SDN 下的 IPv6 任意播实现负载均衡的路由算法研究

樊 伟,徐 华,李 京

(中国科学技术大学 计算机科学与技术学院,合肥 230026)
E-mail: slimfun@ mail. ustc. edu. cn

摘 要: 在下一代网络协议 IPv6 中,任意播作为一种新型的一对多的通讯模式,能够实现单个发送者与一组接收者中任意一个主机间的数据通讯,因此在负载均衡方面有重要的应用. 而路由选择策略是任意播实现负载均衡的关键,但在传统网络模式下,任意播选择策略存在不够灵活,简单策略无法选择最佳服务器,复杂策略又会增加路由设备负载的缺点. 因此本论文利用软件定义网络技术,实现了一套任意播负载均衡路由框架,利用软件定义网络的可编程性和集中化管理,能够有效提升任意播的效率和灵活性. 与此同时,论文设计了一种软件定义网络下的基于权值的任意播负载均衡路由选择算法,并引入负反馈机制实现了权重的动态自调节,以增强负载均衡策略对不同应用场景的适应能力. 最后论文对提出的负载均衡路由策略进行了性能评估实验,实验将论文提出的算法与轮询算法和基于响应时间的算法进行了性能比较,论文提出的算法的响应时间相比另外两种算法分别缩短了 18. 1% 和 17%,证明了本论文提出的软件定义网络下的负载均衡框架和任意播路由选择策略有较好的负载均衡性能.

关 键 词: 任意播; 负载均衡; 路由算法; 软件定义网络
中图分类号: TP393 **文献标识码:** A **文 章 编 号:** 1000-1220(2019)03-0545-07

Research on SDN-based Load Balancing Routing Algorithm for Anycast in IPv6

FAN Wei, XU Hua, LI Jing
(Department of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, China)

Abstract: As a new "one-to-many" communication mode in IPv6 networks, anycast enables a single sender to communicate with any single member of a group of potential receivers. Therefore, it plays a great role in load balancing application, where the routing selection algorithm is the key component. However, the existing anycast routing selection strategies are not flexible enough. The simple strategies cannot select the best server, and the complex strategies will increase the load of the routing devices. In order to overcome these drawbacks, this paper proposes a new load balancing anycast routing framework based on Software Defined Networks(SDN). Due to the programmability and centralized management characteristics of SDN, the framework can effectively improve the efficiency and flexibility of the anycast routing system. Moreover, this paper designs a weight-based anycast load balancing routing selection algorithm in the framework. Specifically, we introduce a negative feedback mechanism to achieve dynamic self-adjusting of weights, so as to enhance the adaptability of the algorithm for different scenarios. Finally, this paper carries out a load balancing performance evaluation experiment. The experiment compares the load balancing performance of the proposed algorithm with round-robin and latency-based algorithm. The results of experiment indicate that our algorithm can reduce the latency by 18. 1% and 17% respectively.

Key words: anycast; load balancing; routing algorithm; software defined network

1 引 言

随着互联网的发展,IPv4 地址空间不足的问题日益严重,针对此问题被提出的下一代网络协议 IPv6 在过去的一段时间内得到了迅速的发展. 大量学者认为 IPv4 网络服务逐步向 IPv6 网络过渡将成为不可逆转的趋势. 任意播[1] 是 IPv6 中提出的一种新型的"一对多"通讯模式,得益于其路由策略会将数据分组路由到"最近"服务器的特点,任意播能够节约路由和链路资源,增加资源利用率和避免单点失效,因此在 DNS、网站镜像、视频点播等网络服务的负载均衡方面有着重要的应用.

路由选择算法是 IPv6 任意播实现负载均衡的关键,因此针对任意播路由选择算法如何有效实现负载均衡的研究是当前相关领域研究的一个重要方向. 然而现有对任意播实现负载均衡的路由算法的研究存在以下三个难以解决的问题:

1) 较为简单的任意播负载均衡路由选择算法不能准确地选择出最佳服务器. 在现有研究的网络架构中,路由选择策略往往以分布式算法的形式实现并集成于分散的网络交换设备当中,网络架构缺乏集中管理的控制平面导致分布式的路由选择策略难以获取和维护全局的网络信息和服务节点负载信息,因此一些较为简单的任意播路由策略[2-3] 仅通过链路的时延,服务节点的响应时间或者当前活跃的会话次数作为

收稿日期: 2018-01-26 收修改稿日期: 2018-07-26 基金项目: 赛尔网络下一代互联网技术创新项目(NGII20150110) 资助. 作者简介: 樊 伟,男,1995 年生,硕士研究生,研究方向为软件定义网络以及 IPv6 网络技术; 徐 华,男,1990 年生,博士,研究方向为云计算与大数据处理; 李 京,男,1966 年生,博士,教授,CCF 高级会员,研究方向为云计算与大数据处理.

依据来估计网络和服务节点的负载,然而仅仅通过这些信息并不能准确的估计服务节点的负载,导致算法无法较精准的选择出最佳服务器;

2) 通过试图设计较为复杂的分布式路由算法来更准确地计算网络和服务节点的负载,虽然能够比较准确地选择出最佳服务器,但这大大增加了网络路由设备的负载. 例如一些文献[7-9]设计的较为复杂的分布式路由算法,通过路由器采集链路和服务节点的负载信息,或者对任意播树自身信息与请求进行分布式维护与处理来进行路由选择,这类算法虽能较为准确地计算网络和服务节点的负载,但在文献的网络架构下,这些算法既需要路由设备对多种负载参数信息进行采集,还需要对数据包进行转发,这就增加了网络路由设备的负载;

3) 现有研究所处的网络架构下实现的任意播路由策略缺乏灵活性和可扩展性. 在现有研究的网络架构下,任意播路由选择策略均集成于分布式的网络交换设备当中,即数据流的控制平面与转发平面以紧耦合的方式集成于路由设备中. 因此,更新网络中任意播负载均衡路由策略通常需要管理员配置不计其数的网络设备和策略机制. 同时,网络中不同路由设备间的管理协议不同,这更是增加了网络中路由策略更新的复杂度. 所以,在这样的网络架构下,任意播负载均衡路由策略很难实现较高灵活性和可扩展性.

导致现有相关研究存在上述难以解决的问题的本质原因是研究所处的网络架构存在以下局限性: 第一,网络架构缺乏集中的控制平面,导致路由策略难以掌握全局网络信息和准确的服务器负载信息; 第二,网络架构中控制平面集成于网络交换设备当中,导致复杂路由算法会增加交换设备负载; 第三,控制平面和转发平面紧耦合,导致路由策略缺乏灵活性. 软件定义网络[3](SDN)作为一种新兴的基于软件的网络架构及技术,能够将交换设备中的控制平面从设备中抽离,以实现控制平面与数据平面相分离,并通过逻辑上集中的控制层面,对分散的网络设备进行集中控制管理. 因此为了解决现有研究中任意播负载均衡路由策略的诸多问题,论文基于 SDN 架构对 IPv6 任意播实现负载均衡的路由选择算法进行研究,提出了 SDN 下的一套任意播路由转发框架以及一种基于权值的任意播实现负载均衡的路由选择算法.

论文提出的任意播负载均衡路由架构利用 SDN 技术下集中的控制平面,周期采集全局的网络负载信息以及网络中的任意播服务节点的负载信息,然后基于所采集的链路以及节点的负载信息按照不同权重对服务节点的综合负载进行评估,最终路由系统将新产生的任意播服务请求转发给综合负载较轻的服务节点. 考虑到在不同应用场景下网络服务对不同网络资源的消耗不同,可能存在节点的某些网络资源占用很少,但某些网络资源消耗几乎殆尽而导致性能急剧下降的情况,论文对评估节点综合负载的权重算法进行改进,引入了负反馈机制,类似短板效应,在考量节点的综合负载时,着重考虑其网络资源消耗最多的因素.

最终,论文通过实验证明了论文提出的任意播负载均衡路由架构和负载均衡路由选择算法有较好的负载均衡性能. 同时实验表明,论文对权重算法引入的负反馈机制能够有效避免网络资源消耗不均匀导致一些资源消耗过度使得性能急剧下降的情况.

2 相关工作

在 IPv6 任意播实现负载均衡的路由选择算法方面,国内外的不同学者们从不同角度对该问题进行了很多研究.

一些学者从减轻网络交换设备的负载方面出发,通过设计简单的算法,采集较少的负载估计信息,以减轻网络交换设备的负载,但这类研究由于算法较简单,采集的信息较少,导致不能准确选择出最佳服务器. 例如在 Agarwal[3] 等人提出的可扩展部署的任意播系统结构(CCAD 模型)中,以各任意播组成员的响应时间作为依据进行最佳服务器的路由选择,此外还有 Zaumen[4] 以及 Wang[5] 等人也是基于网络链路或服务节点的时延来进行任意播路由选择,其中文献[4]以最小时延为目的,提出了负载均衡的任意播路由协议 MIDAS,而文献[5]则以高概率选择时延小的路径进行数据传输来达到负载均衡. 而文献[6]则通过选取当前会话次数最少的任意播成员作为最佳服务器. 然而,上述研究提出的负载均衡算法较为简单,通过任意播组成员的响应时间或活跃的会话数往往不能准确的估计服务节点的负载,导致无法较精准的选择出最佳服务器.

另外一些学者则从提高路由选择算法对最佳服务器选取的准确性方面出发,通过设计较为复杂的任意播负载均衡路由策略,采集更多的负载信息或者维护更加复杂的状态,来提高算法路由选择的准确性,但这类算法复杂度较高,且要求路由设备采集较多数据,增加了路由设备的负载. 例如 Yamamoto[7] 等人给出了一种主动任意播实现负载均衡的网络构架,并在该构架的基础上进行了改进[8],最终在他们提出的构架中,综合考虑路径拥塞,往返时间以及服务器负载信息作为路由选择的参数,但是该构架中,路由器既需要采集这些参数信息进行路由选择,还需要对数据包进行转发,这大大增加了路由器的负载. 在文献[9]中提出的任意播通信模型中,需要对任意播树自身信息与请求进行分布式维护与处理,这在增加设备负载的同时也增加了算法复杂度.

由于在传统网络架构中,往往存在负载均衡路由策略不灵活,简单负载均衡算法无法进行准确的路由选择,而复杂算法又会增加网络交换设备的负载的缺点. 随着 SDN 网络架构技术的兴起,研究学者们开始尝试使用 SDN 技术解决传统网络架构下负载均衡存在的缺陷.

利用 SDN 技术实现负载均衡的主体思路为: 利用 SDN 的全局性实时监测网络以及服务节点的全局负载信息,控制器通过下发数据转发规则,控制数据流转发向不同的服务节点,从而实现负载均衡. 而各项研究在 SDN 负载均衡框架的设计以及技术的实现上仍然存在较大差异和诸多问题,例如,文献[10]只能实现 Flow level 的负载均衡转发,无法对 TCP 等基于连接的协议提供支持; 而文献[11-14]则需要将客户端产生的请求包上发给控制器,控制器需要对每一次连接进行负载均衡分发,控制器负载大,且存在单点失效的风险; 文献[15]则通过预先下发数据转发规则到 OpenFlow 交换机,将客户端地址进行分组来实现负载均衡. 然而该文献为了保持客户端和服务端的连接,需要控制器对每一个源地址产生的

数据流进行状态维护,因此当网络中存在大量数据流时,仍然无法解决控制器负载较大的问题. 在协议支持方面,长连接的应用场景下,往往无法确定一次长连接中两次数据通信的时间间隔,而文献[15]需要间隔固定时长更新客户端的状态,因此文献[15]提出的路由系统不能很好支持长连接应用. 而对于类似 FTP 服务这样具有数据信道和控制信道双信道的协议,上述的文献均不能很好支持.

本论文通过控制器集中采集全局的网络负载信息,通过预先下发数据转发规则到 OpenFlow 交换机,适度赋予了 OpenFlow 交换机处理局部网络问题的权利. 论文利用 Linux 的 Conntrack[16] 内核模块,使得 OpenFlow 交换机能够追踪数据流的连接状态,从而对基于连接的协议提供支持. 由于控制器不需要维护每个数据流的连接状态,有效减轻了控制器的负载. 同时,由于 conntrack 模块能够对数据流的"NEW,EST,REL,RPL,INV,TRK"六种连接状态进行跟踪,因而本论文基于 Conntrack 实现的负载均衡路由系统能够更好的支持长连接以及 FTP 等特殊应用.

3　基于 SDN 的任意播路由转发系统

本章节主要对 SDN 下的任意播路由转发网络架构,原型系统的设计以及 OpenFlow 交换机流表的设计进行阐述.

3.1　基于 SDN 的任意播负载均衡路由网络框架

如图 1 所示,本论文设计的基于 SDN 的任意播路由转发网络架构的主干网络由 OpenFlow 交换机构成,且要求任意播服务的客户端和服务端都需要直接或间接连接到该主干网络,即均通过由 OpenFlow 交换机构成的网络互联,OpenFlow 交换机中的数据流转发规则由 SDN 控制器集中控制,因此利用 SDN 的可编程性,向控制器中集成实现负载均衡的任意播路由选择算法,将不同任意播数据流根据控制器采集到的负载信息转发向不同服务节点,便可实现负载均衡.

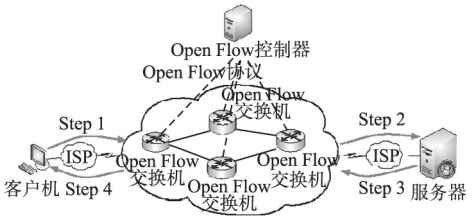

图 1　基于 SDN 的任意播路由转发网络架构

Fig. 1　Network framework of SDN-based anycast route system

本论文实现负载均衡的任意播路由转发过程如图 1 所示,分为以下 4 个步骤:

Step 1. 客户端以任意播地址为目的地址请求任意播服务;

Step 2. 数据分组经过 OpenFlow 交换机构成的网络时,通过集成于 SDN 控制器中的负载均衡路由选择策略选择出最佳任意播服务节点,然后将数据分组的目的地址更改为服务节点的单播地址,并将数据分组路由到该服务节点;

Step 3. 服务节点对服务请求进行响应;

Step 4. OpenFlow 网络将响应数据分组的源地址由服务节点的单播地址更改为任意播地址,并转发给客户端.

由上述过程可见,论文的任意播负载均衡路由转发是基于 NAT 技术实现的,但与传统的 NAT 技术不同的是,传统网络下的 NAT 技术往往是通过一台主机实现数据分组的地址转换的,而本论文的地址转换是通过 SDN 网络的若干 OpenFlow 交换机实现的,而地址转换规则则由 SDN 控制器统一集中管理,这就有效避免了一台主机进行地址转换造成性能瓶颈的缺点以及单点失效的风险.

3.2　基于 SDN 的任意播负载均衡路由系统架构

如上一节上述,任意播的负载均衡路由转发主要通过由 OpenFlow 交换机组成的 SDN 网络以及 SDN 控制器共同实现,其中 SDN 网络主要根据 SDN 控制器下发的数据流转发规则进行地址转换和数据转发,因此 SDN 控制器如何生成合理的数据转发规则是路由系统的核心.

本论文选取 Ryu 作为路由系统的控制器,由于 Ryu 是基于模块化的设计架构实现的,所以本论文通过向 Ryu 控制器中添加模块的方式对 Ryu 控制器进行扩展,故路由系统的架构如图 2 所示,图中未画出 Ryu 的原有模块.

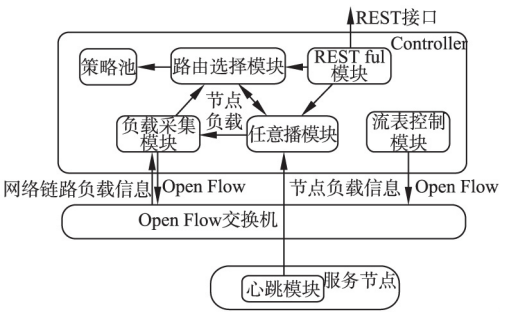

图 2　基于 SDN 的任意播负载均衡路由系统架构

Fig. 2　Framework of SDN-based anycast route system

如图 2 所示,架构中主要包含 6 个模块和一个策略池:

1) 任意播模块: 该模块是系统的核心模块,该模块维护了一个任意播地址到任意播成员地址的映射关系表. 同时该模块还是心跳解析模块,通过解析任意播成员节点提交的心跳信息,将任意播组成员的信息更新到映射表并把负载信息提交给负载采集模块. 该模块通过心跳判断服务节点的存活状态,如果超过一段时间未接收到节点发来的心跳信息,映射表中的节点信息将被清理线程删除;

2) 负载采集模块: 该模块除了从任意播模块获取服务节点的负载信息外,还通过 OpenFlow 协议定期主动获取网络链路负载信息;

3) 路由选择模块: 该模块基于负载采集模块采集的链路及节点负载,利用策略池提供的负载均衡路由选择策略进行路由选择,最终将路由选择的结果反馈给任意播模块;

4) 流表控制模块: 根据负载均衡路由选择的结果生成 OpenFlow 流表更新规则,并通过 OpenFlow 协议更新 OpenFlow 交换机的流表,从而更新 SDN 网络的任意播数据转发规则.

5) 心跳模块: 任意播服务节点通过该模块将节点注册到控制器的任意播模块,并通过周期发出心跳信息将节点的负载信息更新到任意播模块;

6) REST ful 模块: 将路由系统中维护的任意播服务信息

(如存活的服务节点及其负载信息,可用的负载均衡策略等)通过 REST 接口暴露给上层应用,方便上层应用的开发;

7) 策略池: 为了实现 SDN 任意播路由系统中负载均衡策略的灵活性和扩展性,在论文的路由系统中添加了策略池,用于集成不同的负载均衡路由策略.

3.3 流表设计

目前基于 SDN 的负载均衡路由系统主要有主动下发流表和被动下发流表两类实现方式. 其中主动下发流表方式是指控制器通过 OpenFlow 协议主动去更新交换机中的流表; 而被动下发流表则是交换机接收到流表中无匹配项的数据包时,交换机通过 Packet-In 消息通知控制器,控制器再通过 OpenFlow 协议下发流表,告诉交换机如何处理该数据包. 由于在任意播实现负载均衡的场景下,每一次对任意播服务的请求都需要进行服务定位,如果采用被动下发流表的方式实现,所有对任意播服务的请求都需要经过控制器,这样会造成性能瓶颈和单点失效的问题,这与我们的设计初衷相悖,因此本论文采用主动下发流表的方式实现.

由于 OpenFlow 协议本身不支持有状态协议,即 OpenFlow 交换机内的流表只能基于规则对数据包进行无状态的匹配转发. 这会导致基于 SDN 实现的路由系统无法支持类似 TCP 这样有状态,基于连接的协议. 在被动下发流表实现方案下,控制器会针对每一次连接生成相应的数据包匹配处理流表,通过对原地址和目的地址的匹配,能够区分数据包所属的连接,这样就可以将属于同一连接的数据包路由到同一服务节点,实现对有状态协议的支持. 然而在主动下发实现方案中,控制器只是周期地更新 SDN 网络中任意播的处理流表,无法针对每次连接生成区分其他连接的匹配域,因此本论文利用 Conntrack [16,17] 模块对数据包的连接状态进行追踪和分类. Conntrack 是 Linux 内核中的一个跟踪记录连接状态的模块,通常该模块被用于实现有状态的防火墙,本论文则利用该模块在交换机内实现有状态的负载均衡. Conntrack 内部为每一个连接实现了状态机,论文利用该状态机,将无状态的数据包转变为带有连接状态的数据包,从而区分数据包所属的连接.

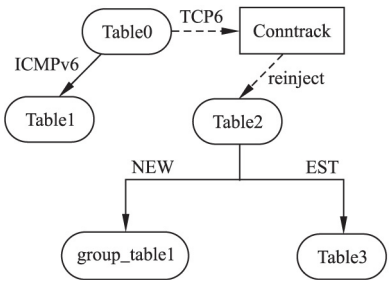

图 3 主动下发流表实现方案的流表设计

Fig. 3 Design of flow table in active mode

论文采用的主动下发流表实现方案的流表设计如图 3 所示. 其中流表 0 对数据包类型进行分类; 流表 1 为 ICMPv6 协议的处理流表,而基于连接的 TCP 协议数据包则需要通过 Conntrack 模块进行追踪标记,以区分数据包所属的连接以及连接所处状态,本论文将连接分为两个状态: 连接尚未建立(NEW)状态,连接已建立(EST)状态. NEW 状态指三次握手还未完成的阶段,在流表中特指客户端发出 SYN 数据包请求连接的状态. NEW 状态的数据包将通过组表 1 进行处理,由于该状态的数据包企图建立一次新的 TCP 连接,因此组表将会选择一台任意播服务节点对该新连接产生 NAT 规则,并将该规则提交到 Conntrack 模块. 组表 1 中包含了任意播服务器组中服务节点数目个 buckets,其中,第 i 个 bucket 中的 actions 定义为: (nat(server_i_uni_ip),commit),组表的 Group Type 设为 SELECT. EST 状态的数据包则表明数据包所属连接已经通过三次握手建立,因此仅需要通过组表向 Conntrack 模块提交的 NAT 规则对数据包进行地址转换即可,即流表 3 仅进行地址转换操作.

3.4 系统可扩展性分析

OpenFlow 将控制平面从网络设备中分离,并移出到 OpenFlow 控制器上,由控制器完成控制层的所有功能,OpenFlow 交换机只保留了基本的数据转发功能. 因此基于 SDN 的负载均衡路由系统的可扩展性问题也就演化成了控制器在复杂网络结构和大量数据量下的处理性能问题 [18] . 为了提高控制器对网络请求的处理能力,可以通过多线程或分布式的技术手段,提高控制器软件本身的处理速度,例如 Beacon, HyperFlow 等; 也可以通过对流表的巧妙设计来减少对控制器的请求,例如文献 [18,19] 通过设计流表,对数据流初始化请求进行聚类处理来减少对控制器请求,但是通过对数据分组源地址进行聚类的方法无法实现细粒度的负载均衡,且文献 [18,19] 提出的实现均不支持基于连接的服务. 本文通过预先下发数据转发规则到 OpenFlow 交换机,赋予了 OpenFlow 交换机处理局域网络问题的权利,并利用 Conntrack 技术使得 OpenFlow 交换机能够独立跟踪数据流的连接状态,不再需要控制器维护连接状态,使得论文提出的负载均衡路由系统能够在支持基于连接协议的前提下,也减少了控制器负载,从而增强了系统的可扩展能力.

为探究论文所提出的路由系统的可扩展能力,论文分别对路由系统能够维持的最大交换机数量(下文称为信道容量实验)和在大量数据流下观测网络丢包率(下文称为丢包率实验)两类情况进行了实验. 论文使用 Ryu 作为网络的 SDN 控制器,控制器的运行环境为: Ubuntu 系统,4 核 CPU,8G 内存.

在信道容量实验中,论文分别对网络中存在不同交换机数量进行了实验,实验结果表明,当网络中交换机数量增加时,控制器处理交换机各种请求时所占用的系统资源也随之增加. 实验以内存为例,当交换机数量为 200,400,600,800,1000 个时,控制器占用内存大约为: 38MB,49MB,63MB,78MB,92MB. 在实验中,当交换机数量达到 800 时,出现少量通道连接断开,Feature Request 消息不发送等问题,当交换机数量达到 1000 时,连接断开现象更加明显,存在接近 9% 的通道连接会断开.

在丢包率实验中,分别对论文提出的主动下发流表策略实现的路由系统和被动下发流表策略实现的路由系统进行实验. 实验中,网络中存在 20 个交换机,交换机首尾相连形成环,每个交换机连接一台主机,共 20 台主机,其中 10 台客户机,10 台服务机. 每台客户机产生一定数量线程,每个线程间隔 0.1s 对随机服务器发起连接请求,持续请求 60s. 论文对线

程数量为 2,4,6,8,10 时进行实验,发现被动下发流表实现方案在线程数为 6 时,便有大约 5% 的连接失败,在线程数为 10 时,连接失败数约占 11%. 而论文设计的主动下发流表实现方案,在线程数为 10 时,连接失败数仍未超过 5%.

通过上述可扩展实验得知,论文设计的主动下发流表的实现方案,通过控制器预先下发转发规则,赋予 OpenFlow 交换机独立转发连接请求的权利,相比被动下发流表实现方案,能够有效减少对控制器的上发请求次数,有效避免了控制器处理过多不必要的负载,因此论文提出的主动下发流表实现方案具有更强可扩展性.

4　负载均衡路由选择算法

任意播服务器的路由选择策略是提高任意播负载均衡效率的关键,本论文采用权值调度算法,综合考虑系统收集到的节点负载信息来设计任意播负载均衡路由选择算法. 在实验过程中,我们发现对服务器的处理能力影响较大的因素主要是服务器的 CPU 占用率,链路的跳数,服务器的响应时间,内存的占用率以及链路带宽的占用率. 其中服务节点的链路带宽占用率取到达该服务节点所经过路径的带宽占用率的最大值,如第 i 个节点的带宽占用率由公式(1)计算得到,其中 u_{ij} 为到达第 i 个服务节点的第 j 跳链路的带宽利用率.

$$l_i = \max(u_{ij}) \tag{1}$$

综合考虑以上五种负载指标,按照不同的权重来计算服务节点的权值,该权值代表了服务器的综合负载,且权值越高,服务器的综合负载越大,故论文设计的权值计算公式如下:

$$\mathrm{H} = W^T L \tag{2}$$

其中 L 是上述五个负载指标的负载情况向量,W 则是影响因素对应的权重向量. 通过公式(2)得到的服务节点综合负载可认为是服务节点处理性能的度量值,其值越大,服务节点的处理性能越差.

然而在实验中,我们发现,当任意负载指标出现瓶颈时都会导致服务节点处理性能急剧下降,即当服务节点 CPU 占用率超过一定阈值时,即使到达该节点的链路的带宽占用率很低,服务节点的处理性能仍然很差. 因此在评估服务节点的处理性能时,应该着重考虑达到瓶颈或接近瓶颈的负载指标,所以在计算服务节点综合负载大小时,还需根据单个负载指标的负载情况调整其在综合负载评价中的权重大小. 因此论文引入负反馈的方式,根据单个负载指标的负载情况来动态调整其权重大小. 权重的负反馈调整公式如下:

$$w_k' = w_k - \alpha^3 \sqrt{\tilde{l}_k - l_k} \tag{3}$$

$$w_k = \frac{w_k'}{\sum_{i=1}^{5} w_i'} \tag{4}$$

其中 w_k 为第 k 个负载指标的权重系数,$\tilde{l}_k$ 为第 k 个负载指标的门限负载,l_k 为当前测得的第 k 个负载指标的负载值,α 为权重调整系数,其值越大,权重调整越快,负载指标的权重与该负载指标的实际负载情况相关性越大. 公式(3)引入负反馈,通过单个负载指标的负载与门限负载间的差值来动态调整该负载指标的权重系数: 当某一负载指标的负载值大于门限负载,则该负载指标对应的权重系数将增加; 当负载值小于门限负载时,对应权重系数将减小. 公式(4)是对权重系数进行归一化处理.

5　实验评估

本文针对 SDN 下任意播实现负载均衡的路由转发进行模拟实验. 实验以服务器的响应时间作为负载均衡策略的性能指标,将论文中设计的负载均衡路由策略与基于 SDN 实现的轮询策略[14]和基于响应时间的策略[11]进行性能对比.

表 1　节点配置

Table 1　Configuration of hosts

节点	CPU/核	内存/MB
SDN 控制器	4	8192
Client	4	2048
Server1	4	2048
Server2	2	1024
Server3	1	512

实验在五台虚拟机上部署实现了论文提出的 SDN 任意播实现负载均衡的路由转发系统,并通过 OpenvSwitch 将虚拟机互连,网络拓扑如图 4 所示. 其中一个节点作为 SDN 控制器,Client 节点作为任意播服务的客户端,另外三台作为任意播服务的服务端. 各虚拟节点配置如表 1 所示.

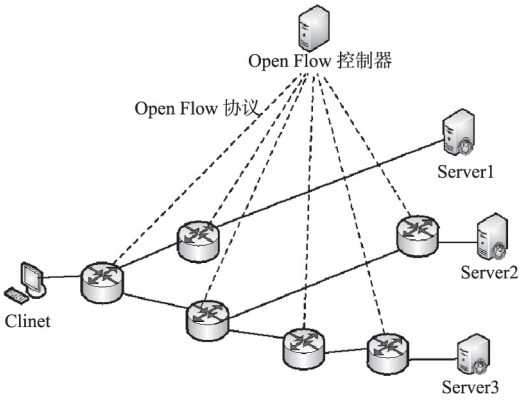

图 4　网络拓扑

Fig. 4　Topology of network

如前文所述: 在不同的应用场景下,可能存在节点的某些资源占用率较低,但某些网络资源消耗几乎殆尽而导致性能急剧下降的情况,因此为了验证论文提出的权重自调节的基于权重的负载均衡路由选择算法能够适应对网络资源消耗不同的应用场景,论文分为以下两类不同应用场景进行实验.

场景 1. 非 CPU 敏感型应用场景. 在此场景下,任意播服务是一个 HTTP 应用,客户端通过任意播地址请求 HTTP 服务,三台任意播服务器中的一台将进行响应并返回一个 HTML 静态页面. 该场景主要在网络中传输一个 HTML 静态页面,对网络带宽消耗相对较多而对服务节点的 CPU 资源消耗较小.

场景 2. CPU 敏感型应用场景. 客户端仍通过 HTTP 协议进行任意播请求,与场景 1 不同的是,服务返回的是 300 次浮点运算的结果,即返回计算密集型运算的结果. 该场景由于需

要在服务节点进行计算密集型运算,因此该应用场景对服务节点的 CPU 资源消耗较大而对带宽等其他资源消耗较小.

实验时,客户机开启 100 个线程对任意播服务并发访问 60s,每次访问间隔 0.1s,最终对不同负载均衡策略下的任意播服务的响应时间进行统计比较.

为验证算法在节点负载不均衡的情况下仍然具有较好的负载均衡效果,在场景 1 中实验对服务节点初始负载不同的情况进行了讨论. 在场景 1 中,实验将服务节点的 CPU 初始负载情况分为以下 4 类:

1) 服务节点的 CPU 初始占用率均为 0%;

2) 服务节点的 CPU 初始占用率均为 20%;

3) 服务节点的 CPU 初始占用率均为 50%;

4) 服务节点的 CPU 初始占用率不同,节点 Server1,Server2,Server3 的 CPU 初始占用率分别为 0%,20%,50%.

对上述四类情况分别进行实验,并且实验结果统计如图 5 所示. 图 5 中横坐标的 0%,20%,50%,DIFF 分别对应了上述的四类情况. 由实验结果可知,对于服务节点的上述四类初始情况,论文提出的基于权值的算法的服务响应时间均最少. 其中,在三台服务器节点初始负载不相同(即第四类)的情况下,论文提出的负载均衡策略的响应时间较轮选算法缩短了大约 17.9%,较基于响应时间算法缩短了大约 14.04%. 结果证明,在场景 1 下,论文提出的带有负反馈的基于权重的负载均衡路由选择算法在上述四类的情况下均有较好负载均衡效果.

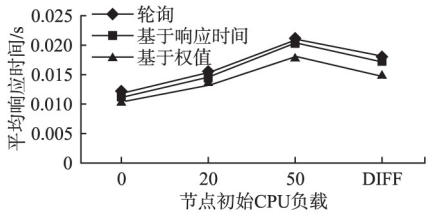

图 5 场景 1 下实验结果数据

Fig. 5 Statistics data of experiment in scenario 1

场景 2 下实验仅对服务器初始负载不相同的情况进行实验. 在该场景下任意播服务对服务节点 CPU 资源消耗较大,因此该场景下 CPU 资源将是影响服务质量的最主要因素,若算法能够有效均衡节点间 CPU 资源消耗量,便能避免 CPU 资源几乎消耗殆尽导致服务质量严重变差的问题. 为验证论文为基于权重算法引入的负反馈机制能有效控制服务对资源的过量消耗,论文在该场景下分别对引入负反馈机制的权重负载均衡算法(Weight-based Load-balancing algorithm with feedback,WLAWF)和未引入负反馈的权重负载均衡算法(Weight-based Load-balancing algorithm with no feedback,WLAWNF)进行实验,并统计对比不同算法下服务节点 CPU 资源消耗的情况,得出算法对资源阈值的控制能力. 该场景下实验结果如表 2 所示.

其中 WLAWF 算法的 CPU 占用率,链路的跳数,服务器的响应时间,内存的占用率以及链路带宽的占用率对应的权重为: [0.2,0.2,0.2,0.1,0.3]; WLAWF 算法的初始权重与 WLAWNF 算法权重相同,而公式(2)中权重调整系数设为 0.5,门限负载 $\tilde{l}_k$ 设置为 80%. 为了避免在负载较低时,负反馈过大导致权重调整越界的情况,算法设定在负载超过 60% 时才开启负反馈机制.

由表 2 数据可知,在该应用场景下,引入负反馈机制的基于权值算法的响应时间较轮询算法缩短了 18.1%,较基于响应时间算法缩短了 17%,且基于权值算法的响应时间方差最小. 从服务的平均响应时间和响应时间方差可看出,WLAWF 算法负载均衡性能优于 WLAWNF 算法.

表 2 场景 2 下实验结果数据

Table 2 Statistics data of experiment in scenario 2

实现方案	算法	平均响应时间/s	响应时间方差
主动下发流表	轮询算法	0.01758	0.0003572
	基于响应时间算法	0.01734	0.0003078
	负反馈基于权值算法	0.01439	0.0002795
	朴素基于权重算法	0.01492	0.0002843
被动下发流表	基于权值算法	0.09472	0.0002897

论文还对流表的两种不同下发实现方案进行了比较,见表 2,被动下发流表实现方案由于每次对任意播服务的请求都需要上发控制器,因此产生了较大的额外时延,在实验中,上发控制器产生了平均 8ms 的额外时延. 因此被动下发流表实现方案效率低下,且存在前文所述的性能瓶颈和单点失效的缺点.

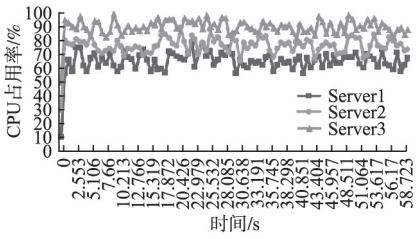

图 6 场景 2 下轮询算法各节点 CPU 占用情况

Fig. 6 CPU utilization of servers of Round-Robin algorithm in scenario 2

不同算法在场景 2 下各服务节点 CPU 占用率的统计图如图 6 至图 8 所示. 由图 6 可以看出,轮询算法只能做到均衡地把服务请求分发给不同服务节点,而无法真正均衡节点间

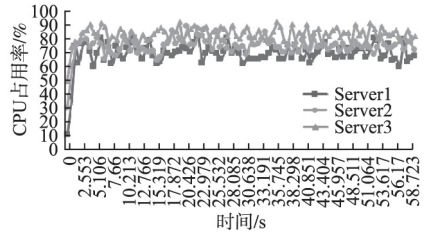

图 7 场景 2 下 WLAWNF 算法各节点 CPU 占用情况

Fig. 7 CPU utilization of servers of WLAWNF in scenario 2

的负载,从而使得性能最优且初始负载最小的 Server1 的 CPU 资源没能充分利用,而性能最差的 Server3 的 CPU 资源接近饱和,自然该算法下的服务质量最差. 而对比图 7、图 8 可知,WLAWNF 的和 WLAWF 算法均能较为有效地利用各

节点的 CPU 资源,均衡节点间 CPU 资源的消耗. 然而 WLAWNF 算法仅能基于固定权重评估节点的综合负载,无法考虑资源消耗接近饱和或超过一定阈值的情况,因而没能通过动态调整权重来避免将新的请求分配给资源消耗超过阈值的节点. 对于WLAWF算法,利用资源当前的消耗量来动

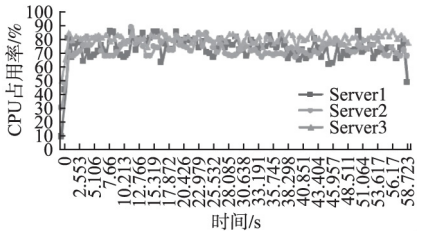

图 8　场景 2 下有负反馈的基于权重算法各节点 CPU 占用情况

Fig. 8　CPU utilization of servers of WLAWF in scenario 2

态调整权重,考量节点综合负载时,优先考虑资源消耗大的因素,因此在图 8 中,当节点的 CPU 占用率较高时,其 CPU 占用率成为节点负载的主要考量因素,因此算法会将新的请求分配给当前 CPU 负载较小的节点,达到均衡负载的效果.

6　结　论

论文对 SDN 下任意播实现负载均衡的路由转发进行研究,实现了一套基于 SDN 技术的任意播负载均衡路由转发原型系统,讨论了不同流表下发实现方案的优缺点,同时设计了一套权重自适应调整的基于权值的任意播负载均衡路由选择算法,实验证明了论文提出的 SDN 下的任意播负载均衡路由系统以及策略具有较好的负载均衡性能.

References:

[1] Partridge C, Mendez T, Milliken W. RFC1546: Host Anycasting service [S]. IEFT, 1993.

[2] Mckeown N, Anderson T, Balakrishnan H, et al. Openflow: enabling innovation in campus networks [J]. Acm Sigcomm Computer Communication Review, 2008, 38(2): 69-74.

[3] Agarwal G, Shah R, Walrand J. Content distribution architecture using network layer anycast [C]. Internet Applications, 2001, WIAPP 2001, Proceedings, The Second IEEE Workshop on, IEEE, 2001: 124-132.

[4] Zaumen W T, Vutukury S, Garcia-Luna-Aceves J J. Load-balanced anycast routing in computer networks [C]. Computers and Communications, 2000, Proceedings, ISCC 2000, Fifth IEEE Symposium on, IEEE, 2000: 566-574.

[5] Wang Jian-xin. Anycast service model and its QoS routing algorithm [J]. Journal of Central South University, 2001, 8(2): 135-139.

[6] Wang Xiao-nan, Qian Huan-yan. A scheme on implementing load-balance anycast service on IP layer in IPV6 [J]. Computer Science, 2006, 33(7): 52-54.

[7] Ikeda H, Yamamoto M, Mura H, et al. A network-supported server load balancing method: active anycast [J]. IEICE TRANSACTIONS on Communications, 2001, E84-B(6): 1561-1568.

[8] Miura H, Yamamoto M. Server selection policy in active anycast [J]. IEEE Transactions on Communications, 2001, E84-B(10): 1-4.

[9] Wang Xiao-nan, Qian Huan-yan, Zhong Lin. A communication model on how to solve anycast scalability in IPv6 [J]. Computer Science, 2006, 33(8): 28-31.

[10] Yong W, Xiaoling T, Qian H E, et al. A dynamic load balancing method of cloud-center based on SDN [J]. China Communications, 2016, 13(2): 130-137.

[11] Zhong H, Fang Y, Cui J. LBBSRT: an efficient SDN load balancing scheme based on server response time [J]. Future Generation Computer Systems, 2017, 68: 183-190.

[12] Alnajjar A, Teed S, Indulska J, et al. Flow-based load balancing of web traffic using openflow [C]. IEEE International Telecommunication Networks and Applications Conference, 2017: 1-6.

[13] Chen-Xiao C, Ya-Bin X. Research on load balance method in SDN [J]. International Journal of Grid and Distributed Computing, 2016, 9(1): 25-36.

[14] Kaur S, Kumar K, Singh J, et al. Round-robin based load balancing in Software defined networking [C]. International Conference on Computing for Sustainable Global Development, 2015.

[15] Mao Q, Shen W. A load balancing method based on SDN [C]. Seventh International Conference on Measuring Technology and Mechatronics Automation, 2015: 18-21.

[16] Ayuso P. NNetfilter´s connection tracking system [J]. Login the magazine of USENIX & SAGE, 2006, 31(3): 34-39.

[17] Kozakai Y, Yoshifuji H, Esaki H, et al. IPv6 specific issues to track states of network flows [C]. SIGCOMM, 2007.

[18] Lin Ping-ping, Bi Jun, Hu Hong-yu, et al. MSDN: a mechanism for scalable intra-domain control plane in SDN [J]. Journal of Chinese Computer Systems, 2013, 34(9): 1969-1974.

[19] Wang R, Butnariu D, Rexford J. Openflow-based server load balancing gone wild [J]. Hot-ICE, 2011, 11: 12-12.

附中文参考文献:

[6] 王晓楠,钱焕延. IPv6 中 IP 层实现 Anycast 均衡负载的一种设计方案 [J]. 计算机科学 2006, 33(7): 52-54.

[9] 王晓楠,钱焕延,钟　林. IPv6 中解决 Anycast 扩展局限性的一种通信模型 [J]. 计算机科学, 2006, 33(8): 28-31.

[18] 林萍萍,毕　军,胡虹雨,等. 一种面向 SDN 域内控制平面可扩展性的机制 [J]. 小型微型计算机系统, 2013, 34(9): 1969-1974.