benson T, Akella A, Maltz D A. Network traffic characteristics of data centers in the wild [C]// Proceedings of the 10th ACM SIGCOMM conference on Internet measurement. ACM, 2010: 267-280.

[8] Cong L, Yong Hao W. Strategy of Data Manage Center Network Traffic Scheduling Based on SDN [C]// International Conference on Intelligent Transportation. IEEE Computer Society, 2016.

[9] Abdullah, Zahraa N, Ahmad, *et al.* Segment Routing in Software Defined Networks: A Survey [J]. Communications Surveys & Tutorials, IEEE, 2019, 21 (1): 464-486.

[10] Filsfils C, Previdi S, Ginsberg L, *et al.* Segment Routing Architecture (RFC8402) [EB/OL]. 2018. https://www. rfceditor. org/info/rfc8402.

[11] 周建新, 张志鹏, 周宁. 基于 CKSP 的分段路由负载均衡技术 [J]. 计算机科学, 2020, 47 (04): 256-261. (Zhou Jianxin, Zhang Zhipeng, Zhou Ning. Load balancing technology of segment routing based on CKSP [J]. Computer Science, 2020, 47 (04): 256-261.)

[12] 吴伟, 张文强, 杨广铭, 等. 5G 承载网的 SRv6+EVPN 技术研究与规模部署 [J]. 电信科学, 2020, 1-14. (Wu Wei, Zhang Wenqiang, Yang Guangming, et, al. SRV6+EVPN technology research and scale deployment of 5G bearer network [J]. Telecommunication Science, 2020, 1-14.)

[13] AlFares M, Radhakrishnan S, Raghavan B, *et al.* Hedera: dynamic flow scheduling for data center networks [C]// Proc of the 7th Usenix Symposium on Networked Systems Design and Implementation, 2010: 281-296.

[14] Curtis A R, Kim W, Yalagandula P. Mahout: Low overhead datacenter traffic management using end-host-based elephant detection [C]// IEEE INFOCOM. IEEE, 2011: 1629-1637.

[15] Zhang H, Tang F, Barolli L. Efficient flow detection and scheduling for SDN-based big data centers [J]. Journal of Ambient Intelligence and Humanized Computing, 2018.

[16] 兰巨龙, 于倡和, 胡宇翔, 等. 基于深度增强学习的软件定义网络路由优化机制 [J]. 电子与信息学报, 2019, 41 (11): 2669-2674. (Lan Julong, Yu Changhe, Hu Yuxiang, et, al. a SDN routing optimization mechanism based on deep reinforcement learning [J]. Journal of Electronics&Information Technology, 2019, 41 (11): 2669-2674.)

[17] 沈少辉. 基于 SDN 的分段路由技术研究 [D]. 四川: 电子科技大学, 2018. (Shen Shaohui. Research on SDN-based Segment Routing Technology [D]. Sichuan: University of Electronic Science and Technology of China, 2018.)

[18] 林智华, 高文, 吴春明, 等. 基于离散粒子群算法的数据中心网络流量调度研究 [J]. 电子学报, 2016 (44): 2197-2202. (Data Center Network Flow Scheduling Based on DPSO Algorithm [j]. Journal of Electronics&Information Technology, 2016, 44 (9): 2197－2202.

[19] 唐宏, 王欣欣, 刘亦星. 面向带宽碎片最小化和 QoS 保障的数据中心网络流量调度算法 [J]. 电子与信息学报, 2019, 41 (4): 987-994. (Tang hong, Wang Xinxin, Liu Yixing. A traffic scheduling algorithm for bandwidth fragmentation minimization and qos guarantee in data center network [J]. Journal of Electronics&Information Technology, 2019, 41 (4): 987-994.)