基于 SDN 的负载均衡网络控制器算法

窦浩铭 姜 慧 陈思光
(南京邮电大学江苏省通信与网络技术工程研究中心 南京 210003)

摘 要 当前网络新兴科技呈井喷式发展势头,这些新兴科技在为人们的生活带来极大便利与乐趣的同时,对网络处理海量数据并兼顾安全性和稳定性也提出了更高的要求。一方面,传统网络架构的处理能力很难满足该要求;另一方面,为获得更高的网络效益而开展流量调度优化的研究也大多停留在链路模块,缺少对服务器模块的关注。在此基础上,针对目前绝大多数流量调度优化算法所存在的不足,提出了额外增加了对服务器模块进行考量的路径-服务器流量调度(Path-Server Traffic Scheduling,PSTS)算法,并基于软件定义网络(Software Defined Network,SDN)范式利用 Ryu 控制器进行模块化功能实现。实现过程中,通过对链路层面和服务器层面的影响因子(性能指标)进行度量,并引入之前已获取的影响因子信息计算权重,来实现对每条链路和每个服务器的排序和筛选,为最终的最佳流量调度提供支撑。仿真结果表明,在流量负载相同的情况下,相较于目前广泛接受的动态负载均衡(Dynamic Load Balancing,DLB)算法,所提出的 PSTS 算法可以实现更高的平均带宽利用率和更低的平均传输时延;同时,在负载均衡方面,当网络中有大量数据流时,PSTS 算法可以更为有效地将数据流均衡地分配给各个服务器,极大地避免了网络中局部拥塞情况的发生,提高了数据流的处理速度,进而提升了网络的整体性能。

关键词 软件定义网络,流量调度,负载均衡,路径-服务器流量调度算法
中图法分类号 TP393 **文献标识码** A

SDN-based Network Controller Algorithm for Load Balancing

DOU Hao-ming JIANG Hui CHEN Si-guang
(Jiangsu Engineering Research Center of Communication and Network Technology, Nanjing University of Posts and Telecommunications, Nanjing 210003, China)

Abstract Currently, the emerging technologies of network show the booming development trend, which bring great convenience and fun for people's life. However, they put forward the newer and higher requirements for efficient processing ofbig data withdesired security and reliability. On the one hand, the processing ability of the traditional network is difficult to meet these performance and security requirements; on the other hand, in order to obtain higher network benefits, researches of traffic scheduling optimization almost focuse on considering the link module factor, lacking the consideration of server module. In this paper, aiming at the shortcomings of current existing traffic scheduling optimization algorithms, an optimization algorithm called PSTS (Path-Server Traffic Scheduling) which introduces the additional consideration of server module was proposed. The PSTS algorithm is based on the SDN (Software Defined Network) paradigm and finished the modular function realization by using the Ryu controller. In the implementation process, by means of measuring the impact factors (performance metrics) of link and server levels and introducing impact factors which are obtained previously, the proposed algorithm realizes the sorting and filtering operations on each link and each server by calculating the weights. Meanwhile, the sorting and filtering results providestrong support for the final optimal traffic scheduling. The simulation results show that PSTS algorithm can achieve higher average bandwidth utilization and lower average transmission delay compared with DLB (Dynamic Load Balancing) algorithm when they have the same traffic load. At the same time, the proposed algorithm can effectively distribute the data stream more balanced to each serverwhen the network has a large number of data streams, which indicates that it can avoid the local congestion of network significantly, improve the processing speed of data stream, andfinally enhance the overall performance of the network.

Keywords SDN, Traffic scheduling, Load balancing, PSTS algorithm

---

本文受国家自然科学基金项目(61771258,61772034),江苏省"六大人才高峰"高层次人才项目(XYDXXJS-044),江苏省"333 高层次人才培养工程",南京邮电大学'1311'人才计划,中国博士后科学基金(面上一等资助)(2018M630590),南京邮电大学国家自然科学基金孵化项目(NY217057,NY218058),江苏省通信与网络技术工程研究中心开放课题重点项目(JSGCZX17011)资助。

窦浩铭(1993-),男,硕士生,主要研究方向为软件定义网络与物联网,E-mail:shelddhm@163.com;**姜 慧**(1995-),女,硕士生,主要研究方向为软件定义网络与物联网;**陈思光**(1984-),男,博士,副教授,主要研究方向为物联网与信息安全,E-mail:sgchen@njupt.edu.cn(通信作者)。

随着云计算、物联网、大数据等新兴技术在当前互联网领域的不断深入及广泛应用，对网络在处理海量数据并兼顾安全性等方面的能力提出了新的更高的要求。而传统网络架构的处理能力将很难满足该要求。此时，一种新型的网络体系架构——软件定义网络（Software Defined Network，SDN）应运而生。

SDN 因其相对于传统网络架构存在的诸多优点，引发了学界众多学者的研究与关注。同时，SDN 也被越来越多地与物联网、云计算、大数据等新兴技术联系起来。文献[1]以小型化的物联网云计算环境为例，提出了基于 SDN 并利用流转向动态地调整覆盖数据路径的物联网云服务应用。文献[2]介绍了 SDN 如何在即将商用的 5G 网络中使用，以及在无线传感网即物联网中的实际部署和使用案例。文献[3]提出了 SVirt，用于为公有云的租户提供理想的虚拟 SDN 服务。当前，围绕 SDN 展开的各方面研究正如火如荼地进行着。

在网络流量管控方面，SDN 因其控制平面与转发平面相分离，且具有开放可编程接口的特点[4]，可以更好地实现对全局网络逻辑集中的管控。目前，基于 SDN 的流量管控主要有两大类研究方案，分别是从架构模型和算法进行的研究，具体表现在优化流量调度、均衡服务器负载等方面。然而，目前绝大多数流量管控算法所考虑的影响因素过于单一，仅停留在链路层面，缺少对服务器层面的考虑。如文献[5]中的动态负载均衡（Dynamic Load Balance，DLB）算法，仅通过选取可用带宽最大的链路这一单一的手段，便确定最终的转发路径。显然，DLB 算法所考虑的影响因素太过单一。

基于当前流量管控算法，特别是流量调度优化这一方面研究存在的不足，本文提出了一个路径-服务器流量调度（Path-Server Traffic Scheduling，PSTS）算法用于解决上述问题。该算法的主要贡献如下。

1）针对目前绝大多数流量调度优化算法仅仅围绕链路这一层面展开研究，而本文所提出的 PSTS 算法则额外增加了对服务器层面的考量。对影响因素近乎全面的考量使得该算法可以实现更加合理的流量调度，并获得更高的网络效益。

2）基于 Ryu 控制器进行模块化功能实现，使得本文所提出的 PSTS 能有效实现对链路层面和服务器层面影响因素的获取。实现过程中，通过对链路层面和服务器层面的影响因子（性能指标）进行度量，并引入之前已获取的影响因子信息计算权重，实现对每条链路和每个服务器的排序和筛选，为最终的最佳流量调度方案提供支撑。

3）仿真结果表明，在流量负载相同的情况下，相较于目前广泛接受的 DLB 算法，本文提出的 PSTS 算法可以实现更高的平均带宽利用率和更低的平均传输时延；在负载分配方面，当网络中有大量并行数据流时，PSTS 算法可以更为有效地将数据流均衡地分配给各个服务器，极大地避免了网络中局部拥塞情况的发生，提高了数据流的处理速度，进而提升了网络的整体性能。

本文第 1 节主要简要介绍了相关技术背景及研究现状；第 2 节描述了所提出 PSTS 算法的详细设计；第 3 节详细阐述了 PSTS 算法的具体实现；第 4 节进行了仿真实现与性能分析，验证了所提算法的性能优势；最后总结全文，再次说明了本文工作的有效性。

1　相关工作

目前，围绕 SDN 展开的对流量管控（流量调度优化）的研究已成为国际上关注的焦点，所提出的流量管控（调度）方案可大致分为如下两类。

第一类方案的思路是设计并构建新型模型或架构对流量管控进行优化操作。文献[6-8]以流量的优化分配为出发点，提出新型的流量分配与调度模型，以实现负载均衡。文献[9-10]以 OpenFlow 中的流表为出发点，改进现有的路由调度机制，以期望达到更好的负载均衡的效果。文献[11]提出了一种基于 SDN 的实时虚拟机管理框架，以时序网络信息为核心驱动虚拟机的运行，并实现对整体网络中流量的优化调度。实验结果表明，该管理结构可以有效降低通信成本，提高网络吞吐量。

第二类方案的思路是研究相应的算法对流量管控进行优化操作。文献[12]提出了一种有效的资源管理算法，主要对流量调度在内的影响负载均衡效果的因素进行优化。文献[13-14]基于 SDN 框架，就负载均衡算法的动态性及静态性进行了分析研究，实验结果证明，两种算法均可达到负载均衡的效果，但动态算法的性能要优于静态算法。

与第一类研究方案需要对整个流量管控的模型或架构进行重新设计并构建相比，第二类研究方案仅需对算法进行设计并改进，显得更简便并具更强的可操作性，但目前在算法层面对流量管控的优化仅考虑了影响数据传输的链路因素，未将影响网络数据传输的服务器负载等因素考虑在内。

2　PSTS 算法的设计

本文基于上述问题，提出了 PSTS 算法，该算法综合考虑网络的链路状况和服务器负载状况，利用加权排序思想选择最优的（路径，服务器）组合作为最终的流量调度方案。

2.1　链路模块设计

对链路模块设计的考虑包括可用带宽和链路时延这两个影响因子。

首先，假定 PSTS 算法存储了网络中每一对终端主机（$host_{src}$，$host_{dst}$）之间所有可能的路径集，记为：$Path=\{(host_{src},host_{dst})\}$。

路径 $path_i$ 由多条链路组成：$path_i=\{link_1,link_2,\cdots,link_j,\cdots,link_m\}$，每条链路 $link_j$（$1\leqslant j\leqslant m$）由两个属性决定，分别为当前可用带宽 bw 和链路时延 td。因此，有 $link_j=(bw,td)$。其中，链路可用带宽 $Path_i.bw$ 由本链路最小带宽决定，链路时延 $Path_i.td$ 由链路总时延决定。因此，有：

$$path_i.bw=\min\{link_i.bw\},j=1,2,3,\cdots m \tag{1}$$

$$path_i.td=\sum_{j=1}^{m}link_j.td \tag{2}$$

其中，可用带宽 dm 必须大于流量传输所需要的最小带宽 $dmBw$，因此规定带宽受限的路径集合，记：$lowBwPath=\{path_k\mid path_k.bw<dmBw,path_k\in path\}$。

此时，根据两主机间的总路径集 $Path$ 以及带宽受限的路径集 $lowBwPath$，便可筛选出候选的路径集 $candidatePath$，其中最大可用带宽为：

$$bw_{\max}=\max\{candidatePath.bw\} \tag{3}$$

最小时延为：

$$td_{\min}=\min\{candidatePath.td\} \tag{4}$$

为每条候选路径 $cPath_q$ 分配权重 w_q^p，计算式如下：

$$w_q^p = a \times \frac{cPath_q.bw}{bw_{\max}} + b \times \frac{td_{\min}}{cPath_q.td} \tag{5}$$

其中，a 和 b 是权重因子，且 $a+b=1$，$0<a<1$，$0<b<1$。

2.2 服务器模块设计

对服务器模块设计的考虑包括 CPU 负载和内存使用率这两个影响因子。

首先，PSTS 算法定义网络中所有的服务器集合，记：$Server=\{s_i \mid i=1,2,3,\cdots,n\}$，其中 s_i 代表各服务器。每个服务器 s_i 有两个重要属性，即 CPU 负载 cpu 和内存使用率 mem，因此服务器的总状态为：$s_i=(cpu,mem)$。一段时间 t 内服务器 s_j 的 CPU 负载 cpu 平均值为：

$$\overline{S_{j,cpu}} = \frac{\sum s_{j,cpu}}{t} \tag{6}$$

同理，一段时间 t 内，服务器 s_j 的内存使用率 mem 的平均值为：

$$\overline{S_{j,mem}} = \frac{\sum s_{j,mem}}{t} \tag{7}$$

综上，根据服务器集合 $Server$ 以及过载服务器集合 $overServer$，就可以筛选出可用服务器集，记：

$candidateServer=\{s_k \mid s_k \in Server \cap s_k \notin overServer\}$。

之后，分别计算 CPU 负载 cpu 和内存使用率 mem 相对于门限值的差异程度 $degree_{k,r}$，计算式如下：

$$degree_{k,r} = \frac{Th_{k,r} - \overline{s_{k,r}}}{Th_{k,r}} \tag{8}$$

其中，影响因子 $r \in \{cpu,mem\}$。将 r 分别取 cpu 和 mem 带入式(8)进行计算，并将所得结果中 $degree_{k,r}$ 最小的值作为当前服务器 s_k 的可利用度 $degree_k$，记：$degree_k=\min\{degree_{k,cpu},degree_{k,mem}\}$。

最后，为每个服务器 s_k 分配权重，计算式如下：

$$w_k^s = \frac{degree_k}{\sum degree_k} \tag{9}$$

2.3 最佳调度方案的选择

结合前两节已获取到的候选路径集 $candidatePath$ 以及候选服务器集 $candidateServer$，控制器就可以选择最佳的转发策略。将候选路径集和候选服务器集整合到一个总的候选集合 $candidate$ 中，记：$candidate=\{(cPathi,cServeri) \mid cPathi \in candidatePath, cServeri \in candidateServer\}$。结合候选路径集的权重 w_i^p 以及候选服务器的权重 w_i^s，目标函数为：

$$Y=\max(\alpha \times w_i^p + \beta \times w_i^s) \tag{10}$$

其中，$\alpha+\beta=1$，$0<\alpha<1$，$0<\beta<1$。

3 PSTS 算法的实现

本节主要基于上一节所设计的 PSTS 控制器算法，在 Ryu 控制器上实现对网络的实时监控，获取网络拓扑信息、网络负载信息以及服务器负载信息，实现最佳路由选择方案，从而达到提升总体网络性能的目标。

3.1 PSTS 算法架构与控制器工作流程

3.1.1 PSTS 算法架构

本文在 Ryu 控制器中通过自定义的模块，实现 PSTS 算法。模块中各组件之间的交互如图 1 所示。

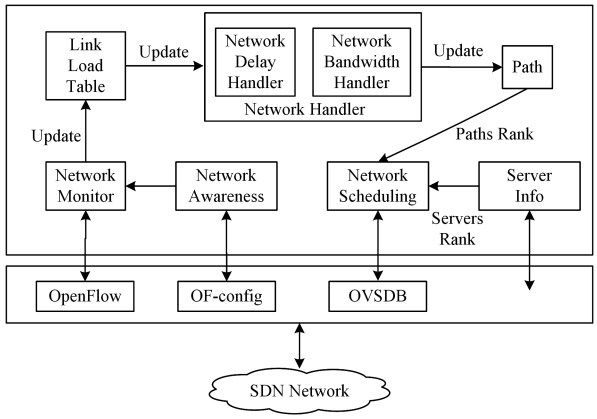

图 1 PSTS 算法的整体实现架构

自定义的模块共分 5 个组件：Network Awareness，Network Monitor，Network Handler，Server Info，Traffic Scheduling。

3.1.2 控制器工作流程

根据图 1 中各组件间的交互关系，可以设计出 Ryu 控制器的工作流程，分别是初始化阶段、状态获取阶段、计算筛选阶段以及策略执行阶段，如图 2 所示。

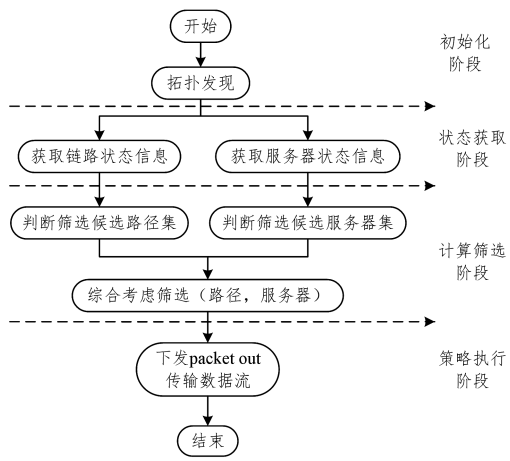

图 2 Ryu 控制器的工作流程

3.2 链路时延的获取与计算

Network Monitor 部分负责监控获取网络链路的可用带宽以及时延，并将获取到的数据存储在 networkx 的图结构中，Network Handler 组件实时地读取 networkx 图中的相关参数值，根据 PSTS 算法筛选出符合条件的路径集。

1)控制器向的交换机 A 发送 packet_out 报文，该报文的数据段携带了 LLDP 协议，并打上了发送报文的时间戳。

2)packet_out 报文的动作字段指示交换机 A 将该报文进行泛洪或者转发到其他端口。实际工作中，报文由交换机 A 转发到交换机 B。

3)交换机 B 在收到交换机 A 的报文后，因无法匹配对应流表项，所以交换机 B 向控制器发送 packet_in 报文。

4)控制器在收到 packet_in 报文后，立即解析数据包，提取出其中数据字段的时间戳，并将当下时间和该时间戳相减，得 T_1。

5)同理，可获取由控制器到交换机 B 再到交换机 A 最后返回控制器这一过程的总时间，将其记为 T_2。(T_1+T_2)等于控制器与交换机 A 之间的往返时间(Round-Trip Time,

RTT)，加上交换机 A 与交换机 B 之间的 RTT，再加上控制器与交换机 B 之间的 RTT。

6)随后，控制器向交换机 A 和交换机 B 分别发送带有时间戳的 echo_request 报文。交换机在收到该报文后，立即回复带有 echo_request 时间戳的 echo_reply 报文。记 T_a，T_b 分别为交换机 A，B 与控制器之间的 RTT。

最后，可计算出交换机 A 和交换机 B 之间链路的前后向平均时延，计算式如下：

$$T_d=\frac{(T_1+T_2)-(T_a+T_b)}{2} \tag{11}$$

上述工作中实现探测网络时延功能的类是 NetworkDelayDetector 类。

类中的主要函数有：1)get_total_delay()函数，负责将每段链路的时延进行叠加以得到每条路径的时延，并将结果保存在 paths 二维数组中；2)show_delay_statis()函数，用于显示链路时延信息(见图 3)，其中包含了源地址、目的地址、时延。

```
src   dst      delay
---------------------------
1 <--> 1 :  0
1 <--> 2 :  0.0007663965225 22
1 <--> 3 :  0.0005439519882 2
```

图 3　链路时延显示图

3.3　链路带宽的获取与计算

本文利用 OpenFlow 协议具有通过报文来获取交换机端口、流表等信息的功能，从而在 SDN 网络中实现对链路可用带宽的获取。

借助图 4 对当前链路可用带宽的获取与计算做简要介绍。

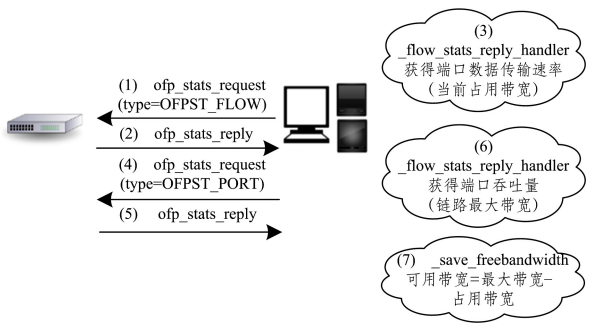

图 4　链路可用带宽的获取步骤

3.3.1　计算端口的数据传输速率

1)当控制器查询端口数据流信息时，将 ofp_stats_request 消息的 type 字段设置为 OFPST_FLOW，发送给交换机；交换机通过上传 ofp_stats_reply 消息进行回应。

2)控制器匹配 ofp_stats_reply 消息中 body 字段的 *duration_sec*，*duration_nsec*，*byte_count* 值，其中 *duration_sec*，*duration_nsec* 表示数据流的持续时间，*byte_count* 表示数据流的比特大小。由此，便可计算出数据流的实际传输速率：

$$speed=\frac{byte\_count}{duration_{sec}+duration_{nsec}/10^9}\times 8 \tag{12}$$

3.3.2　计算链路吞吐量

与 3.3.1 节中对数据流实际传输速率的获取过程类似。控制器向交换机发送 type 字段为 OFPST_PORT 的 ofp_stats_request 消息，交换机应答该消息。为了计算链路吞吐量，定义先后两个时刻 t_1，t_2，则链路吞吐量可以由下式得出：

$$capacity=[(tx\_bytes_{t_2}+rx\_bytes_{t_2})-(tx\_bytes_{t_1}+rx\_bytes_{t_1})]/(t_2-t_1)\times 8 \tag{13}$$

其中，*tx_bytes* 表示发送字节数，*rx_bytes* 表示接收字节数。

3.3.3　计算链路的可用带宽

由图 5 对链路可用带宽获取的步骤可知，当获取到当前占用带宽 *speed* 和链路吞吐量 *capacity* 后，便可计算出当前链路的可用带宽。

$$free\_bandwith=capacity-speed \tag{14}$$

得到当前链路的可用带宽后，循环执行上述操作，便可得到整个网络所有链路的可用带宽。

3.4　服务器状态获取与权衡

本文在 Server Info 部分中，调用 command 组件，对 access_port 中的每一个服务器执行各相应命令，以分别实现对内存使用率 *mem* 和 CPU 负载 *cpu* 的获取。

free 命令用于获取内存使用情况，如图 5 所示。计算内存使用率时，我们使用粗略的公式，计算式如下：

$$p_{usedmem}=used/total \tag{15}$$

```
root@crystalj:/home/crystalj# free
              total        used        free      shared  buffers     cached
Mem:       2036176     1856476      179700       17056   218576     791996
-/+ buffers/cache:      845904     1190272
Swap:      1046524           0     1046524
```

图 5　free 命令返回的结果

uptime 命令用于获取 CPU 负载信息，如图 6 所示。load average 参数后的 3 个数分别代表 1 min 内、5 min 内、15 min 内的系统负载均值，以判断系统负载的发展趋势。

```
root@crystalj:/home/crystalj# uptime
 21:05:27 up 0 min,  2 users,  load average: 0.67, 0.20, 0.07
```

图 6　uptime 命令返回的结果

为保证 CPU 性能，定义 CPU 负载门限值 $Th_{i,cpu}$ 为：

$$Th_{i,cpu}=cpu_s\times cores\times 0.7 \tag{16}$$

4　仿真与分析

利用 Mininet 仿真平台模拟 SDN 网络，并对 PSTS 算法进行仿真，在仿真基础上对算法的性能进行分析比较。由于实验条件限制以及 Mininet 仿真平台没有虚拟化服务器设备，因此，本实验只测试 PSTS 算法的链路模块。为评价 PSTS 算法与 DLB 算法的性能，定义以下两个指标：

1)平均带宽利用率 η

平均带宽利用率 η 指的是每一条数据流 i 获得的实际带宽 $real\_bw_i$ 与该条数据流的指定带宽 $assigned\_bw_i$ 的比值的平均值，计算式如下：

$$\eta=\frac{\sum_{i=1}^{n}\frac{real\_bw_i}{assigned\_bw_i}}{n} \tag{17}$$

2)平均传输时延 τ

平均传输时延 τ 指的是每一条数据流 i 到达服务器的时刻减去该条数据流从客户端发出时刻所用的平均时间，计算式如下：

$$\tau=\frac{\sum_{i=1}^{n}(server\_t_i-client\_t_i)}{n} \tag{18}$$

为验证本文提出的 PSTS 控制器算法，仿真网络拓扑如图 7 所示(因空间有限，部分交换机与主机省略)。

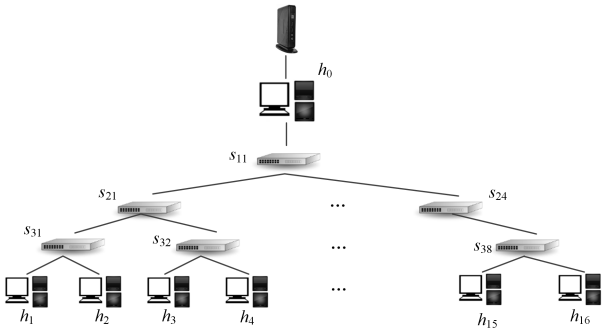

图 7 仿真网络拓扑图

本仿真首先以流量负载(Traffic Load)为自变量。将算法的评估具体化为分析不同流量负载情况下网络性能的变化。考虑图 7 的拓扑结构，本文将 h_0 设置为 Iperf 客户端，将 h_1～h_{16}作为 Iperf 服务器端，测试 10 组数据，得到的实验结果如图 8－图 11 所示。

图 8 中，随着流量负载的增加，DLB 算法和本文的 PSTS 算法的平均带宽利用率都在下降，但在 PSTS 算法中，平均带宽利用率降低得更慢一些。当流量负载达到 0.6 时，PSTS 算法比 DLB 算法高出约 37%，之后 DLB 算法的下降速率更快。

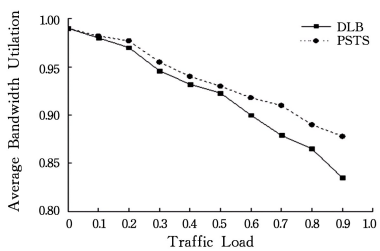

图 8 PSTS 算法和 DLB 算法的平均带宽利用率对比图

图 9 给出了不同流量负载下，DLB 算法和 PSTS 算法的平均传输时延。可以看出，PSTS 算法的性能明显优于 DLB 算法。

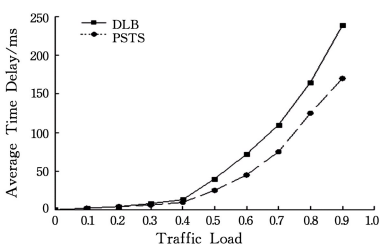

图 9 PSTS 算法和 DLB 算法的平均传输时延对比图

根据上述结果，将流量负载设置为 0.5。让客户端发送带宽为 5M 的数据流，共发送 16 次，并由控制器决定数据流传输路径和目的主机。统计各主机收到的字节数，得到各主机的负载分布图。

图 10 中，在 DLB 算法中，控制器几乎都选择同一条数据传输路径，使得 h_1 的负载剧增，数据等待队列变长，从而影响 h_1 的性能，降低数据处理速度，甚至由于局部的拥塞可能会导致丢包的产生。而在 PSTS 中，数据流则每一次到达不同的主机上，这是因为当每个主机的负载情况相同时，PSTS 算法会采用轮询的方式来选择目的主机，以缓解主机间的压力。

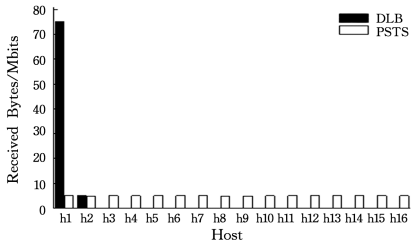

图 10 PSTS 算法和 DLB 算法的负载分布对比图

将 Iperf 客户端发送数据流的次数改为 3 000 次，记录每个主机被选择的次数，如图 11 所示。

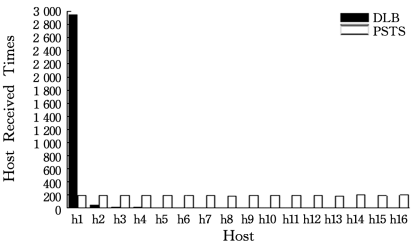

图 11 PSTS 算法和 DLB 算法目的主机选择次数对比图

图 11 表明，在网络通信带宽良好的情况下，DLB 算法选择的目的主机几乎不会改变。该实验进一步证明了本文研究的算法的合理性。

由以上 4 个统计对比图可以看出，本文提出的 PSTS 算法的性能明显优于文献[9]提出的 DLB 算法。

结束语 针对目前广泛接受的 DLB 算法，仅将每一跳中带宽最大的链路作为最终选定路径的这一弊端，本文认为流量调度应综合考虑全局网络状况，一方面，在一跳中某段链路的带宽最大，并不能代表该链路的其他条件也优于这一跳中的其余链路；另一方面，影响流量调度的因素除网络链路状况外，还包括服务器负载状况等。基于上述考量，本文提出了一种基于 SDN 的负载均衡网络控制器算法，该算法综合考虑了网络链路状态和服务器负载状态。仿真结果表明，本文设计的 PSTS 算法可以获得更大的平均带宽利用率、更小的平均传输时延、更均衡的网络负载，有效避免了网络局部拥塞情况的出现，表现出了良好的网络性能。

参 考 文 献

[1] HEEBUM Y, SEUNGRYONG K, TAEKHO N, et al. Dynamic flow steering IoTmonitoring data in SDN-coordinated IoT-cloud services[C]// Proceedings of International Conference on Information Networking. New York: IEEE Press, 2017: 625-627.

[2] BIZANIS N, KUIPERS F A. SDN and virtualization solutions for the Internet of Things: asurvey [J]. IEEE Access, 2016, 4(99): 5591-5606.

[3] YU Y, LI D, HUANG Y. SVirt: A substrate-agnostic SDN virtualization architecture for multi-tenant cloud[C]// Proceedings of the IEEE International Conference on Network Protocols. New York: IEEE Press, 2015: 313-322.

[4] MCKEOWN N, ANDERSON T, BALAKRISHNAN H, et al. OpenFlow: enabling innovation in campus networks[J]. ACM SIGCOMM Computer Communication Review, 2016, 38: 69-74.

(下转第 331 页)