一种基于 QoS 和动态负载均衡的路由策略

孙 杰,李 莉,沈苏彬
(南京邮电大学 计算机学院、软件学院,江苏 南京 210003)

摘 要:为了提高 SDN 网络负载均衡调度的针对性和准确性,设计了 SAS(Scheduling According to Stickiness)算法。该算法提出了链路粘值的概念,用链路粘值预估调度对流性能影响的大小,通过优先调度粘值低的流来减小调度对流性能的影响,达到优化调度的目的。在重路由过程中,该算法兼顾了流的 QoS 需求,并用近似算法进行多 QoS 约束的最优路径选择。基于 Floodlight 开源控制平台设计和实现了相应的原型系统,并根据负载均衡效果和负载均衡生成流的 QoS 两个测试目标,设计和实现了测试的方案。实验测试结果表明,在不同 QoS 需求的数据流竞争网络资源、导致网络负载偏离均衡状态的情况下,相较 DLB、LABERIO 机制,提出的技术方案在提高带宽利用率的同时可以兼顾分组流的 QoS 需求,并且可以降低对流性能的影响。

关键词:软件定义网络;路由策略;动态负载均衡;链路粘值
中图分类号:TP392 **文献标识码**:A **文章编号**:1673-629X(2016)11-0188-07
doi:10.3969/j.issn.1673-629X.2016.11.041

A Routing Strategy Based on QoS and Dynamic Load Balancing

SUN Jie, LI Li, SHEN Su-bin
(School of Computer Science & Technology, School of Software, Nanjing University of Posts and Telecommunications, Nanjing 210003, China)

Abstract: In order to improve pertinence and accuracy of network load balancing scheduling for SDN (Software-Defined Networking), a SAS algorithm is designed. This algorithm puts forward the concept of link stickiness, and applies link stickiness to estimate the impacts of scheduling on flow performance, by preferentially scheduling flow with low stickiness to reduce the impacts of scheduling on flow performance, so as to achieve the purpose of optimized scheduling. In the process of re-routing, it takes QoS needs of flow into consideration, and applies approximation algorithm to calculate the optimal route under multi-QoS constraints. Then a prototype system is designed and implemented based on Floodlight, an open control platform. Aiming at two test targets, including the load_balancing effect and its generated flows' QoS, the test approach is designed and implemented. Experimental test results show that in case of different QoS requirement flows competing for limited network resources and resulting in network load deviations from the balance state, compared with DLB and LABERIO, the technology approach proposed can improve the bandwidth utilization while fulfilling the requirements of QoS on flow, and reduce the influence on flow's performance.

Key words: Software-Defined Networking; routing strategy; dynamic load balancing; link stickness

0 引 言

随着大量的网络设备和服务的出现,SDN(Software-Defined Networking)[1] 网络也慢慢成为一个新的研究热点。与传统的分布式 IP 网络的不同之处在于它对控制平面与转发平面的分离。SDN 网络是一种不需要控制逻辑和数据转发功能紧耦合在网络设备上,控制器利用控制-转发接口[2] 对数据平面的交换机进行集中式控制的新型网络体系结构。SDN 交换机以提取并解析分组头域,逐级与流表项进行匹配,按照其中指定的动作进行处理和转发的通信模式,可以灵活地实现节点的数据传输。新型网络业务被广泛应用,网络业务类型不断增加,网络拥堵越来越严重。实现负载均衡可以优化利用网络资源,提高网络资源的使用率。研究 SDN 网络负载均衡解决方案具有很大

收稿日期:2016-01-18 **修回日期**:2016-05-10 **网络出版时间**:2016-10-24
基金项目:江苏省未来网络前瞻性研究资助项目(BY2013095-1-08)
作者简介:孙 杰(1990-),男,硕士研究生,研究方向为计算机应用技术;沈苏彬,博士生导师,研究方向为计算网络、下一代电信网及网络安全。

的实用价值。

已有一些经典的 SDN 网络负载均衡解决方案。比如,文献[3]中提出的负载均衡模型,考虑到网络拥塞和服务器负载,使用平均响应时间和网络最新状态,为网络应用的请求合理分配。文献[4]中提出了一种支持多服务的 OpenFlow 负载均衡,对于不同的服务有不同的控制器用于处理,不同的控制器的负载均衡算法选择可以是不同的。文献[5]中提出了基于通配符匹配的负载均衡算法,以发起请求的客户端 IP 地址作为通配符前缀,接着用规则集来控制交换机,从而减少控制器进行干预的次数,降低了平均请求延迟。

文中认为不同类型的流其 QoS 受调度的影响程度不尽相同。网络中的数据流有时延敏感的小数据流和吞吐量敏感的大数据流[6]。在网络负载不均衡程度超过一定值时,在同样都能起到均衡负载的情况下,调度前一种对 QoS 的影响要远大于后一种,所以应该选择后者调度。文中提出的调度策略不仅考虑到传统流的分类,而且考虑到调度可能对流的 QoS 性能指标产生的影响大小,因此更有针对性。这里将所引入的流的适合调度值借用了一个物理学中的概念-链路粘值(stickness)来表述,并使用改进的动态负载均衡算法更新链路粘值。

算法通过北向接口完成对流链路粘度的初始区分,这个过程其实是获取业务对流链路粘度的需求过程。在改进的动态负载均衡算法更新下,多次被调度的流的链路粘值会不断增大,对调度的拒绝程度越来越高。这时,即使是初始的链路粘值很高的流,也会被合理地调度到。在动态重路由的过程中,算法兼顾了流的 QoS 需求,为流寻找一条有足够资源、能满足 QoS 要求的可行路径。对于这种具有 NP 难度的多目标选路问题,文中选择了一种现有的带多约束的最优路径算法-Iwata[7],作为一个具体的计算方案。带着这样的目的,理论上便可以在满足流 QoS 需求的同时,保证动态负载均衡调度的针对性和准确性。

1　相关技术分析

相较于传统网络中的负载均衡策略,在 SDN 网络中主流的支持负载均衡机制都基于 SDN 控制器的全局视野。SDN 控制器对网络状态进行检测,利用根据链路负载的实时状态对影响负载均衡的最大流进行调度的负载均衡策略。该策略的代表算法为 LABERIO[8]。SDN 控制器查询链路负载的状态作为动态负载均衡的选路依据,每次都使用贪心策略选取当前空闲带宽最大的链路。该策略的代表算法为 DLB[9]。

在算法 LABERIO 中,考虑到仅仅依靠 SDN 网络的初始路由计算,即使在刚开始满足负载均衡的需求,也会随着网络的变化变得不再满足,并不能保证传输数据过程中的网络动态负载均衡。为此,需要对 SDN 网络的状态进行周期更新,实时监控网络负载均衡程度,对负载过重的链路上的流量进行调度,以使网络实现动态负载均衡。对负载过重的链路上的流量进行调度时,优先对网络中负载均衡影响最大的流进行调度。但是这种算法忽略了流对 QoS 的要求,可能会使有些流因为调度而导致时延过大。算法对那些时延敏感的流并不合适。

在算法 DLB 中,将流作为更小的单元,采用了深度优先的算法,开始从源节点向上传输到高层次的节点直到第一层节点,然后继续往下转发目的节点。在每一跳 DLB 算法用贪心策略来选择更大的空闲带宽的链路。DLB 算法在 SDN 网络有着比传统负载均衡算法更好的性能。但贪心路由决策没考虑整个网络的负载分布,由算法选出的源节点到目的节点的路径可能并不合适。

上述介绍的机制都是基于 SDN 网络的动态重路由机制[8]并已验证其有效性。同样是通过动态的重路由机制解决 SDN 网络中的负载均衡问题,文中采用的算法更多关注了调度对流的性能所造成的影响。显而易见,参照测算出的重路由对流产生的影响信息,尽量选择影响小的流进行重路由,比仅在初始设定流的优先级更有针对性和准确性。SAS 算法中用链路粘值作为流拒绝重路由程度的量化的值。动态地测算流的链路粘值,并按照链路粘值来对流进行调度,能较好地解决 SDN 网络中的动态负载均衡问题。

2　问题分析与方案设计

2.1　动态负载均衡的路由方案

在介绍基于 QoS 和动态负载均衡的路由方案之前,这里先定义一个负载均衡参数 $\varepsilon(t)$,用来描述在 t 时刻全网负载均衡的程度。$\varepsilon(t)$ 值越趋近于 0 表示全网链路的负载越均衡,$\varepsilon(t)$ 值越大表示全网的负载越不均衡。$\varepsilon(t)$ 值的计算如式(1)所示。

$$\varepsilon(t)=\frac{\sum\left(\mathrm{load}_{ij}(t)-\overline{\mathrm{load}(t)}\right)^2}{n} \tag{1}$$

其中,$\overline{\mathrm{load}(t)}$ 表示网络链路在 t 时刻的负载平均值;$\mathrm{load}_{ij}(t)$ 表示在 t 时刻链路 l:$[i_l,j_l,\mathrm{CAP}_l]$ 的负载值;n 表示网络中所有链路总数。

这里使用了数学中的方差公式来研究链路负载与全网平均负载的偏离程度即负载均衡程度。其中 $\mathrm{load}_{ij}(t)$ 为当前链路占用带宽与链路 l 的最大带宽能力 CAP_l 的比值。$\mathrm{load}_{ij}(t)$ 值的获取通过控制器读取链路 l:$[i_l,j_l,\mathrm{CAP}_l]$ 流统计信息计算得到,其值如式(2)

所示。$\overline{\text{load}(t)}$ 值的计算通过对所有链路(l:[i_l,j_l,CAP_l],$l \in L$)的负载值 $\text{load}_{ij}(t)$ 求和算均值来获得,其值如式(3)所示。

$$\text{load}_{ij}(t) = \frac{(B_t - B_{t-p})/p}{\text{CAP}_l} \tag{2}$$

$$\overline{\text{load}(t)} = \frac{\sum \text{load}_{ij}(t)}{n} \tag{3}$$

其中, B_t 表示在 t 时刻链路 l:[i_l,j_l,CAP_l] 上已传输的总 byte。每条链路上的 B_t 值都可以通过 SDN 数据统计机制来获取。

这里需要设置一个负载均衡阈值 $\varepsilon*$ 。$\varepsilon*$ 值的设置依据是不同 $\varepsilon*$ 值会导致不同网络吞吐量,研究吞吐量与 $\varepsilon*$ 的变化关系并选取使得网络吞吐量最大的 ε 值作为 $\varepsilon*$ 。当 $\varepsilon(t)$ 在 t 时刻大于 $\varepsilon*$ (即网络负载不均衡程度超出预设值时),触发联合路由方案启动负载均衡调度。控制器将找到占据在最拥堵链路上且链路粘值最小的流,将其移动到其他可用的路径上。这里的可用路径需要满足两个条件:路径是包含首尾节点的通路;路径上的所有链路可以满足流 f 所属类型 c 的 QoS 需求(即满足式(5))。在计算出所有可用路径的集合后,选择其中负载最小的路径作为流 f 的替换路径。如果没有可替换的路径表明流 f 对当前链路的依赖程度(即粘值)高出了预期,则为流 f 增加一个定值 s 作为新的 s_f 再重新执行启动负载均衡的调度。基于 QoS 和动态负载均衡的路由方案步骤的具体描述如下:

步骤 1:控制器不断检测网络的负载均衡状态,每隔 n s 计算一次网络的负载均衡参数 $\varepsilon(t)$ 值。如果 $\varepsilon(t)$ 大于阈值 $\varepsilon*$,执行步骤 2,否则继续执行步骤 1。

步骤 2:找出负载最大的链路,即 $\text{load}_{ij}(t)$ 值最大的链路 l:[i_l,j_l,CAP_l] 。执行步骤 3。

步骤 3:找出链路 l:[i_l,j_l,CAP_l] 承载的所有流中链路粘值 s_f 最小的流 f 。执行步骤 4。

步骤 4:找出所有流 f 的可用替代路径 p_f 集合。路径 p_f 的首节点为 n_i 、尾节点为 n_j (即为 $n_i \rightarrow n_m \rightarrow n_j$ 序列),p_f 满足式(2)。如果 p_f 集合不为空则执行步骤 5,否则执行步骤 7。

步骤 5:选择 p_f 集合中负载最小的路径 $\min(p_f)$ (如果有多个负载最小的路径,则选择其中剩余带宽最大的路径)作为流 f 的替代路径。执行步骤 6。

步骤 6:删除链路 l:[i_l,j_l,CAP_l] 上 f 的表项。安装 $\min(p_f)$ 上 f 的流表项,建立并关联相应计量表项。执行步骤 7。

步骤 7:更新流 f 的链路粘值 s_f 。如果流 f 迁移成功,执行 2.3 中的流 f 链路粘值更新策略,未成功则 s_f 加上一个常数 s 作为新的 s_f 。继续执行步骤 1。至此,联合路由选择的一个周期结束。

基于 QoS 和动态负载均衡的路由算法依据网络当前的负载均衡状态对过载链路进行动态的流调度,通过数据流的链路粘值来描述数据流对流调度的拒绝程度,进而得到应当对哪条流进行调度。因此算法的时间复杂度直接取决于链路个数 n_l 和数据流的个数 n_f ,为 $O(n_l * n_f)$ 。

2.2 基于 QoS 约束的选路策略

在 SDN 网络这种基于流的网络架构中,可以区分并保证每一个业务流的服务质量,完成最细粒度的服务质量保证。而 SDN 网络拥有的网络全局视野更能够满足业务流对链路的需求。边界交换机将数据流分类后,要为数据流分配能够满足数据流相应需求的路径。数据流 f 可以用源节点 s_f 、目的节点 d_f 、服务类别 c_f 这三个属性来表示(f:[s_f , d_f , c_f], $s_f \in N$, $d_f \in N$, $c_f \in C$)。其中, C 代表所有服务类别的集合。服务类别包含的参数包括数据速率 r_c 、丢包率 p_c 、时延 t_c 和用来为动态负载路由算法服务的初始链路粘度 s_c(c:[r_c,p_c,t_c,s_c]) 。这里借用了粘值(Stickiness)的物理学概念。物理学中的粘值[10]是指流体的内摩擦表现出的宏观属性,这里的粘值表示服务类别 c 的数据流 f 对当前的链路的依赖性或者称为数据流对重路由的排斥程度。这个变量的定义将在 2.3 小节加以具体描述。

为了方便描述 SDN 网络中的动态路由的选路策略,将 SDN 网络用有向图 $G(N,L)$ 表示。其中, N 表示网络交换节点的集合, L 表示网络中的链路集合。用 l 表示其中一条链路(l:[a_l,b_l,CAP_l],$l \in L$),其中, a_l 为链路首节点, b_l 为链路尾节点($a_l \in N$, $b_l \in N$)。用 p_f 表示数据流 f 流经的路径($p_f = \sum p_f^l$, $p_f \in p$),其中 p_f^l 表示数据流 f 流经的链路 l , p 表示网络中所有存在的路径。每个流 f 在 p_f^l 都关联了一个二元矩阵 m_f^l[m_r , l],这个二元矩阵表示的数据流 f 在链路 l 上是否关联了计量速率为 r 的计量表 m_r ,其值如式(4)所示。计量表是在 OpenFlow1.3 协议[11]中提出的一种表项,用来在流表关联计量表项时设置转发速率。文中用计量表来代替对交换机队列的直接使用。

$$m_f^l[m_r,l] = \begin{cases} 1, \text{流 } f \text{ 在 } p_f^l \text{ 上关联了 } m_r \text{ 的计量表} \\ 0, \text{其他} \end{cases} \tag{4}$$

路由选择需要考虑链路能否满足数据流所属类型的 QoS 值。当网络中增加一个新流 f (流所属类别的属性为 r_{new} ,p_{new} ,t_{new} 和 s_{new})。流 f 流经的所有链路 l 都需要满足式(5) ~ (7)。

$$\text{CAP}_l \geqslant \sum_{f \in f_l} [m_r,l] * r_f + r_{\text{new}} \tag{5}$$

$$p_{\text{new}} \geqslant \frac{p_{f_{\text{src}}} - p_{f_{\text{dst}}}}{p_{f_{\text{src}}}} \tag{6}$$

$$t_{\text{new}} \geqslant \sum_{l \in p_f} t_l - 2 * \sum_{n_i \in n_f} d_{c\_n_i} \tag{7}$$

其中,f_l 表示在链路 l 上承载所有的流集合;n_f 表示流 f 流经的所有的交换机集合;CAP_l 表示链路 l 的最大容量;$p_{f_{\text{src}}}$,$p_{f_{\text{dst}}}$ 分别表示在流 f 转发路径的起始交换机 f_{src} 和目的交换机 f_{dst} 上已接收到的流 f 包的总数。

式(5)的前提是:链路的所有流必须跟某个 meter 表项关联。对网络上大量的 non-qos 流,算法为它们关联一个基本 QoS 的 meter 表项。式(5)的参数 f_l 以及 $m_f^l[m_r, l]$,可以在控制器通过 OpenFlow 协议读取交换机的表项信息来获取。式(6)的参数 $p_{f_{\text{src}}}$ 和 $p_{f_{\text{dst}}}$,可以通过 SDN 统计机制获取。而流路径时延需要通过测算获得,流路径时延的测算方法如式(7)中大于号右边的式子所示,通过控制器对链路时延 t_l 以及控制器与交换节点间总时延 $d_{c\_n_i}$ 的测算得出。

常规的网络时延包含处理时延、排队时延、传输时延、传播时延。这里的 t_l 不仅仅包含链路时延,文献[12]将传输时延和传播时延合起来作为链路时延(例如 ns2 仿真器)。这里的 t_l 不仅包含传输时延和传播时延,还包含了跟控制器的交互。因此,在式(7)中大于号右边减去了节点 n_i 和控制器交互的时延 $d_{c\_n_i}$。

为了方便计算 t_l,文中提出了用于 SDN 交换机的服务质量监测机制,即一种基于 OpenFlow 协议实现的链路时延测算机制。在 LLDP(Link Layer Discovery Protocol[13],链路层发现协议)分组基础上进行改进,采用链路时延监测分组(Link Delay Discovery Packet,LDDP)来监测链路 l 从链路源节点 $n_{l_{\text{src}}}$($n_{l_{\text{src}}} \in N$)接受到分组到链路 l 目的节点 $n_{l_{\text{dst}}}$($n_{l_{\text{dst}}} \in N$)接受到分组的时延。LLDP 是 IEEE 定义的一种链路层协议,被大部分网络设备所支持,一些 OpenFlow 控制器利用该协议实现网络拓扑的发现功能。

而 LLDP 分组通过控制器发出分组到链路 l 的端节点。LLDP 分组内容包括链路 l 的源节点 mac 地址 $\text{mac}_{l_{\text{src}}}$、链路 l 的源节点端口 $\text{port}_{l_{\text{src}}}$、控制器在 $n_{l_{\text{src}}}$ 处理此 LLDP 分组时的时刻 $t_{l_{\text{src}}}$(LLDP:[$\text{mac}_{l_{\text{src}}}$, $\text{port}_{l_{\text{src}}}$, $t_{l_{\text{src}}}$])。LLDP 分组发出后其中的动作字段为转发到节点的指定端口。LLDP 分组从节点 $n_{l_{\text{src}}}$ 的端口转发出去,对端节点 $n_{l_{\text{dst}}}$ 将会收到传入的 LLDP 分组,但它并不能识别这个 LLDP 分组,只当普通数据报文处理,此 LLDP 分组将被交换机封装成 Packet-In 消息发送到控制器中。控制器将 LLDP 分组中的 $\text{mac}_{l_{\text{src}}}$ 和 $t_{l_{\text{src}}}$ 消息记录下来,根据 $\text{mac}_{l_{\text{src}}}$ 和当前节点 mac 标记出链路 l,根据当前系统时刻 $t_{l_{\text{dst}}}$、l 源节点处理时刻 $t_{l_{\text{src}}}$ 即可求出链路 l 的时延 t_l 值如式(8)所示。

$$t_l = t_{l_{\text{dst}}} - t_{l_{\text{src}}} \tag{8}$$

文中对控制器 c 与节点 n_i 间的时延 $d_{c\_n_i}$(包括控制器处理发送 OpenFlow 请求到控制器接收 OpenFlow 响应之间的时延)的计算,这里的 OpenFlow 请求和响应,文中使用的是特征请求消息和特性响应消息。在控制器向将节点 n_i 发送特征请求消息时记录下当前时刻 $t_{c\_n_i}$,当控制器接收到相应节点的特性响应消息时记录下时刻 $t_{n_i\_c}$。如此,$d_{c\_n_i}$ 的值通过式(9)得出。将式(8)和式(9)带入式(7),即可完成对路径是否满足流时延要求的判断。

$$d_{c\_n_i} = \frac{t_{n_i\_c} - t_{c\_n_i}}{2} \tag{9}$$

因此,在进行重路由之前需要先选取全网所有满足式(5)~(7)的链路。选出的链路信息用来为动态负载均衡重路由选取提供拓扑结构。

当新的流需要进行重路由时,它需要的路径不仅要满足 2.1 节的替换路径的要求,还需要满足式(5)~(7)对链路的要求。这种多目标的选路策略是有 NP 难度的,因此文中采用了一种现有的带多约束条件的 QoS 路由算法-Iwata[7],作为一种具体的计算方案。Iwata 算法先以一个 QoS 度量为关键字来计算最短路径,然后检测该路径是否满足其他 QoS 度量;如不满足,则选取另一 QoS 度量来计算最短路径,同样检测该路径是否满足剩余的 QoS 度量;如此反复直到找到一条满足所有 QoS 度量的可行路径。

2.3　分组流的链路粘值更新策略

为了使得网络达到动态负载均衡状态,需要解决两个问题:网络链路过载发生的检测和检测到过载发生后的处理。这两个问题已在 2.1 小节解答了。一旦链路发生过载,应对链路上的哪个流重路由,使网络状态恢复负载均衡。需要对不同的流进行区分,不同的流可以简单分为对吞吐量敏感的流和对延时敏感的流。考虑到延时敏感流传输时间相对较短,改变其路径很可能会增加延迟和开销,因此应迁移非延时敏感流[14]。利用概念粘值,链路粘值 s_f 表示数据流 f 对迁移的拒绝程度。而前面提到的不同的流(吞吐量敏感的流和对延时敏感的流)对更改路径的成本高低,都可以简单地用对迁移的拒绝程度高或低即链路粘值 s_f 来表示。计算链路粘值如式(10)所示。

$$s_f = \begin{cases} s_{fl} + u_{fl}, & \text{流 } f \text{ 至少迁移过一次} \\ s_c, & \text{流 } f \text{ 未迁移过} \end{cases} \tag{10}$$

其中,s_{fl} 表示 f 最近一次 s_f 值;u_{fl} 表示最近一次迁移对数据流的影响值。

通过式(10)可以看出,如果数据流 f 从未迁移过,它会有一个初始值 s_c。这个值路由模块执行时为数据流区分的服务类型 c 所包含的属性 s_c。这个 s_c 其实

就是控制器根据分类机制为数据流 f 设置的初始粘值。考虑到网络的动态性,那些被迁移的流的链路粘值只有根据迁移对流的影响值来动态更新,才能保证 s_f 的准确性。并且式中 s_f 的计算过程实际为一个迭代过程,每一次迁移的影响值都会被考虑在内。这样就保证了一些初始粘值较小或者每次的迁移影响值较小的流,不会被频繁调度,保证了流调度的合理性。式(10)中影响值 u_{fl} 可以通过对 SDN 数据统计信息的计算来完成。u_{fl} 的值如式(11)所示。

$$u_{fl} = \alpha * d_{fl} + \beta * b_{fl} \tag{11}$$

其中,d_{fl} 表示最近一次迁移流 f 丢包率的变化程度;b_{fl} 表示最近一次迁移流 f 传输带宽的变化程度;α 和 β 分别表示 d_{fl}、b_{fl} 在影响值 u_{fl} 中所占的比例。

d_{fl} , b_{fl} 分别通过式(12)和式(13)计算获得。

$$d_{fl} = \frac{(p_{t_1} - p_{t_1-n})/n}{(p_{t_2} - p_{t_2-m})/m} \tag{12}$$

$$b_{fl} = \frac{(b_{t_1} - b_{t_1-n})/n}{(b_{t_2} - b_{t_2-m})/m} \tag{13}$$

其中,p_{t_1} 表示 t_1 时刻流 f 的总丢包数;b_{t_1} 表示 t_1 时刻流 f 的已传输的总 byte。

控制器在流迁移前隔 m s 读取一次链路的统计信息,在流迁移后隔 n s 读取一次链路的统计信息。两次统计信息的比值准确反映了链路迁移对流 f 的丢包率和带宽影响。

控制器记录并更新每条流的最新的链路粘值。当 SDN 网络发生过载时,控制器对比过载链路上所承载的流的链路粘值并决定对哪条流进行流调度。关于根据链路粘值进行流调度以及调度后的路由策略已在 2.1 小节说明了。

由上可知,SAS 是一种基于 QoS 和动态负载均衡的路由策略,所以在实验部分需要使用不同类别流传输对动态负载均衡的实现进行功能测试,使用不同的传输速率对使用 SAS 机制后的带宽、丢包率、抖动变化进行性能分析。

3 实验搭建及性能分析

实验对 SAS 算法系统软件进行测试,测试环境如图 1 所示。在源末主机上部署 iperf 发包工具,模拟端到端的不同业务流的传输环境。在两种业务流传输情况下分别执行 LABERIO、DLB 以及文中的 SAS 系统,统计各机制运行过程中带宽、丢包率、抖动等性能指标来进行比较。

3.1 实验环境搭建

搭建 SDN 实验网络。首先在一台 DELL 服务器上部署最新版本 floodlight 控制器,作为控制层设备。其次,由于目前 OpenFlow 物理交换机价格昂贵,文中选用一款开源的 OpenFlow 软件交换机 OVS 安装在三台 DELL 台式机上作为数据层转发设备。交换机和控制器的主机系统均为 32 位 Ubuntu 12.04,而通信的主机节点均为 Windows7 系统。

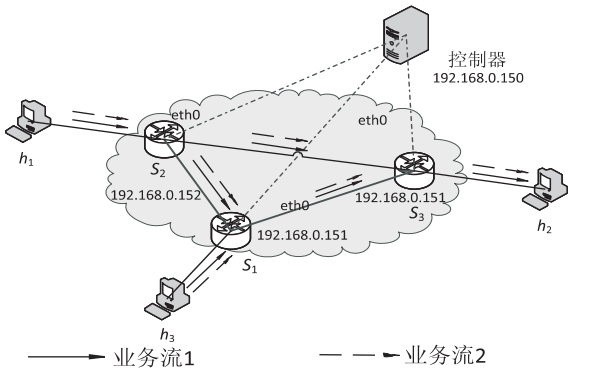

图 1 测试网络拓扑图

实验中,三台 OVS 软件换机分别与控制器相连受控制器控制。文中将控制器 IP 地址设置为 192.168.0.150,交换器控制端口 IP 地址分别设置为 192.168.0.151,192.168.0.152 和 192.168.0.153。将通信主机与三台交换机中的任意一台相连,通信主机分别运行数据传输的客户端和服务器端(文中是用 iperf 发包工具)用于发送和请求数据分组。设置负载均衡阈值 $\varepsilon*$ 为 1%,粘值增量常数 s 为 1。

3.2 实验性能评估指标

文中使用平均带宽利用率、平均丢包率和平均抖动等性能指标对负载均衡算法的性能进行评估。

(1)平均带宽利用率。

指定带宽 ar_f 是指由客户机发送流的带宽,而实际带宽 rr_f 是指服务器实际获取的流带宽。由于传输期间可能发生网络拥塞将导致分组丢失,从而服务器接收的实际带宽小于或等于由客户机发送的指定带宽。平均带宽利用率 α 是指每条流实际获得的带宽与其所指定带宽的比值的平均值。

$$\alpha = \frac{\sum_{f \in F} \mathrm{rr}_f / \mathrm{ar}_f}{n} \tag{14}$$

平均带宽利用率反映了网络的均衡程度,利用率越高说明网络的负载越均衡。

(2)平均丢包率百分比。

由 SDN 统计机制获得每条流的丢包信息,丢包数与传输指定的数目的比值就是这条流的丢包率。平均丢包率百分比 β 是指所有流的丢包率的平均值,如式(7)所示。平均丢包率百分比越大说明网络拥塞程度越高,流的 QoS 传输效果越差。

$$\beta = \frac{\sum_{f \in F} p_f}{n} \tag{15}$$

(3)平均抖动。

根据 iperf 客户端工具输出的测试报告,得到每个流的抖动 j_f。平均丢包率百分比 γ 是指所有流的抖动的平均值。平均抖动越大则说明流的 QoS 传输效果越差。

$$\gamma = \frac{\sum_{f \in F} j_f}{n} \tag{16}$$

3.3　实验结果分析

图 2 是在不同业务流传输时,三项机制的平均带宽利用率比较。

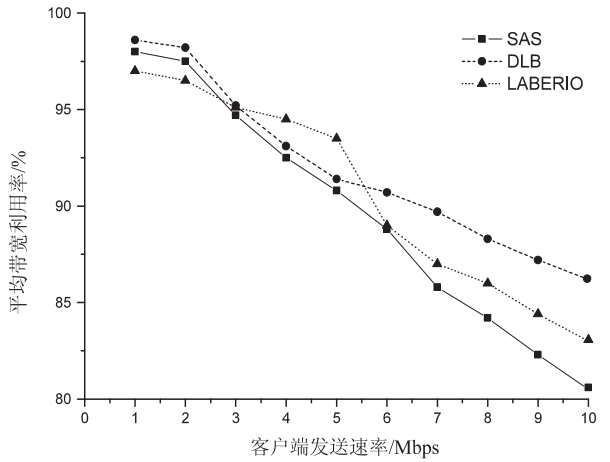

图 2　多业务流时不同机制对平均带宽利用率的影响

从图中可以看到,随着流量负荷增加,三种算法的平均带宽利用率在逐步下降。SAS 算法在平均带宽利用率上表现一般。甚至在高流量负荷时,比 LABERIO 表现要差。尤其在客户端发送速率增加至 6 Mbps 后,利用率平均低了 3% 左右。因为在重负荷之下,LABERIO 算法这种优先对网络中占用带宽最多的流进行调度的机制,虽然忽略了流对 QoS 的要求,但负载均衡效果更好。在重负荷之下 DBL 算法的单跳最优策略更容易引发网络的局部拥塞,增大了报文的丢包率,降低了数据流的带宽利用率,而 SAS 算法在考虑全局路径负载情况后做出的选路降低了发生拥塞的概率,相比于 DBL 有更高的带宽利用率。

随着数据传输速率的增加,网络动态的调整,重新寻找满足流 QoS 要求的可行路径。相较于 DLB、LABERIO 机制,SAS 算法使用了测算流 QoS 变化的负载均衡改进策略,优势在于提高了流的负载均衡调度的针对性和准确性。

SAS 通过改进的动态负载均衡算法,按链路粘值为流进行调度。链路粘值考虑到了丢包率的影响,优先调度对丢包率影响较小的流。由于 SAS 是多目标的计算综合最优结果的算法,SAS 相比 LABERIO 这种单纯考虑负载均衡效果的算法,SAS 会在负载均衡调度时考虑到调度对流丢包率的影响,优先调度丢包率影响较小的流而不是对负载均衡影响最大的流,因此会产生更低的网络丢包率。作为代价,SAS 算法在平均带宽利用率上表现一般,甚至在高流量负荷时,比 LABERIO 表现要差。

图 3 给出了 3 种算法的平均丢包率的实验结果。

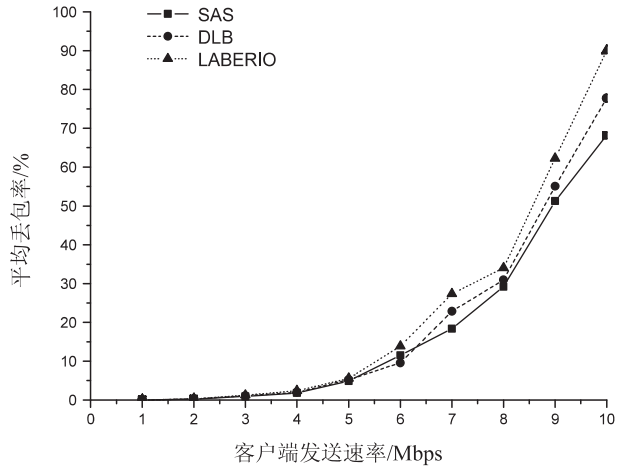

图 3　多业务流时不同机制对平均丢包率的影响

从平均丢包率随流量负荷变化而变化的情况看,SAS 平均丢包率最低,LABERIO 平均丢包率低于 DLB。整体上 LABERIO 好于 DLB,这归功于全局选路降低了拥塞出现的概率,减少了需经过拥塞路径才能递交的报文。SAS 丢包率的变化趋势跟其他算法一致,在具体数值上小了几个百分点。当客户端发送速率达到 9 Mbps 时,三种算法的平均丢包率都超过了 50%,这也说明当网络流量负荷较高时,网络中各链路都较为拥堵,报文在交换机中丢包率会更高。SAS 算法在平均丢包率上表现相对较好,但在高流量负荷时丢包率同样很高。跟现有方案相比,SAS 在丢包率表现上有一定程度的改进。

图 4 给出了三种算法平均抖动的实验结果。

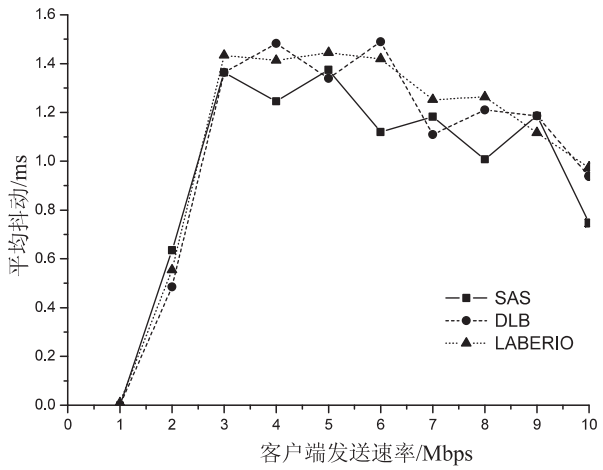

图 4　多业务流时不同机制对平均抖动的影响

从整个曲线上看,SAS 处在最下方,LABERIO 处在 DLB 的下方。SAS 在平均抖动上性能要优于 LABERIO 和 DLB。曲线开始的一段时间内,抖动呈上升趋势,这是因为数据流刚刚开始产生,网络中同时传

输的数据流还未达到最大值,只有部分链路上有流量经过。当传输的数据流达到 4 Mpbs 时,抖动也基本达到峰值,这之后的变化幅度相对更小。

从图 4 中的抖动变化能够看出 LABERIO 平均抖动性能优于 DLB,相对于 DLB 流量分布更为均匀,网络中的流量分配更为均衡。而 LABERIO 是对网络中影响负载均衡的最大流进行调度,当调度到时延敏感的数据流时,肯定会出现流的抖动波动较大的现象。SAS 按流调度对流受调度影响的历史信息进行流调度,优先对受调度影响较小的流。因此,会产生更低的平均抖动。在具体数值上,LABERIO 和 DLB 虽然比文中方案在数值上高了零点几毫秒,但数据的变化趋势相似,SAS 的表现相对较好。跟现有方案相比,SAS 在抖动表现上有一定程度的改进。

4 结束语

文中从调度会对流 QoS 产生影响的角度,尽可能利用流的调度受影响的历史信息,提出了一种基于 QoS 和动态负载均衡的路由策略 SAS。在 floodlight 控制器平台上编写实现 SAS 算法并与 DLB、LABERIO 进行对比。统计各项性能指标可以看出,SAS 算法按照流的链路粘值进行调度,可以保证在满足不同业务流 QoS 需求的同时,有效提高带宽利用率,产生较低的抖动和丢包率,提高 SDN 网络路由转发的效率。

参考文献:

[1] Open Networking Foundation. Software-defined networking: the new norm for networks[EB/OL]. 2012. https://www.opennetworking.org/sdn-resources/sdn-library/whitepapers.

[2] 沈苏彬. 软件定义联网的建模与分析[J]. 南京邮电大学学报:自然科学版,2014,34(3):1-9.

[3] 张园园. 利用 openflow 技术实现网络负载均衡[D/OL]. [2010-12-09]. http://www.paper.edu.cn/releasepaper/content/201012-258.

[4] Handigol N, Seetharaman S, Flajslik M, et al. Plug-n-Serve: load-balancing web traffic using OpenFlow[J]. ACM Sigcomm Demo, 2009, 4(5): 6.

[5] Koerner M, Kao O. Multiple service load-balancing with OpenFlow[C]//IEEE international conference on high performance switching & routing. [s. l.]: IEEE, 2012: 210-214.

[6] 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. [s. l.]: ACM, 2010: 267-280.

[7] Fujita N, Iwata A. Adaptive and efficient multiple path pre-computation for QoS routing protocols[C]//Global telecommunications conference. [s. l.]: IEEE, 2001: 2215-2219.

[8] Long H, Shen Y, Guo M, et al. LABERIO: dynamic load-balanced routing in OpenFlow-enabled networks[C]//IEEE 27th international conference on advanced information networking and applications. [s. l.]: IEEE, 2013: 290-297.

[9] Li Y, Pan D. OpenFlow based load balancing for Fat-Tree networks with multipath support[C]//Proceedings of 12th IEEE international conference on communications. Budapest, Hungary: IEEE, 2013: 1-5.

[10] Kovtun P K, Son D T, Starinets A O. Viscosity in strongly interacting quantum field theories from black hole physics[J]. Physical Review Letters, 2005, 94(11): 111601.

[11] Open Networking Foundation. OpenFlow switch specification, version1. 3[EB/OL]. 2012. https://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.0.pdf.

[12] Issariyakul T, Hossain E. Introduction to network simulator NS2[M]. [s. l.]: Springer-Verlag, 2012.

[13] Surhone L M, Tennoe M T, Henssonow S F, et al. Link layer discovery protocol[M]. [s. l.]: Betascript Publishing, 2010.

[14] 樊自甫,伍春玲,王金红. 基于 SDN 架构的数据中心网络路由算法需求分析[J]. 电信科学,2015,31(2):36-45.