密级：　　保密期限：

北京郵電大學

硕士学位论文

题目：基于 SDN 网络的发布/订阅中间件
路由计算模块的性能优化

学　　号：2014111348

姓　　名：刘昌威

专　　业：信息与通信工程

导　　师：陈俊亮

学　　院：网络技术研究院

2016 年 12 月 24 日

独创性（或创新性）声明

本人声明所呈交的论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京邮电大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名：_____ 日期：_____

关于论文使用授权的说明

本人完全了解并同意北京邮电大学有关保留、使用学位论文的规定，即：北京邮电大学拥有以下关于学位论文的无偿使用权，具体包括：学校有权保留并向国家有关部门或机构送交学位论文，有权允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，有权允许采用影印、缩印或其它复制手段保存、汇编学位论文，将学位论文的全部或部分内容编入有关数据库进行检索。（保密的学位论文在解密后遵守此规定）

本人签名：_____ 日期：_____

导师签名：_____ 日期：_____

基于 SDN 网络的发布/订阅中间件路由计算模块的性能优化

摘 要

物联网中的海量实时数据，对物联网中发布/订阅消息中间件的性能和稳定性提出了更高的要求，而 SDN（软件定义网络）这种全新的网络架构能够很好的满足上述要求。在 SDN 网络中，发布/订阅系统可以通过控制器查看 OpenFlow 交换机的数据，同时对交换机下发流表，最终达到间接控制网络中数据转发的目的。

参考当前已有的基于传统网络的发布/订阅系统，同时结合 SDN 网络的特点，本文提出了针对基于 SDN 网络的发布/订阅系统的路由计算模块的优化方案。通过基于 Steiner 树的路由算法，结合群间拓扑与订阅信息，实现了分布式的路由计算，并能够根据计算结果生成带有消息主题的流表项。

该方案同时将拓扑管理模块和订阅管理模块一并进行了改进。改进后的拓扑管理模块能够与 OpenDaylight 协作，感知集群中的 OpenFlow 交换机；而通过实现握手机制，实现了集群间邻居关系的构建和维护。通过对订阅表进行聚合和分裂，以及在 OpenFlow 交换机中实现的多级流表机制，节点得以在控制流表尺寸的同时，提升路由查找的效率。

优化后的路由计算模块与原系统的模块相比，路由的计算和查询效率都得到了提高，同时分布式的路由计算方案也使得系统的扩展性更好。最后，在基于 OpenvSwitch 和 KVM 的多集群测试网络中，对系统的路由计算及相关模块进行了功能和性能测试，验证了本文设计的可行性和有效性。

关键词 发布/订阅系统 SDN OpenDaylight 拓扑管理 路由计算

OPTIMIZATION AND IMPLEMENTATION OF ROUTE MODULE OF SDN-BASED PUB/SUB INFORMATION MIDWARE

ABSTRACT

In the Internet of Thing, data flow is in real time and large scale. Therefore, it requires information middleware to be more efficient and reliable. This requirement brings challenges to distributed information middleware. The main concept of SDN is the separation of control layer and data layer of network devices. Therefore, with this concept, SDN-Based Pub/Sub Information Middleware can acquire OpenFlow switches' status and down flows into them using RESTful APIs of controller, and eventually control data transmission in the network.

With work over Pub/Sub system in the traditional network and new features of SDN, this paper introduces a new approach to optimize the efficiency of the route calculation module of SDN-Based Pub/Sub Information Middleware. Route calculation module is redesigned using new algorithm based on Steiner Tree. With inter-group topology and subscription information, distributed route calculation and flow generation is enabled in route module.

This new approach also upgrades topology module and subscription module. In association with a new controller, OpenDaylight, system is able to sense and construct physical topology of the group. The topology of the whole network can be obtained through Hello process. With unite and split process on subscription table, along with 2-level flow-table design, messages can be forwarded in a much higher speed with better accuracy.

The routing module has a better calculation efficiency compared with the old one. The distributed calculation approach also makes this process more stable. In the last part of our work, we established a multi-group experiment environment based on OpenvSwitch and KVM to check the

functions and efficiency of each module. The results prove our work to be reliable and stable.

KEY WORDS publish/subscribe system, SDN, OpenDaylight, topology management, route calculation

目录

第一章 绪论

1.1 研究背景

当今科技发展日新月异，“物联网”已不仅仅只是一个科学界的专有名词，它在各个领域都迅猛发展，深入到了工业生产和日常生活的方方面面，其中蕴含的信息量更是指数级增长，这样的海量实时数据，对物联网消息传输平台的性能和稳定性提出了更高的要求。

传统的物联网组件和服务器之间“请求/应答”的消息传输模式由于处理周期长、并发性易受多方因素影响，在当前的海量实时数据传输环境中已经不能胜任。而发布/订阅系统在信息的发布者和订阅者之间提供了松耦合的信息分发逻辑，具有更好的可缩放性。但基于逻辑拓扑构建的发布/订阅系统在物联网平台中很难维护，因为物联网组件间通信要求以极低时延传输海量的数据，而在这基础上再进行逻辑拓扑的管理和维护、在每个节点处对消息进行拆包和重新封装，显然是不经济的。同时，传统的发布/订阅系统工作于应用层，无法保障系统的可靠性和实时性，而通过 OpenFlow 协议，系统可以将自身对消息的控制扩展至数据链路层，减少了消息转发过程中的冗余环节，提高了消息转发的效率。

SDN 的核心思想是分离网络设备控制面与数据面，为核心网络提供相对纯净的环境。这种思路应用到发布/订阅消息中间件中，就是强调拆分消息中间件的控制层与转发层，用数据链路层中的“流交换”替换应用层中的“包转换”，用“集中管理，分布计算”取代“单独配置”[1]。发布/订阅程序可以通过控制器开放的接口来向控制器发送请求，从而间接对 OpenFlow 交换机中的流表进行增删改查，实现有效管控数据流转[2]。

虽然将系统从应用层移动到数据链路层意味着能够从更细粒度来对消息的转发进行管理和监控，但是与此同时，更多的底层处理被交还到了开发者手上，在将发布/订阅系统从传统网络迁移到基于 SDN 的下一代网络架构中时，如何在高实时性、大数据量的压力下实现消息的有序传输，如何在基于物理拓扑的网络环境中对节点和集群进行高效管理，这些都是重要的研究课题。

同时，由于控制器对物理拓扑的管理压力会随着网络中节点个数的增长而迅速增大，原先由管理员集中式进行管理的思路已不能满足实际网络情况，因此新

系统将集群间路由的计算等功能设置到了集群中的普通节点上，管理员只从集群控制器中获取各集群的具体情况。

因此，本文在 SDN 网络的基础上，将着重设计并实现基于物理拓扑的路由计算方案，同时优化当前基于集群的拓扑获取与维护模块，从过去集群代表负责整个集群的数据同步与路由计算，改进为基于普通节点的分布式路由计算，实现“分布计算，集中管理”；同时设计两级流表结构以及流表聚合和分裂的实现。

1.2 研究内容

本系统是一个以软件为主、软硬件结合的网络应用系统，其目的是借助 SDN 网络高度集中式管理、可编程的、可动态改变的特点，解决传统网络中无法保证的数据时效性、安全性等问题，构建一个可控且可靠的统一消息中间件网络。

SDN 网络的思想是分离交换设备的转发平面和控制平面，即在 OpenFlow 交换机上实现数据转发，而节点则通过控制器开放的接口，间接对数据的流转进行控制，从而实现了控制层和数据层的分离[3]。根据已有策略，在 OpenFlow 交换机底层进行路由查找和转发。为此节点上的路由模块需要能高效完成全网拓扑的获取，并在满足连通性的基础上，根据网络资源的实时使用情况动态地计算最优路径。

主要实现目标为：完成节点功能的重构与优化，如拓扑的构建和维护，用户订阅及策略信息的管理，多级流表的设计和实现，主题的分裂与聚合，路由的查找和计算以及全网络中集群间路由路径的计算等。

目前，基于 SDN 网络的发布/订阅系统已经实现，但仍存在一些问题，如集中式的路由计算与下发，本课题将着重改造拓扑维护与路由计算模块，将路由计算功能下放到每一个普通节点，将原有的代表/代理双层节点设置，升级成为统一等级的节点，同时基于 SDN 网络的实时统计特性，根据全网流量情况动态计算调整网络流表信息，保证系统数据传输的实时性与路由路径的最优化；通过对异步消息处理机制的研究寻求一个可大幅提高接口速度的方案也是本课题的一个着眼点。

结合以上分析，本课题的主要研究内容和工作包括以下三部分：

1.2.1 基于 SDN 网络的物理拓扑维护

传统发布/订阅系统在传统应用层网络环境中，首先要计算邻居并生成逻辑拓扑，再以此为基础计算路由，这样的思路已不能很好满足当前的业务需要，因

此在升级到基于 SDN 网络的时候，拓扑构建的流程也需要升级。

传统网络中的拓扑是基于逻辑邻居，而在 SDN 网络中，控制器可以直接获取当前网络中所有注册在自身的 OpenFlow 交换机的物理连接情况，因此节点不再需要进行逻辑邻居的计算，而是直接向集群控制器获取信息。而针对集群间的拓扑维护，则需要通过研究基于 OSPF（Open Shortest Path First，开放式最短路径优先算法）协议产生的 Hello 探测机制，结合 SDN 网络的特点，设计并实现集群间邻居关系的探测与构建[4]。

1.2.2 基于 SDN 网络的分布式路由计算

基于 SDN 网络的发布/订阅吸收了基于传统 IP 网络的发布/订阅系统和基于 SDN 网络的发布/订阅系统的优点，将其结合在一起形成了“集中管理，分布计算”的体系结构，将根据 Steiner 树计算路由的功能下发到每个节点中，只在流表下发时才统一通过集群控制器进行。

利用流表的好处在于，节点将不需要执行消息拆包、读取主题、查找路由等操作，OpenFlow 交换机可以直接将消息头部的匹配项进行解析并对照流表逐项匹配，进行转发。路由网络中数据流的典型操作分两个步骤：首先根据收集到的拓扑信息，针对数据包的主题，计算出路由路径；之后向集群控制器请求，添加相应的流表项到路由路径上的所有交换机。控制器是整个系统的大脑，负责整个网络的控制逻辑。

1.2.3 基于 SDN 网络的订阅动态管理

在存储订阅信息时，订阅会根据其主题在主题树中的位置，对订阅关系和 OpenFlow 交换机上的流表，进行合并更新，也就是主题聚合，通过聚合系统可以实现微冗余传输，大幅度的减少 OpenFlow 交换机中主题的无效匹配次数，提高匹配的命中率。而分裂则能保证流表的冗余控制在一个合理的范围里。

1.3 论文组织

本文研究发布/订阅系统在 SDN 网络中的路由及其相关模块的优化与实现，重点放在路由算法的分布式计算架构上。鉴于此，本文的组织结构如下：

第一章为绪论，介绍了本文研究的背景和主要内容；

第二章为关键技术概述，主要介绍了基于传统网络的发布/订阅系统、基于 OSPF 协议的链路维护方法、分布式路由算法以及 SDN 网络的相关技术；

第三章叙述系统的需求分析，包括拓扑构建与维护、订阅管理、路由计算、

自定义匹配以及多级流表，为系统的设计与实现提供了目标；

第四章为系统的设计方案，简要描述了与需求相对应的拓扑构建与维护、订阅表的聚合和分裂、多级流表以及分布式路由计算等功能模块的优化设计方案；

第五章叙述路由计算以及其他相关模块的优化与实现，包括各模块中的类图设计、业务流程等；

第六章讨论系统的测试和验证，主要包括测试环境的描述、测试网络的搭建以及测试用例和结果分析。

第七章为总结和展望，对本系统的优化工作进行总结，并对未来系统的进一步改进提供意见。

1.4 本章总结

本章简要说明了基于 SDN 网络的发布/订阅消息中间件的建模，以及它与传统网络下的发布/订阅系统的联系，同时介绍了本文研究的主要内容，立足于 SDN 网络的特性，结合基于传统网络的发布/订阅系统的不足，提出相应的设计方案。确立本文论述的主题。同时，本章描述了本文的结构层次，方便读者更好的理解针对新系统进行的优化与实现的过程。

第二章 关键技术概述

本文论述的重点是基于 SDN 网络对发布/订阅系统进行的优化，其中既有对传统发布/订阅中拓扑维护和路由计算模块的改进，也有针对控制器以及 OpenFlow 交换机进行的有针对性的设计与优化。

因为系统是脱胎于早先基于传统 IP 网络的发布/订阅系统，为了使之更好的利用 SDN 网络所具有的特性，提高系统整体的运行效率，因此本文决定从传统发布/订阅系统进行管理与计算的中间层，及其负责与控制器通信交互的接口层来进行优化，下面就简单介绍其中涉及到的关键技术。

2.1 发布/订阅系统

发布/订阅系统中发布者发布主题，订阅者订阅特定主题，发布/订阅系统则负责将发布者发布的消息传送给所有相关订阅者。为了适应海量的实时消息、复杂的网络结构，大规模的发布/订阅系统通常采用分布式的系统结构，将事件处理模块分散到网络拓扑的每个节点上。为了实现消息在各节点之间的转发，这些节点通过部署在网络中的 OpenFlow 交换机互连[5]。

发布/订阅系统的分类方法有很多，但主要有以下 5 类：基于主题的、基于内容的、基于渠道的、基于类型的和混合型的[6]。其中，基于主题的发布/订阅系统由于在事件订阅中的简单配置而具有更大的灵活性，也更有利于管理和维护。因此，这类发布/订阅系统已经在很多物联网工程项目（例如锅炉房监控系统）中使用。

基于 SDN 网络的发布/订阅系统有拓扑构建与维护、订阅管理、路由计算、流表生成与下发、流量监控等几大关键技术，以下主要介绍与本文关系最为密切的集群内与集群间的拓扑维护、订阅表的分裂与聚合以及分布式路由计算三方面。

2.1.1 拓扑结构

发布/订阅系统的拓扑结构可以分为集群内与集群间两种，而在这两种结构中拓扑的具体形式又分为集中型、层次型、无环图型、环型、混合拓扑等方式。根据各种结构的特点和具体应用场景的不同，拓扑的选择也应该是不同的。

基于 SDN 网络的发布/订阅系统为分布式架构，不要求高可靠性，对数据的转发效率有较高要求，而层次型拓扑和集中型拓扑从设计思路上来讲属于一类，

对于层次型拓扑而言，通知服务器之间存在层次关系是它的特色[7]。在网络中节点经常新增和消失的环境中，集中型和层次型的网络显然不能胜任，而且这两种拓扑在海量数据下均会产生负载失衡。

在进一步说明网络拓扑的选择前，本文先将网络节点作为基本元素来讨论（一方面，在基于 SDN 的网络中 OpenFlow 交换机的地位十分重要；另一方面，OpenFlow 交换机常常和节点程序一同出现，这样有助于简化配置并使得拓扑更加清晰），最终的网络节点模型如图 2-1 所示。

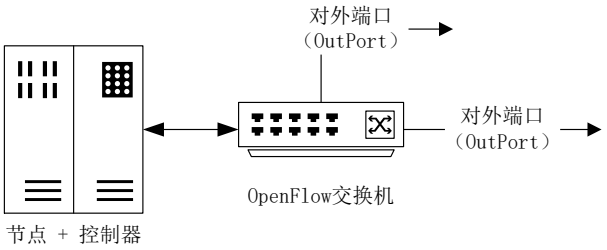

图 2-1 网络节点模型

综合考虑系统稳定性与可扩展性之后，新系统最后在选择拓扑结构时决定采取混合型拓扑，以是否处于相邻地域为依据来构建各个集群，一个集群由一个控制器来进行统一管理，集群内部的 OpenFlow 交换机则直连构成各种形式的拓扑，集群间的连接则是由各集群交换机直连，不设置代表交换机，突出分布式消息系统的特点，同时这样的架构也方便在海量实时消息下进行扩展，同时保证了系统的稳定性与消息转发的实时性。

最终的混合型网络拓扑的架构如图 2-2 所示，网络由于地域以及业务逻辑被分割成若干集群，在不同的集群中分别有自己的组网方式。

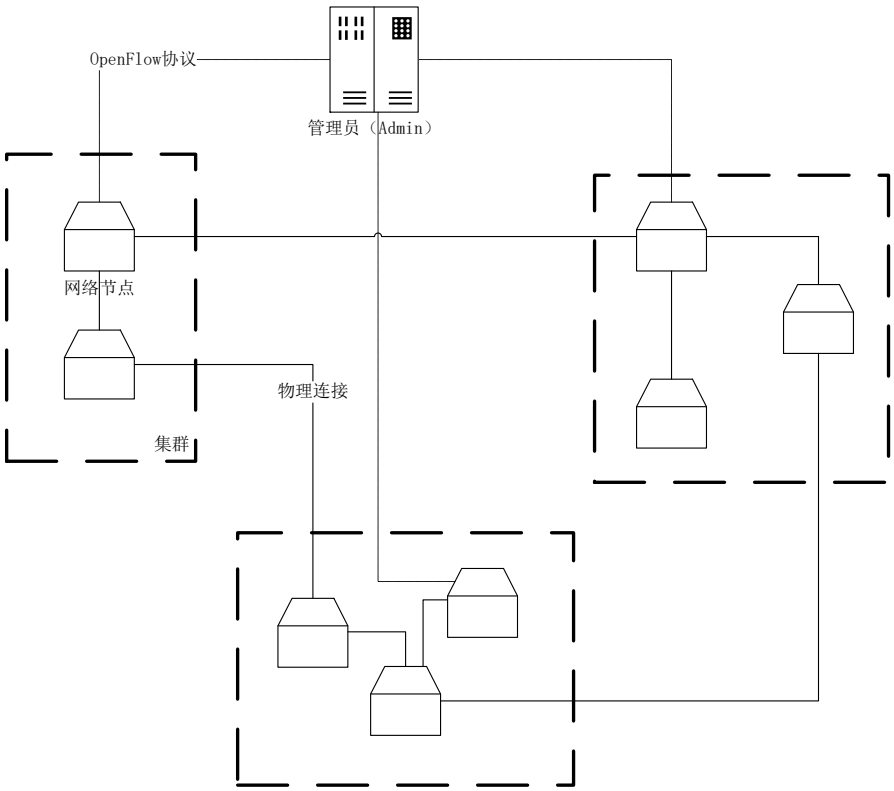

图 2-2 网络拓扑结构图

2.1.2 订阅管理

在进行订阅管理的时候，主要需要考虑主题树的形式以及订阅信息在各节点的保存。如何能够高效节约的在 OpenFlow 交换机上进行主题匹配，在尽量少的匹配次数下完成尽可能多的匹配，成为了亟待解决的问题。

在设计主题树时，参考 SDN 网络的定义，当 OpenFlow 协议规定的流表项匹配时，并不一定需要完全匹配，而只需匹配域值中的某一部分。在这个理论的基础上本文可以采用 LDAP 实现主题树。

LDAP 提供被称为目录服务的信息服务，是一种特殊的数据库，专门针对读取和搜索操作进行了特定的优化。按照业务逻辑将每层主题存储于 LDAP 中，在获取主题的时候自顶向下进行遍历，这样订阅了父亲主题的节点自然也能够收到孩子主题的消息。

在 LDAP 中，连接了 LDAP 数据库的应用实例被称作文件系统代理（DSA，Directory System Agent），它提供了全局的日志系统。在连接数据库的时候，DSA 使用的是异步操作，这样的设计可以保证 LDAP 在被调用的时候能够及时反应。

2.2 OSPF 协议与路由

OSPF 协议能够传递链路状态。它使用“代价（Cost）”作为路由度量，链路状态数据库（LSDB）用来保存当前网络拓扑结构。它的优点包括快速收敛特性（能够在网络的拓扑结构发生变化后立即发送更新报文）、抽象区域划分（自治系统的网络被划分成区域）、有效维护拓扑方法（LSA）和高效的路由计算（SPF 算法）。

2.2.1 集群间状态同步

两个集群如果要交换信息，需要首先建立邻接关系，本文结合 SDN 网络的特点，将不同控制器管理的 OpenFlow 交换机集合抽象为不同集群，其中连接外部其他集群的交换机就被称为边界交换机。在这样的网络环境下，集群间完全建立邻接关系需要经历以下几个状态：

a) 初始态（Down）。该状态表示两个集群的边界交换机之间有物理连接存在，但是尚未与对方有任何信息交互。

b) 尝试态（Attempt）。该状态与初始态类似，表示最近没有从邻居集群收到信息，但由于二者端口仍可能有连接，因此两集群中的一个集群会要求边界交换机再次发送 Hello 消息。

c) 初始态（Init）。该状态表示最近收到过来自邻接集群的消息，意义可类比为 TCP 协议中的一次握手，此时两集群间只有单向通信，并未建立双向连接。

d) 双向通信态（2-Way）。该状态表示两集群间建立了双向通信，这个状态下两集群已经收到对面集群的信息，内部关于边界交换机的数据都已更新。

e) 信息交换初始态（Exstart）。该状态是同步 LSDB 的准备状态。

f) 信息交换态（Exchange）。该状态下，集群将会把它的 LSDB 发送给邻接的集群，与此同时集群有能力接收和发送所有基于 OSPF 协议的包。

g) 信息加载态（Loading）。该状态中的集群开始向之前发现的邻接集群请求最新的 LSA 集合，以此来更新自己的 LSDB。

h) 完成态（Full）。该状态表明两集群间的连接已成功建立，LSA 也已经交换完成，所有的变动都已经在 LSDB 中更新并保存。

在基于 SDN 的网络环境中，根据 OSPF 协议建立邻接关系的状态机如图 2-3 所示。其中灰色为稳定状态，白色为瞬时过渡状态。

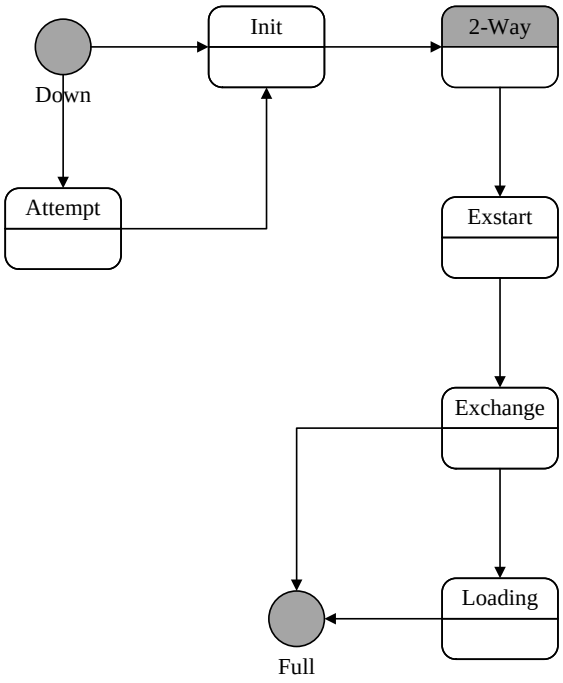

图 2-3 OSPF 协议状态机

2.2.2 SPF 路由算法

OSPF 协议通过感知拓扑中链路的变化，譬如链路失效或者链路复杂过大，以此来计算拓扑中的最短路径树（Shortest-path Tree），并以此树作为后面生成路由表的依据。

在节点中存储的集群信息会根据 LSA 同步获取到的链路信息来更新自己的内容，根据其中的集群间的距离、链路中的数据负载以及链路带宽等数据，计算出链路的“代价”。因为 OSPF 协议基于 SPF 算法，其核心是 Dijkstra 算法。

Dijkstra 算法是求解单元最短路径的有效算法，当图 G = (V，E)不存在负权边时，将图中的顶点划分为两组集合，第一组 set 存储已经求得最短路径的顶点（初始时只有目标节点），第二组 set 存储未确定最短路径的顶点。同时，用 path 记录各个顶点到达目标节点的最短距离（若不存在则记为最大值），从目标节点开始向外扩展，每次找出 path 中的最小值，将其对应的顶点转移至第一组 set 中，并且根据该节点的相邻节点刷新 path 中的内容。直到扩展到第二组 set 为空，此时所有顶点都求得到达目标节点的最短距离。

2.3 SDN 网络中的控制器与流表

SDN 网络主要由控制器和 OpenFlow 交换机两部分组成，控制器作为控制平面，安装在与 OpenFlow 交换机有物理连接的主机上，主要通过 OpenFlow 协议

对注册在它上面的交换机进行管理[8]；而 OpenFlow 交换机则根据自身所带的流表，完成数据平面的消息转发功能。在设计并实现基于 SDN 网络的发布/订阅系统的过程中，新系统最终选择了 OpenDaylight 作为控制器。

2.3.1 OpenFlow 协议

OpenFlow 由斯坦福大学的研究者们于 2008 年提出，在 2009 年被麻省理工大学评委十大前沿技术，2011 年，McKeown 等学者组成了开放式网络基金会，专门负责 SDN 相关标准的制定和推广，发布了一系列标准和文件，极大的推进了 OpenFlow 和 SDN 网络的标准化工作。虽然从 SDN 的整体发展来看，OpenFlow 协议只是众多 SDN 控制平面与数据平面之间通信协议中的一种，但由于它的灵活性与规范性，它已经成为 SDN 通信协议事实上的标准[9]。

OpenFlow 交换机主要负责数据转发功能，其主要由三部分组成：流表（Flow Table）、安全信道和 OpenFlow 协议，其结构示意图如图 2-4 所示。

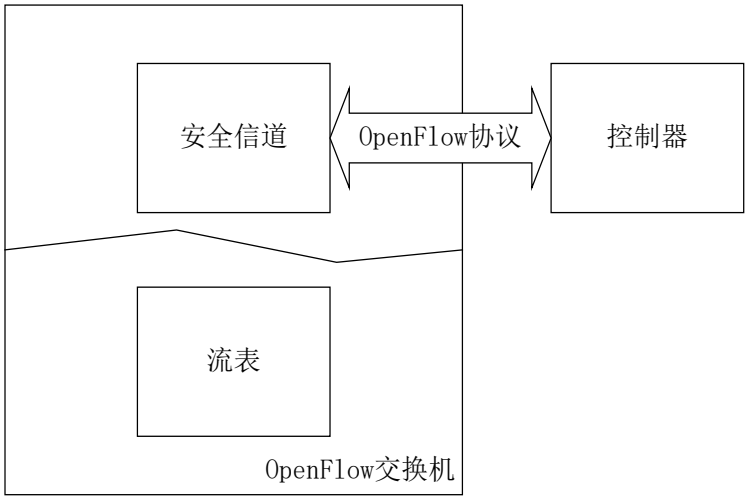

图 2-4 OpenFlow 交换机结构示意图

OpenFlow 交换机的最小处理单元是流表，每个流表中包含若干条流表项，流表项中有针对不同匹配项的转发规则，进入 OpenFlow 交换机的数据包通过查询流表获取对应的操作，流表项中起作用的主要是匹配字段（Match Field）和操作（Instruction）两部分，匹配字段是 OpenFlow 预先定义的项目，其中涵盖了数据链路层、网络层以及传输层大部分标识[10]；而操作则是标明了该流表项匹配的数据包应该执行的下一步操作，这些操作包括计量后丢弃（Meter）、将指定行动添加到正在运行的行动集中（Write-Actions）、立即进行指定行动（Action）和指定流水线处理进程中的下一张流表（Goto-Table）等。

安全通道是 OpenFlow 交换机和控制器之间进行交互的接口，控制器按照 OpenFlow 协议规定的格式通过这个接口来向 OpenFlow 交换机发送控制消息[11]。

OpenFlow 协议规定了 OpenFlow 交换机的端口分类，包括物理端口、逻辑端口以及保留端口，在进行应用时，一般都是将物理端口作为消息的进出端口，流表中进行匹配的也是针对物理端口，对于标准端口 OpenFlow 还定义了端口计数器，方便进行流量监控时直接读取具体数值[12]。

2.3.2 OpenDaylight 控制器的主要应用

控制器可以认为是一个网络操作系统，它实现控制逻辑功能，对网络实现了可编程控制[13]。在一般的 SDN 网络中，控制器是控制核心，OpenFlow 交换机是操作实体，控制器维护全网的拓扑，包括 OpenFlow 交换机的端口信息以及数据包的流量情况。从 SDN 架构看，OpenFlow 是一种控制平面和数据平面间的通信协议，可以称为南向接口；除此之外还应该有位于应用层和控制层之间的接口供开发人员调用，也就是北向接口，一般认为是由控制器提供的，正是通过这种模式，开发人员可以动态的来对 SDN 网络进行配置、管理和优化底层网络资源，实现灵活可控的网络。SDN 网络的架构如图 2-5 所示。

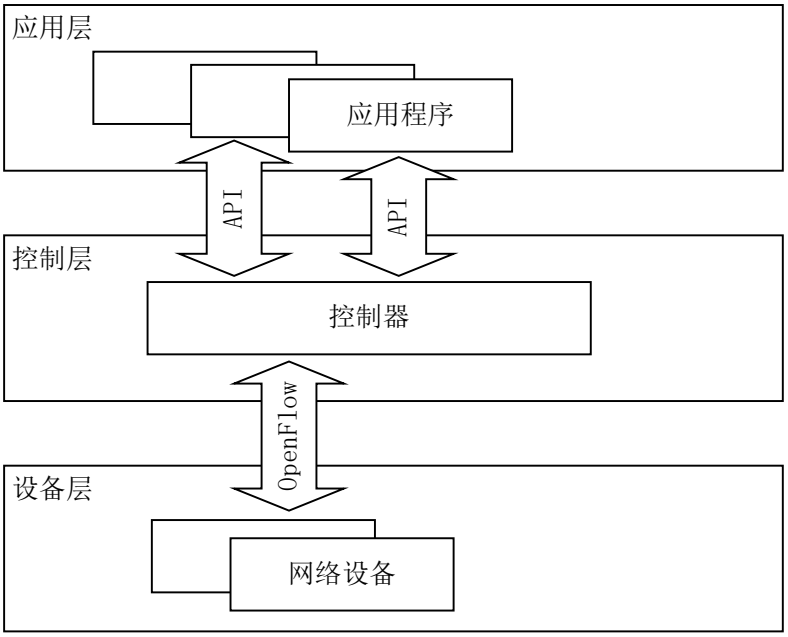

图 2-5 SDN 网络架构示意图

OpenDaylight 是一个开源的、支持 SDN 网络标准的平台，支持多种设备商提供的网络设备及服务，它提供的微服务架构允许用户直接控制应用、协议以及插件，OpenDaylight 同时还提供与外接消费者和生产者联络的接口。当前网络的体系架构已经针对现有的网络需求和负载情况进行过优化，而借助 OpenDaylight 我们可以重新自定义网络的参数，以便进行二次开发。

OpenDaylight 设计之初就是为了适配多种网络模式和对应的开发工作，因此相比 Floodlight 和 Ryu 等控制器，OpenDaylight 规模更大功能也更复杂。它提供

了更多细粒度的接口，包括对 OpenFlow 交换机上的流表进行增删改查以及针对特定主题的消息实现出入队列的操作，让节点可以更精确的管理网络中的设备和流量。

2.3.3 OpenFlow 协议中流表的应用

OpenFlow 协议兼容两种交换机。其中 OpenFlow-only 交换机只支持 OpenFlow 操作，流经它的所有数据包都由 OpenFlow 流水线处理；而 OpenFlow-hybrid 交换机功能更为强大，它可以同时支持 OpenFlow 和普通以太网交换机的操作，即提供二层以太网交换、三层路由的功能。根据流量路由时预先制定的策略，决定流量会被分配到 OpenFlow 流水线，还是普通流水线。

流表被分配到 OpenFlow 交换机的流水线上，同时这些流水线决定了数据包和流表的交互方式。具体过程如图 2-6 所示，每个 OpenFlow 交换机至少包含一张默认流表，考虑到流表丢失（Table Miss）的情况，每个流表中都含有一条最低优先级的流表项来匹配那些没有命中的数据包。

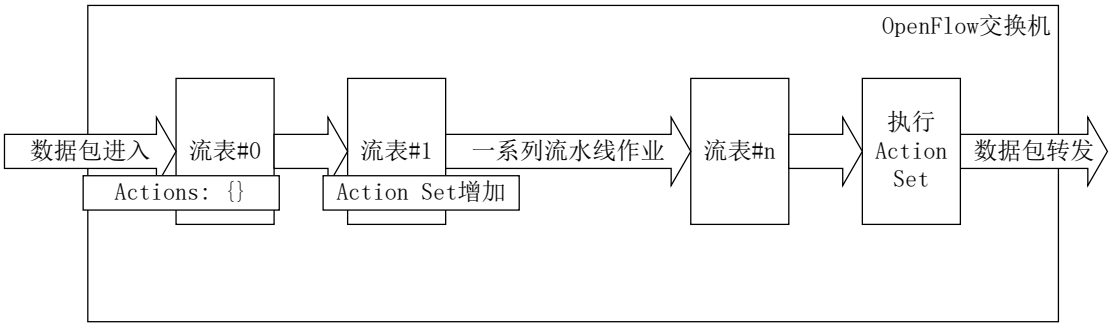

图 2-6 OpenFlow 交换机流水线作业示意图

在 OpenFlow 交换机的流表中的流表项是有序的，他们按照优先级的大小由高到低排列，优先级高的流表项会先被匹配到，在执行过相应的更新动作集、更新匹配项等一系列动作后，数据包会被发给流水线中的下一张流表，流表也是有编号排序的，因此在流水线中前一张流表只能把数据包发给编号比它大的流表[14]。在最后一张流表完成匹配后，OpenFlow 交换机会将数据包所携带的动作集一并执行，通常这意味着数据包要被转发到各处。

2.4 虚拟化技术

搭建虚拟化网络平台进行测试，可以充分发挥配置的灵活性，比如利用基于 Linux 的 OpenvSwitch 和 KVM（Kernel-based Virtual Machine）能够搭建相当稳定的虚拟测试平台。而通过配置脚本文件，实现对集群的批量添加和删除，从而

对路由模块进行更灵活的测试。

2.4.1 OpenvSwitch

OpenvSwitch 是一个适用于生产环境下的、分层设计的虚拟交换机应用，它的设计目的是借助可编程的扩展应用，实现大规模网络的自动化控制。OpenvSwitch 支持如 NetFlow、sFlow、IPFIX、RSPAN、CLI、LACP、802.1ag 等管理接口和协议，此外，类似于 Vmware 的 vNetwork 以及 Cisco 的 Nexus 1000V，它还支持多个物理服务器上发送消息[15]。OpenvSwitch 设计理念是自主、动态地进行大规模网络的控制，与新系统在基于 SDN 网络的发布/订阅消息中间件的设计思路非常契合，因此本文选择 OpenvSwitch 来进行网络设备的虚拟化[16]。

2.4.2 KVM

KVM 是基于内核的虚拟机，能够在 X86 硬件设备下提供一整套完备的虚拟化解决方案。它所提供的内核模块 kvm-ko 针对不同处理器平台提供独立设计的处理器模块，kvm-intel.ko 以及 kvm-amd.ko。KVM 能够为不同的虚拟机分别提供独立的虚拟硬件，同时也支持动态添加删除网卡等操作，这使得测试环境的配置更加简单[17]。

2.5 本章总结

本章主要针对系统实现与优化过程中所使用的相关技术进行了介绍，包括基于传统网络下的发布/订阅系统和 OSPF 算法、SDN 网络中的控制器以及虚拟化技术，其中 OSPF 算法为集群间的消息转发提供了路径，而控制器则为消息直接通过 OpenFlow 交换机的流表进行匹配奠定基础。

第三章 需求分析

虽然本系统的设计模式参考了传统网络中的发布/订阅系统，它的核心目的依旧是为消息的发布与订阅提供接口和转发等功能。设计过程中首先需要将全网物理拓扑进行感知和抽象，将其保存在节点内存。之后对主题树进行编码，将订阅主题结合自定义匹配项，编成 128bit 的二进制字符串下发到流表中。同时在集群间根据 Hello 消息探测机制建立邻居状态，同步链路状态信息与订阅情况，使用 Steiner 树的算法思想计算出路由，通过 OpenDaylight 控制器下发路由涉及的交换机的流表，实现消息的转发和推送。

这三个层次相互关联，协同合作，最终得以保证基于 SDN 网络的发布/订阅系统的可靠性和实时性。

3.1 需求概述

本文中提到的基于 SDN 网络中发布/订阅消息中间件的路由模块的优化，短期目标是在消息传输过程中动态的对订阅主题和流表信息进行调整，探索在 SDN 网络下数据分发实时性与可控性的提高途径。长期目标是进一步解构基于传统网络的发布/订阅系统，将原有的功能进一步拆分，深度利用 OpenFlow 协议定义的规范和控制器能够提供的组件，扩大系统的应用范围，同时增强系统可扩展性和可靠性，实现消息中间件的工程应用。

SDN 网络的主要特性，就是能够直接通过数据包流入流出的情况，获取到当前网络的网络状况，进而对路由计算提供参考和指导，为使具体路由方案能够适应各节点负载和网络状况，集群控制器会监控集群中的链路状态，例如剩余带宽、时延、丢包率等链路状态信息，以及拓扑中节点的上线与否，动态调整算法中的这些参数，实时计算出最优路由方案，提供给控制器，以此来控制 OpenFlow 交换机的包转发来达到效率的最大值[18]。

同时在引入传统发布/订阅系统的主题树时，也需要针对 OpenFlow 协议对其进行重编码，以配合 OpenFlow 交换机中的流表匹配机制，同时由于控制器可以实时获取某特定流表项的匹配情况，而流表项又直接与主题相对应，因此发布/订阅系统可以对订阅主题进行实时地流量管理，实现匹配策略的动态优化。

综上所述，在进行路由计算及其相关模块的优化之前，首先要明确本次优化

的功能需求，如表 3-1 所示。

表 3-1 功能需求描述

功能需求描述	详细描述
集群内物理拓扑的感知	可以通过控制器对集群内的物理拓扑，包括 OpenFlow 交换机和主机，进行感知并直接抽象成对象进行保存
集群间邻居关系的构建	通过发送 Hello 消息进行探测，进行邻居集群的构建
流表设计与下发	针对不同类别的消息，设计构造消息主题的形式
主题树的编码	基于 OpenFlow 流表机制设计主题树的编码，作为流表项中自定义的匹配项
订阅表实时分裂和聚合	在收到订阅和节点运行的过程中，动态调整系统订阅表以及流表的情况
分布式路由计算	在每个集群中控制器节点中，应用 Steiner 树的思想，对相同主题的路由进行计算，生成对应本地集群的流表项并下发

3.2 拓扑构建与维护

传统网络中的拓扑是基于 RNG（Relative Neighborhood Graph）算法生成的逻辑邻居，进而构建逻辑拓扑。在节点数量相对较少时，这个过程并不会对网络带宽资源造成浪费。

但是当网络中节点数量过多时，这些心跳包和随之而来的同步信息（SynLSA）会消耗掉大量的带宽资源，同时，考虑到当网络的规模增大或数据流量增多时，控制网络中的交互信息会显著增加。而 SDN 网络路由的可扩展性还受限于单个处理器的处理能力，因为在保证处理时延的前提下，每个 OpenDaylight 控制器能够管理的网络规模的大小也是有限的[19]。

针对以上情况，最终决定将网络划分为若干集群，每个集群由一个集群控制器进行管理，这样既可以减轻单个控制器的业务压力，也可以保证每个节点的路由层只需给部分交换机下发流表项。这样就可以有效地减少交换机和控制器之间的交互，从而提高发布/订阅管理系统整体的稳定性。

3.2.1 集群内拓扑感知

每个集群都有集群的控制器，它所在的主机也就被称作控制器节点，注意这里控制器节点的定义与基于传统网络的发布/订阅系统中的代表不同，这里只是为标识集群控制器的位置，在节点的功能上以及集群中的地位上，新系统中的控制器节点与其他节点并无任何不同。

节点在感知和维护集群内拓扑时可以直接查询集群 OpenDaylight 控制器，但由于 OpenDaylight 控制器更新这些信息并不是事件驱动的，因此节点在进行路由计算和订阅管理时都需要轮询集群控制器，通过定时任务，实现对节点内数据的持续更新。

同时在轮询集群控制器的同时，控制器节点还需要控制 OpenFlow 交换机进行初始化操作，包括下发初始化流表项、在全网中同步集群内 OpenFlow 交换机信息等[20]。

如图 3-1 所示是集群内拓扑感知维护用例。

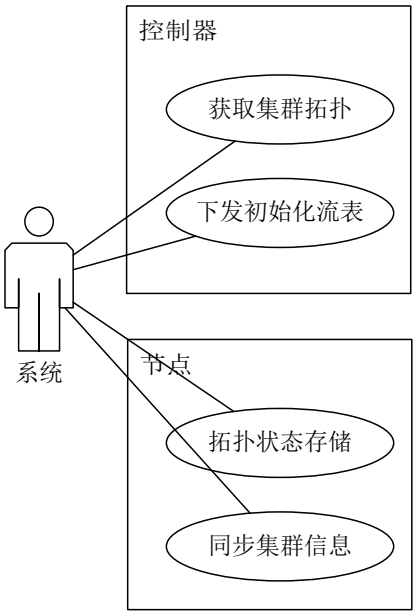

图 3-1 集群内拓扑维护用例

3.2.2 集群间拓扑维护

由于缺少全局控制器来控制全网，各个集群的控制器节点需要构建并维护集群间邻居关系，即由控制器节点采用三次握手的方式，发送 Hello 消息，实现与邻居节点的互连，在连接建立后，将邻居集群的消息保存并在集群内以广播的形式公告。

集群间的拓扑维护同样依靠集群间心跳机制来完成，当集群订阅以及链路状

态等情况发生变化时，控制器节点会在全网同步 LSA，将此处的更新发到其他所有集群。接收到 LSA 信息的节点会将本地存储的相应集群的信息更新，以确保自己在后面计算中的正确性。如图 3-2 所示是集群间拓扑维护用例。

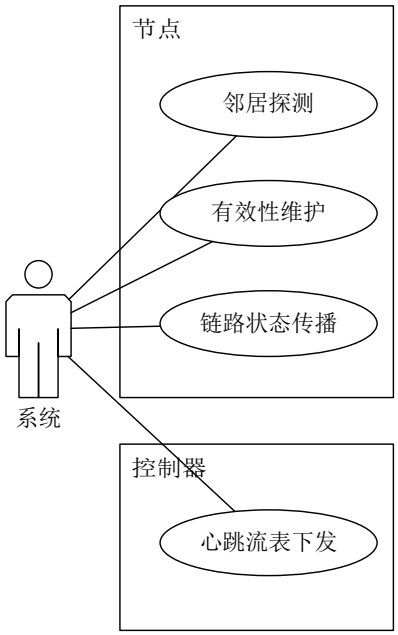

图 3-2 集群间拓扑维护用例

3.3 订阅管理

用户根据主题树对主题进行订阅。这个过程需要遵循树的父子关系，举例来说也就是当父亲主题已经被订阅的时候，孩子主题再被订阅将不会被处理。在用户产生新的订阅后，这条变动消息也要通报整个网络，确保每个节点都知悉这次变更，因为路由计算与订阅主题息息相关，因此只有保证每个节点上保存的订阅情况一致，才能正确的计算出路由，将发布的消息顺利送达订阅者。

同样基于 SDN 网络的发布/订阅系统的主题与流表以及路由绑定，因此主题会与一条自定义匹配项绑定。通过这条匹配项，加上 OpenFlow 交换机支持的流表项精确匹配，节点可以对消息进行更准确的转发，同时节点也可以根据这条匹配项来对流表进行管理。

为了降低流表匹配次数，提高流表匹配命中率，节点还需要对交换机上的订阅表进行聚合和分裂，尽量取得流表匹配次数与冗余流量的平衡。所谓聚合，就是在产生新订阅之后、下发新流表之前进行一次判断，如果这个时候某主题多数孩子主题都被订阅了，那么就把订阅关系转移到这个父亲主题身上，将 OpenFlow 交换机中的流表清理掉一部分。

同样为了避免上述冗余部分过多，节点需要定时进行流量监测，如果某种主题下冗余的流量过大，那么就要对这个主题进行拆分，分裂过程是聚合的逆操作，会将 OpenFlow 交换机中的流表加上一部分。

图 3-3 所示即为系统对订阅管理的用例图。

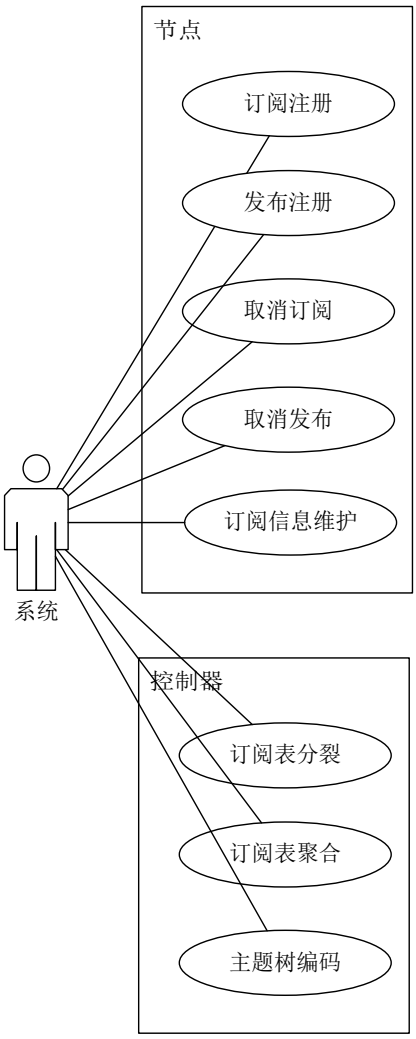

图 3-3 订阅管理用例

3.4 消息路由方案

本文将消息路由的计算设计成事件驱动的分布式计算，每当有新订阅者或新发布者产生的时候，集群都会进行路由的计算，算法是基于 Steiner 树的思想，通过在订阅树上添枝，在确保整个消息通路上的流量最小的同时避免出现中心转发节点导致负载过重，以此保证该主题的消息会被高效转发到所有的订阅用户。在计算路由的时候也要保证不产生回路，确保消息的准确投递。

保证流表下发的同时，借助 OpenFlow 协议以及 OpenDaylight 控制器提供的

流量实时记录功能，新系统能更好地获取到集群间的链路状态，结合集群间的物理距离，产生一个更准确的路径权值，由此节点可以更准确的计算路由，选择经济的路径作为消息的转发通路。

如图 3-4 所示，作为基于 SDN 网络的消息中间件的核心功能，发布/订阅的路由体系是对用户透明的，用户需要关心的只有负责接收它投递消息的 WebService 接口，而用户在完成投递后，路由计算的过程和结果对它都是透明的，它只要等待接收推送（订阅者）或者开始投递（发布者）既可。

而为了减少路由计算的频次，节点会在初始化的时候下发路由同步流表，在计算完某一特定主题的路径后将此路径在集群内同步，当集群内其他节点需要计算相同主题的路径的时候，就可以直接通过本地查询找到，而不需要再重新进行计算。

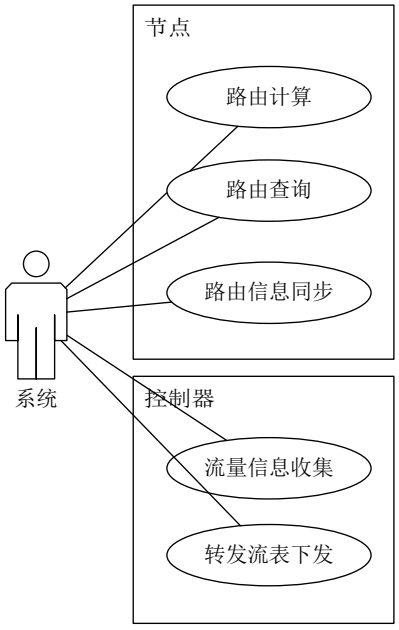

图 3-4 消息路由用例

3.5 自定义流表匹配项

基于 SDN 网络的发布/订阅系统抛弃了基于传统网络的发布/订阅系统的转发逻辑，转而将消息的主题写入消息头部和 OpenFlow 交换机流表项中，以此实现消息直接通过 OpenFlow 交换机来进行转发推送，在数据链路层解决路由选择问题。因此需要在匹配速度与匹配精度上取得平衡，流表尺寸越大，匹配的精度越好，匹配的速度也就越慢；而流表尺寸越小，匹配的精度也就越差，匹配的速度就会提升。

因此将流表分级进行下发会有效减少单张流表的尺寸，上文提到过的订阅表的分裂和聚合则能够提高流表匹配的准确性和效率，而订阅表的分裂和聚合最后还是要落实到流表项的内容上，因此流表管理也需要对匹配项进行自定义。

由于 OpenFlow 协议中定义的默认匹配项，多数只支持二进制编码，因此将文本格式的主题树字串转化成二进制字串有利于节点进行流表的生成与下发。同时还需要将系统管理类消息和普通转发类消息的编码区分开，避免错误匹配流表项导致数据包转发错误。

如图 3-5 所示是系统流表用例，由图可见关于流表的操作需要包括管理员、节点以及控制器的协同配合。

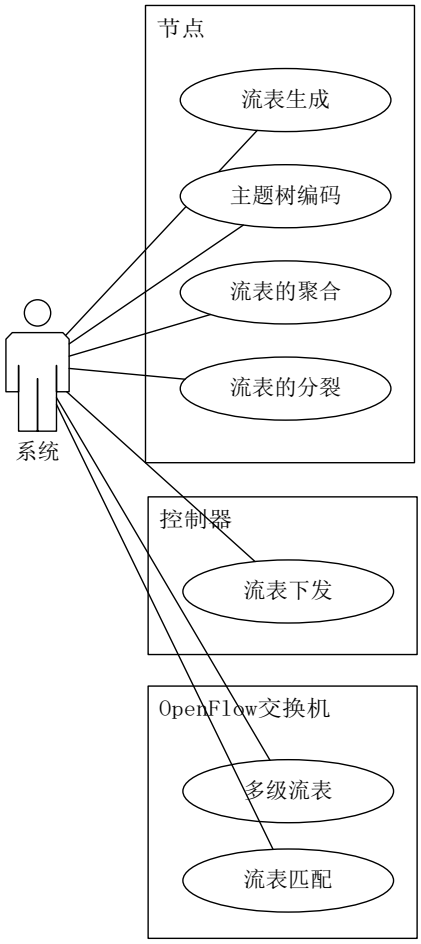

图 3-5 系统流表用例

3.6 本章总结

本章通过对 SDN 网络的发布/订阅系统的介绍，分析了拓扑的构建和维护，详细说明了订阅管理和路由计算方案，阐述了集群间的拓扑感知及维护，解释了集群间链路状态及订阅同步的问题，并对重要的消息路由的计算和下发等内容进

行细致的论述。最后，本章也对订阅管理与流表设计等环节的需求进行了简单分析。

第四章 系统设计

本系统向应用者提供了用来实现消息发布/订阅功能的接口，该系统的核心是路由计算模块，围绕该模块还有若干辅助模块，负责订阅管理和拓扑维护等功能，除此之外该系统还包括了负责集群内事务的集群控制器。下面本文就将针对这几部分的设计思路进行详细说明。

4.1 核心模块架构

在节点程序中，核心功能分为三部分：负责集群内外拓扑维护管理的拓扑层，负责路由计算与流表下发的路由层，以及负责订阅表分裂聚合以及订阅管理的订阅层，如图 4-1 所示。

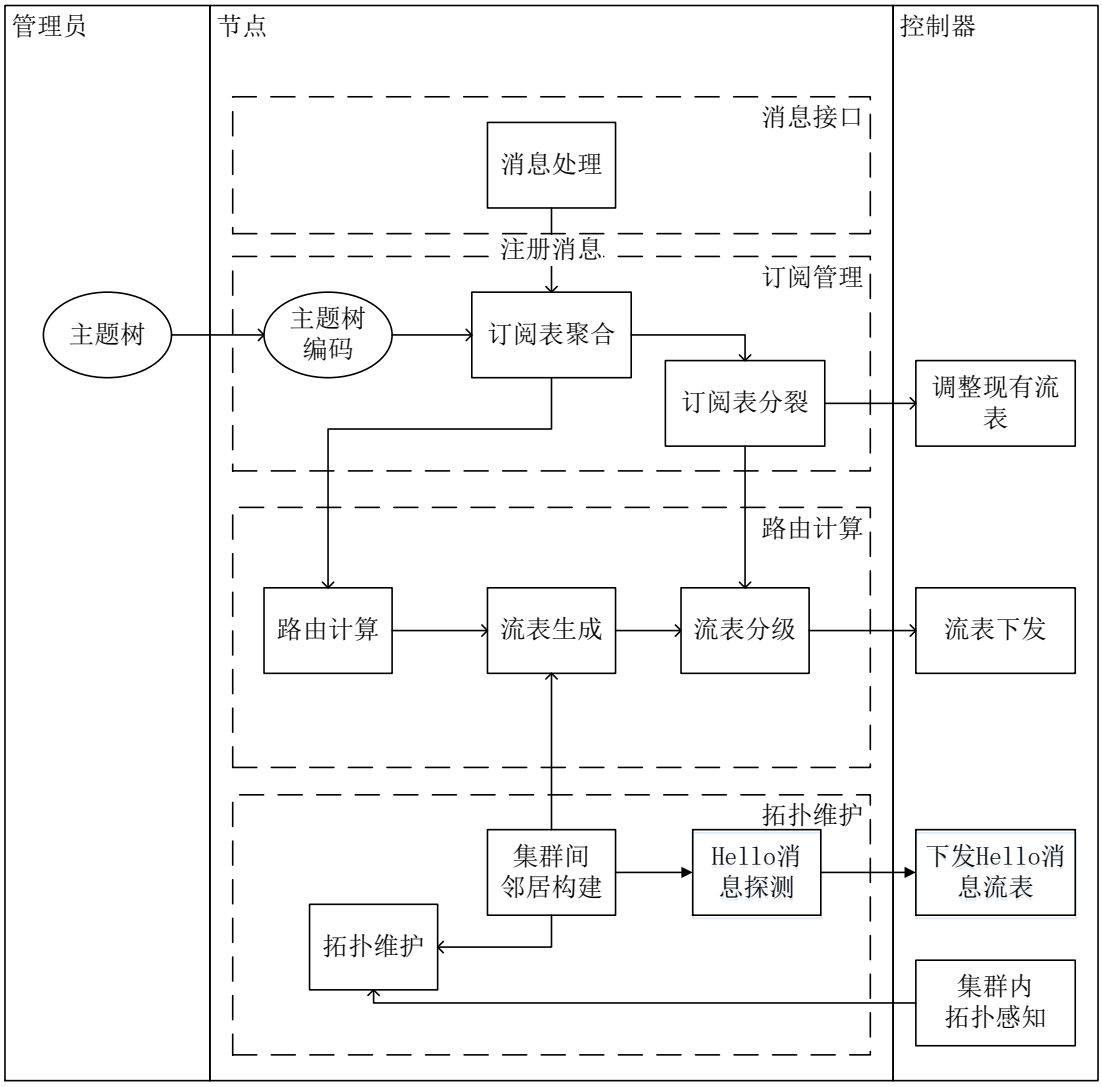

图 4-1 系统核心模块架构图

4.2 基于 SDN 网络的拓扑管理设计

之所以要划分集群，一是由于在网络规模扩大时，链路中的控制信息会不可避免的增加，二是由于单个控制器不能同时管理过多的设备。在基于 OpenFlow 协议的网络环境中，控制器作为重要的一环，可以为节点提供详细的集群内设备和流量信息，而节点在进行拓扑的获取与维护时，也需要通过控制器来进行操作。因此本文决定根据 OpenFlow 交换机所在的位置不同，将其注册到不同的控制器上，这样每个控制器就只能管理全网中一部分 OpenFlow 交换机，其余交换机对这个控制器就是不可见的，控制器只能查看它们与集群外交换机连接的端口号。因此，在设计拓扑管理机制时，需要将拓扑的管理分成两个部分：集群内的拓扑管理和集群间的拓扑管理。

4.2.1 集群内拓扑管理

鉴于集群的划分是基于物理拓扑的，因而传统网络下逻辑拓扑中代表/代理双层管理机制不能很好的满足当前的业务需要。在新的网络架构中，划分集群的目的之一就是减少控制消息的流量，所以为了能让每个节点都获取到实时的集群内拓扑，就需要节点主动向控制器发起查询。

由于查询任务是轮询的，节点会定时向控制器查询集群内的拓扑情况，因此如果出现集群内的节点掉线、OpenFlow 交换机连接丢失、集群内拓扑构造发生改变等情况，节点都可以及时获得。为了方便用户进行集中管理，在节点的配置文件中可以对轮询的时间间隔进行调节。

在基于 SDN 网络的发布/订阅系统中，系统小范围故障的可能性始终存在。在譬如节点掉线、交换机连接丢失等情况下，集群中剩余的节点会按照配置文件中指定的备份控制器 IP 进行重设。而 OpenFlow 交换机上线之初就设置了两个控制器，在发生故障无法连接首选控制器时，它们会尝试连接备份控制器，最终在它的控制下继续正常工作。

由于切换了控制器，之前下发的流表都会消失。因此在新控制器上线后，它所在的节点会重新进行一次启动注册流程，其中除了集群内拓扑的获取外，还要重新对集群间的拓扑进行一次初始化。集群内新增节点的流程如下图所示。

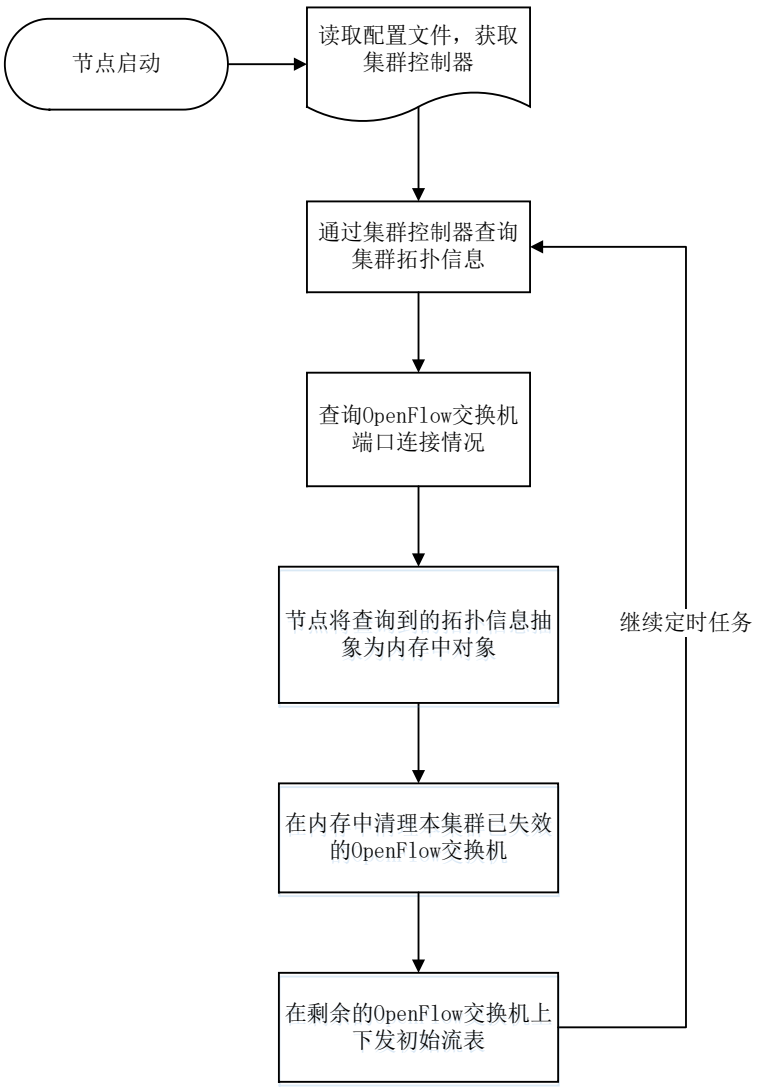

图 4-2 集群内新增节点流程图

4.2.2 集群间拓扑管理

在对集群内拓扑获取功能进行设计时，本文考虑通过定时查询集群控制器来进行拓扑的获取和更新，获取到的数据包括集群中 OpenFlow 交换机的 ID 列表、集群中所有的连接信息（包括主机和交换机的连接，以及交换机之间的连接）、每个 OpenFlow 交换机上已经启动的端口（包括连接集群内设备的端口以及连接其它集群 OpenFlow 交换机的端口）。

而针对集群间的拓扑管理则更复杂一些，其中包括邻居关系的探测与构建、集群间信息的同步等操作。邻居关系的探测可以使用 Hello 消息探测机制，下面就简单就其思路进行说明。

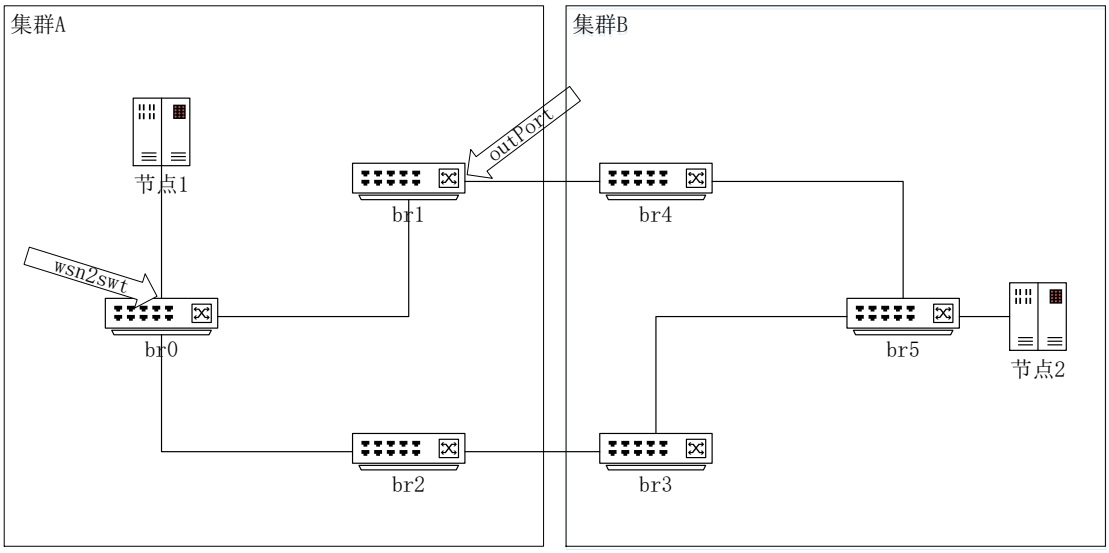

图 4-3 集群间邻居关系构建示意图

如图 4-3，集群 A 中控制器节点启动，它会首先更新本集群内的拓扑，并且将所有的对外连接端口（下文称 outPort）标记出来，此时 A 集群处于 Down 状态。之后 A 集群会转移到 Attempt 状态，按顺序下发通向每个 outPort 的双向流表，而后发送 Hello 消息。邻居集群 B 启动后下发了初始流表，因此可以接收到这条 Hello 消息。这时 B 集群也处于 Init 状态，将会回复一条 Rehello 消息，将传入此条 Hello 消息的边界交换机以及其端口信息封装到 Rehello 消息中。而集群 A 在收到 Rehello 消息后，会更新本地存储的邻居连接状态信息，将集群 B 的信息添加进来，同时将自己的信息封装到 FinalHello 消息中，并发送给新建的邻居集群 B，这样两个集群就建立了邻居关系。此时两集群都转换到了 2-Way 状态，双方可以互相发送心跳消息，也为后面二者交换链路状态信息做好准备。

在集群间邻居信息更新之后，刚才负责收发 Hello 消息的控制器节点，会将更新后的集群信息在全网进行广播，以确保全网所有节点都保有本地集群最新的链路状态信息。这就保证了分布式计算路由信息时，所有计算节点都拥有最新的计算素材。

由于新系统选择构建物理邻居，因此，如果某个集群与过多的集群构成邻居，那么势必会影响它在启动时构建邻居的速度，进而影响它传递消息的实时性和可靠性。所以在组网以及划分集群时，要注意在为每个集群分配相连的 OpenFlow 交换机时贯彻平均化原则。同样的，在划分集群时也需要考虑到效率问题，避免某条集群连接压力过大。

在集群间建立邻居关系后，双方会分别向对方发送自己所保存的全网集群信息，也即 LSDB。LSDB 中保存的 LSA 内容就是集群的状态信息，其消息结构如

表 4-1 所示

表 4-1 LSA 信息结构

名称	含义
序号	LSA 信息的唯一编号
更新时间	此条 LSA 信息的更新时间，决定是否覆盖旧有 LSA
集群名称	发送 LSA 的源集群名称
邻居距离	Map 形式保存源集群到各邻居间的距离，这里的距离是综合网络情况与物理距离后得出的
源集群订阅表	Set 形式保存源集群最新的订阅情况
源集群发布表	Set 形式保存源集群最新的发布情况
失效邻居	源集群丢失的邻居集群的名字

在收到对方发来的 LSDB 后，节点首先要校验其中每条链路状态信息的更新时间，如果此信息更新时间早于本地已存储条目的更新时间，则不进行更新；如果此信息更新时间较新，则将本地条目覆盖。

在首次交换 LSDB 后，邻居集群还会继续同步 LSA 信息，确保链路状态的及时更新。定时同步 LSA 信息还有另一个作用，就是作为心跳信息来确认集群间的连通关系。因为有可能存在这样一种情况，某集群的一个边界交换机激活了 outPort，但是它所连接的另一个集群中的所有节点都失效了，虽然从物理连接上看，集群的邻接关系依旧存在，但是事实上这个节点全部失效的集群已经失去了发布/订阅的功能。只根据物理连接存在与否来判断是否有邻居关系，显然是不合适的。因此新系统使用心跳消息机制，来确保邻居集群中始终有活跃的节点在回复。

而当一个邻居集群长时间不回复心跳消息时，系统就默认这个集群已经失去作用，或称作“下线”。这时节点将操作本集群内存储的 LSDB，更新其中关于这个下线集群的信息，同时将这个变化向全网进行广播。

4.3 OpenFlow 流表设计

在介绍订阅主题的动态管理之前，本文将首先针对基于 OpenFlow 协议的流表机制进行说明，之后对新系统中带有自定义匹配项的流表的设计思路进行说明。因为在订阅管理以及之后的路由计算中，都会牵涉到一个重要的关系，

就是主题与流表项的对应关系，因此在讨论订阅管理之前，本文将首先介绍新系统中，底层消息转发模块中流表的设计。

4.3.1 自定义匹配项设计

基于 SDN 网络的新系统采取了 IPv6 组播的形式进行消息传输，将数据包头中的 ipv6_dst 字段作为自定义的匹配项。发送者向相应主题对应的 IPv6 地址进行 IPv6 组播，接收者加入 IPv6 组播的群组，接收发来的相应主题数据包。这样就解决了数据包头中无法自定义内容的问题。

由于 IPv6 地址有 128bit，因此除了在匹配项中对主题树进行编码外，新系统还将消息类型以及消息入队所需的队列编号在匹配项中进行了定义，使得流表项的分类更加清晰，也充分利用了 IPv6 地址的长度。针对 128bit 的 IPv6 地址，新系统将其中 100bit 的空间预留给了主题树的编码，而从主题树的树形结构编码成自定义匹配项的过程如图 4-4 所示。

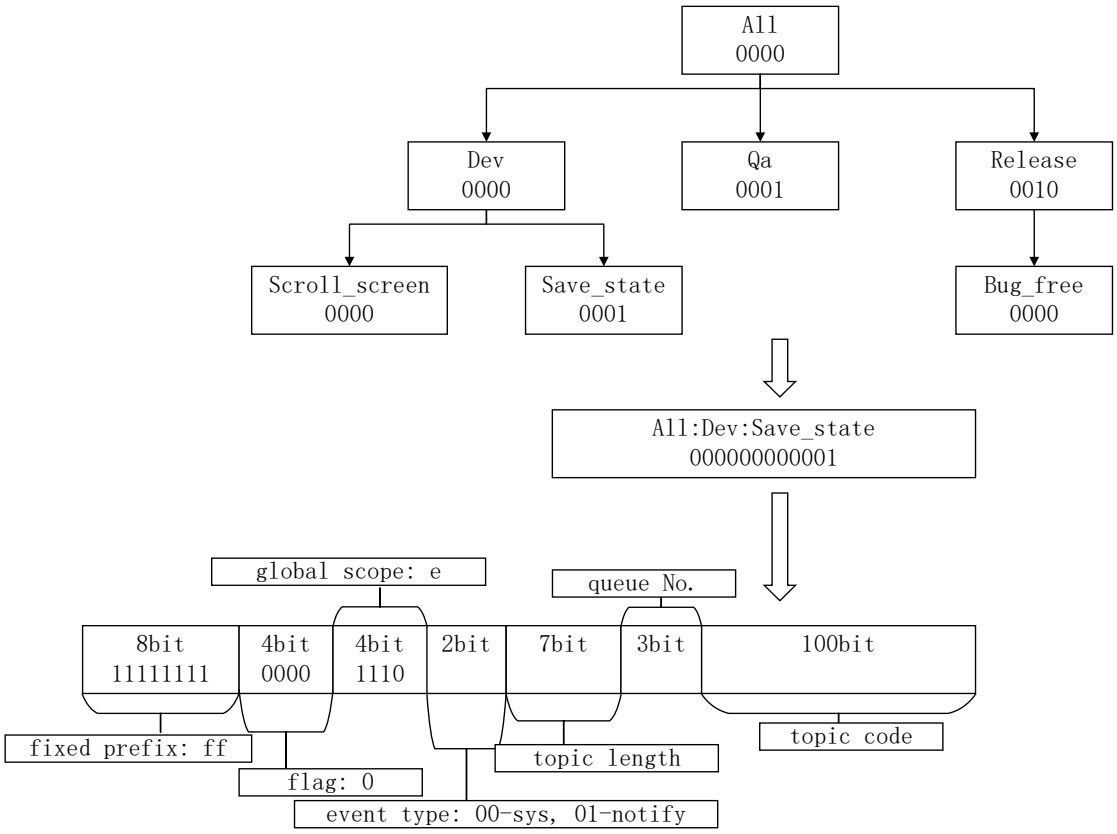

图 4-4 主题树转化为自定义匹配项示意图

在完成主题树的编码后，节点会将主题编码与其他必要的字段拼接，形成 128bit 的 ipv6_dst 字段。而后普通转发类消息的发送和监听都围绕这个 IPv6 地址展开，自定义匹配项每项具体含义如表 4-2 所示。

表 4-2 自定义匹配项结构及含义表

字段	长度	默认值	含义
IPv6 组播固定前缀	8bit	11111111	
标志位	4bit	0000	用于说明组播地址是“永久性（由 IANA 指定的一个地址）”的，还是“临时性”的
范围	4bit	1110	定义组播地址的范围类型，1110 表示该组播地址为全局范围
事件类型	2bit	无	00 代表系统管理类主题，01 代表普通转发类主题
主题长度	7bit	无	主题从主题树编码得到的字符串的长度
队列编号	3bit	000	指定匹配的数据包所要进入的队列，队列定义在端口上
主题编码	100bit	无	将 LDAP 主题树编码得到的二进制字符串

相同的思路，新系统还可以选择 IPv4 组播。但由于 IPv4 地址的长度有限，只有 32bit，算上组播的固定前缀以及事件类型等必要的分类标识，留给主题编码的部分就很少了，并不能很好的覆盖主题树，因此最后新系统还是选择了长度更宽裕的 IPv6 地址。

4.3.2 多级流表匹配设计

OpenFlow 协议中定义的流表项包括匹配域、计数器、指令集等字段，其中匹配域主要负责对消息的头部（Header）进行匹配，确认该数据包是否匹配本条流表项，如果匹配，则应用指令集中的动作（Action），执行入队操作或者进入下一张流表继续匹配；否则就继续向下匹配更低优先级的流表项，直到成功匹配到某一条流表项，或者一直匹配到了流表中最后一条流表项。流表中流表项的基本构造如下图所示。

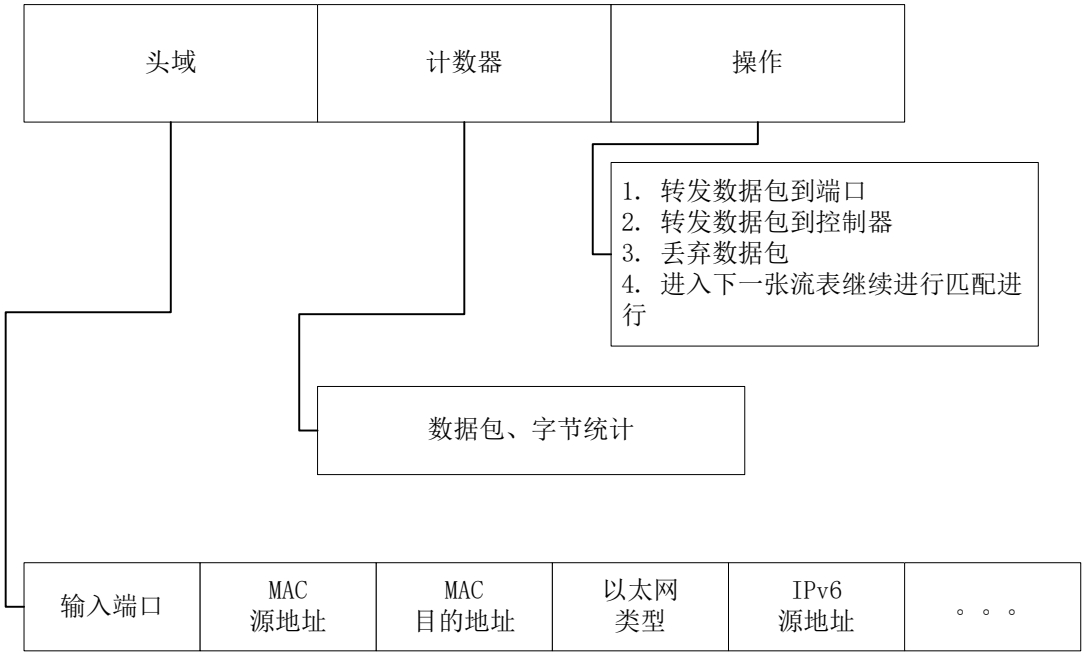

图 4-5 流表项结构

在多级流表中，每张流表的最后一条流表项是节点在进行 OpenFlow 交换机的流表初始化时创建的，它没有特定的匹配项，意味着无论当前进行匹配的数据包有什么具体特征，都会在这个地方被拦截，并且执行其指令集中的动作。如果当前这张流表是 OpenFlow 转发匹配流水线中的最后一张流表的话，那么指令集指定的动作一般是丢包（Drop）；而如果这张流表后面还有其余的流表等待进行匹配的话，那么这最后一条流表项的指令将会是 Goto-Table，并且会指向下一张流表。

在当前设计中，发布/订阅系统共有两级、一共 N+1 张流表（N 为 OpenFlow 交换机的端口数）。#0 流表首先匹配系统管理类消息的主题，详细的系统层级主题及其使用场景如表 4-3 所示。

表 4-3 系统级消息主题表

序号	主题名	使用场景
1	hello	建立邻居关系时发送的探测；收到邻接集群 re_hello 消息后进行回复
2	re_hello	收到 hello 消息后进行回复
3	admin	向管理员请求 LDAP 主题编码树；管理员下发控制指令
4	pub	新产生了发布者

表 4-3 系统级消息主题表（续上表）

序号	主题名	使用场景
5	sub	新产生了订阅者
6	route	同步新计算出的转发路径
7	lsa	发送或同步集群链路信息

在流表中 IPv6 地址这种匹配项是精确匹配，即需要 128bit 每一 bit 都相同。在#0 流表中匹配系统主题无果后，OpenFlow 交换机会继续在#0 号流表中匹配普通转发类消息数据包的进端口，根据端口不同，将数据包提交到#n 流表中继续进行匹配。多级流表结构如图 4-6 所示。

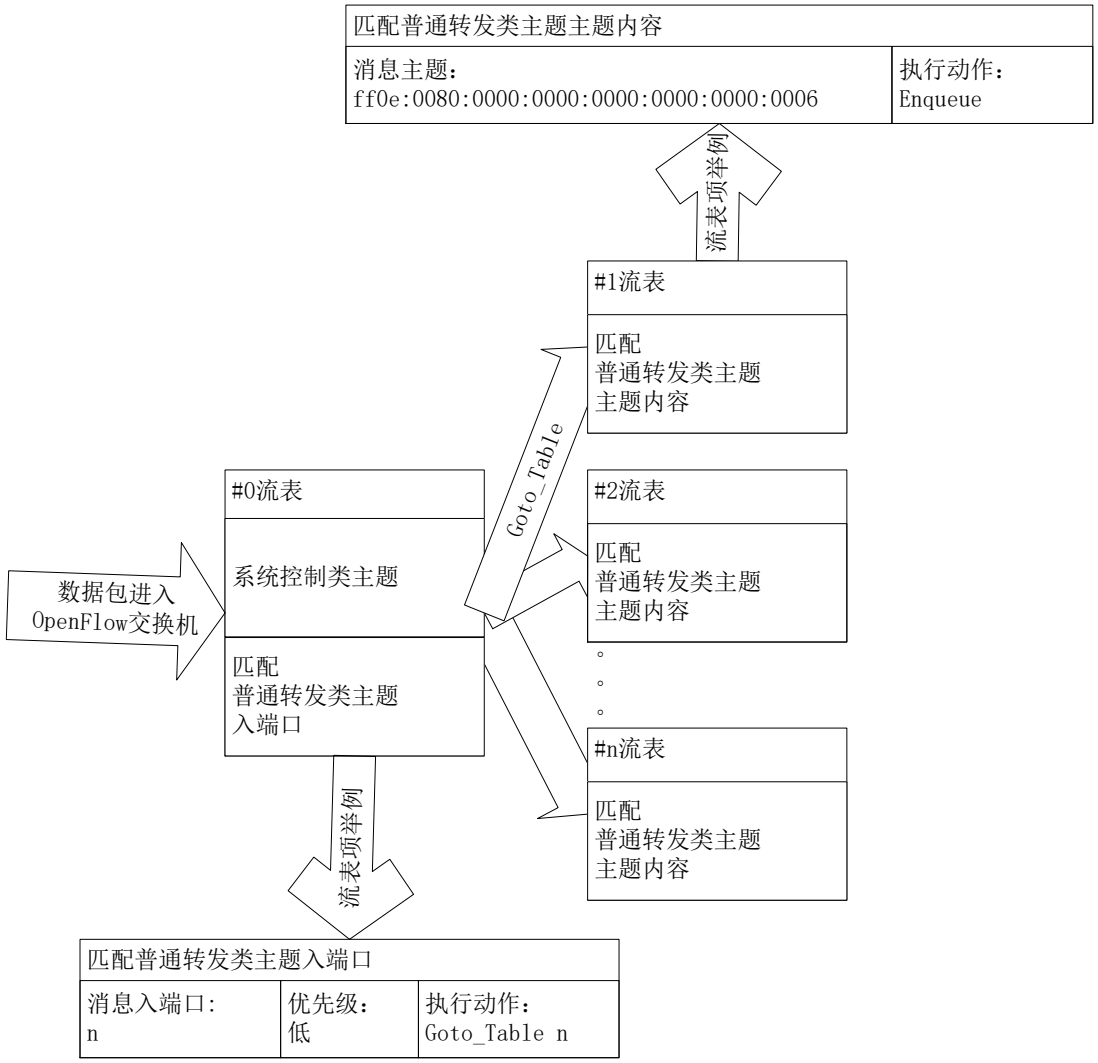

图 4-6 多级流匹配示意图

之所以进行多流表设计，主要目的还是为了提高流表管理的效率。将不同目的的流表项分别放置在不同的流表中，减少了每张流表中的条目数量，也就减少

了匹配的次数，加快了单个数据包的匹配速度，同时增强了流表的可读性，便于后期开发维护。

在数据包流入 OpenFlow 交换机后，数据包首先会匹配这张对应系统管理类主题的流表。如果数据包的 ipv6_dst 字段与系统主题流表中某条流表项的自定义匹配项相同，那么就直接执行对应流表项指令集中的动作，否则就继续向下匹配，找到当前消息的进端口，跳转到对应端口的流表中继续进行消息主题的匹配。

按照 OpenFlow 协议规范中定义的流水线作业原理，新系统完全可以采用通过对写入流表项的优先级进行分类达成目的，但是这样就会使得一张流表中的流表项数量过多，进而影响管理和匹配的效率。同时在将流表由一张分为多张之后，节点中关于流表控制的业务逻辑也会变得更加清晰。

4.4 订阅主题动态管理设计

在基于 SDN 网络的发布/订阅系统中，订阅管理模块需要确保发布表与订阅表的一致性，这样的一致性维护对于后面路由计算具有重要意义，也是消息正确转发的基础。因此本系统采用了订阅者与发布者注册广播机制，同时配合 LSA 周期性同步机制来双重保险，确保订阅信息的一致性。而在系统运行过程中，则通过对订阅主题进行分裂和聚合，动态调整 OpenFlow 交换机上的流表项目以及网络中的流量。

4.4.1 发布者（订阅者）注册

不同于基于传统网络的发布/订阅系统使用 Mina 作为底层消息转发的架构，基于 SDN 网络的新系统在底层使用 OpenFlow 交换机进行消息的转发和推送，因此需要在消息进入发布/订阅消息域之前将路由计算完毕并生成对应流表下发。否则如果消息通过 WebService 接口进入节点时未经注册，那当数据包进入消息域时，就无法正确转发。

因此在任何程序接入发布/订阅系统的消息域之前，都必须向它所在主机上的节点程序进行注册，通知节点自己想要注册的服务类型、对应的主题以及接收或发送消息所使用的地址，这样节点才能够提前计算所需的路由。具体流程如下图所示。

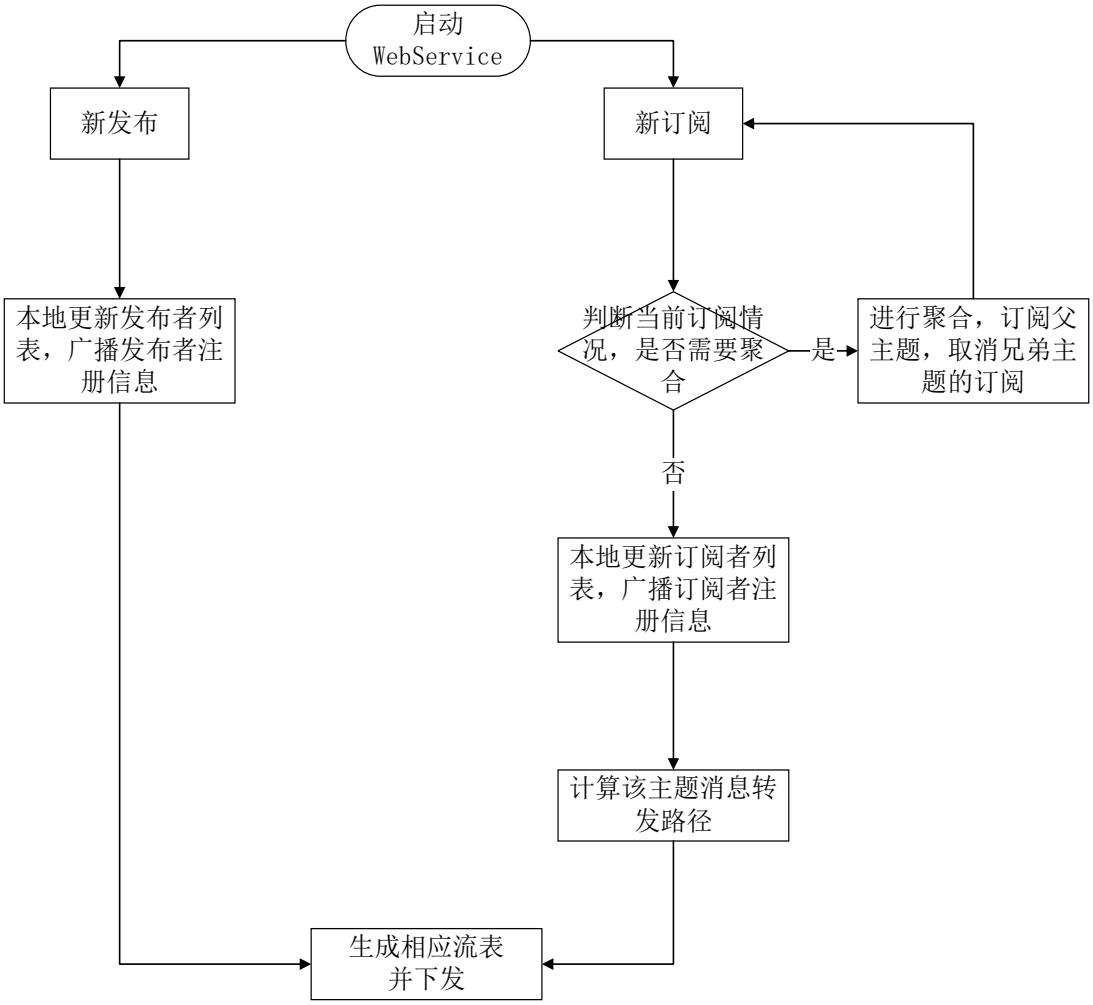

图 4-7 发布/订阅注册流程图

抽象的来看，在发布/订阅系统的眼中，它所处的网络是由集群为基本元素构成的带权无向图，因此在进行广播通知集群订阅状态变化时，需要考虑网络时延或传输过程中集群丢失等情况，避免因此导致的信息更新不一致。借助在集群状态变化消息中添加的信息更新时间以及 LSA 序号，可以提升订阅同步的准确性。

4.4.2 LSA 周期性同步

除了发布者订阅者注册机制外，系统还提供了基于 LSA 消息的定时同步，作为订阅表一致性的第二道保障。

在集群间建立了邻居关系后，两集群会定期与对方交换本地的 LSDB，节点会根据 LSA 信息的内容检查本地的 LSDB 中是否有这个集群的信息，以此来保证本地 LSDB 的正确性。如果之前通过注册者广播机制收到的集群信息有遗漏或者 LSDB 更新不及时，那么在这里节点可以对其进行二次确认。集群间的 LSA 同步如图 4-8 所示。

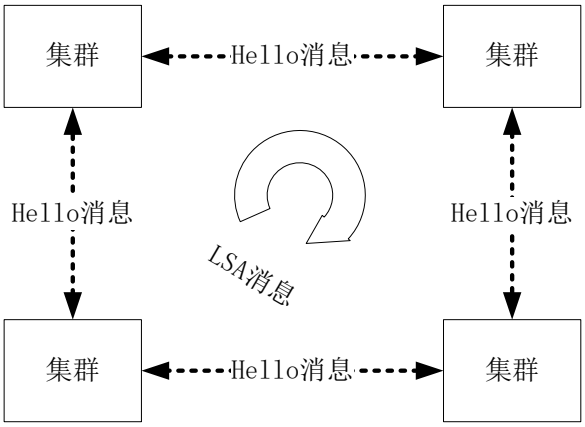

图 4-8 集群间同步 LSA

4.4.3 订阅表分裂与聚合

主题树的层级和每层中主题的数目会随着业务规模的扩大而迅速增长，订阅关系也会变得越来越复杂，将高效匹配与准确转发的压力全部安置在 OpenFlow 交换机上显然是不现实的。随着第二级流表，也就是各端口上普通转发类消息流表项数目越来越多，匹配所需的时间越来越长，最终会直接影响到消息转发的实时性。

因此本文提出了基于 SDN 网络的发布/订阅系统中的订阅表的分裂与聚合。在产生订阅的时候，就会查询本节点上已有的订阅主题，如果已有订阅主题中某个主题的孩子主题数量过半，那么就增加这个父亲主题的订阅，取消父亲主题下面所有孩子主题的订阅，同时将变动保存起来。这样虽然会造成无效流量，但是如果这部分流量所占比例不大的话，那么完全可以交由节点自己消化，而这种聚合带来的匹配速度提升是显而易见的。聚合的流程如图 4-9 所示。

而订阅表的分裂也是动态调整订阅主题的一部分。节点在启动后，定时向集群控制器查询本地集群边界交换机以及控制器交换机的流表匹配情况，若有某个聚合而成的主题在边界交换机上匹配成功次数很多，而这个主题的孩子主题在控制器交换机上匹配成功的次数却很少。那么就意味着大部分边界交换机上的成功匹配，都属于该主题下那些未被订阅的孩子主题，这也就意味着这次聚合产生的订阅结构是不经济的。因此需要取消这个聚合主题的订阅，转而分裂到下面几个之前被订阅的孩子主题上，将它们重新订阅回来。

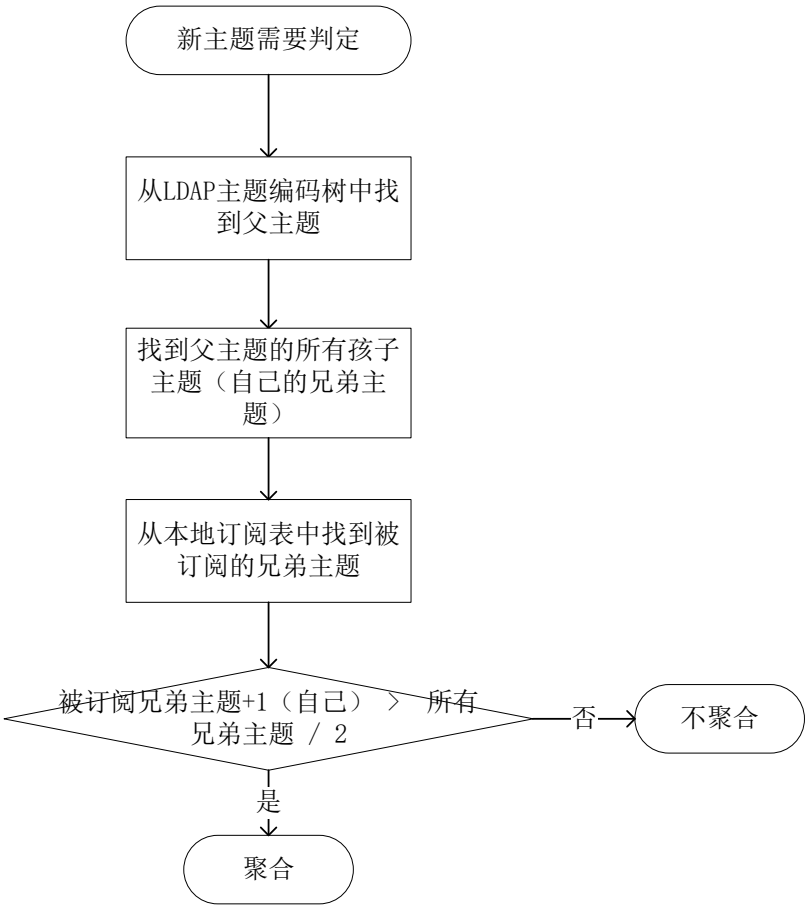

图 4-9 主题聚合流程图

综上，订阅表的聚合目的在于压缩流表的尺寸，加快数据包匹配的效率。而订阅表的分裂目的则在于优化流量分配，防止出现过多无效流量。

4.5 路由算法优化设计

路由算法是用来计算转发路径，为消息提供最优转发通道的。高效的算法以及科学的链路权值设置，是路由算法乃至整个基于 SDN 网络的发布/订阅系统的关键。因此在系统设计之初，本文就将路由模块作为一个独立的系统抽象了出来，它仅提供接口供节点调用，内部的计算逻辑是完全透明的。这样的结构可以保证在后期进行调试时不用大范围的修改代码中的参数。

路由计算的素材来自对应主题的订阅情况，以及全网的拓扑结构，由新的订阅者或者发布者注册事件驱动进行计算。控制器节点收到有集群产生新订阅，那么计算时就首先读取该订阅主题对应的发布订阅树，计算该集群到每个订阅集群的最短路径。这里计算最短路径是使用 SPF 算法，也即 Dijkstra 算法进行计算的，其中的链路权重使用 LSDB 中保存的链路距离，此距离是综合考虑了物理距离、链路带宽以及当前链路中的流量得出的。由于该数据变化较快，因此每次在进行

LSA 同步时都会将它更新后再进行传输，确保所有集群保存的数据是一致的。其总体流程图如图 4-10 所示。

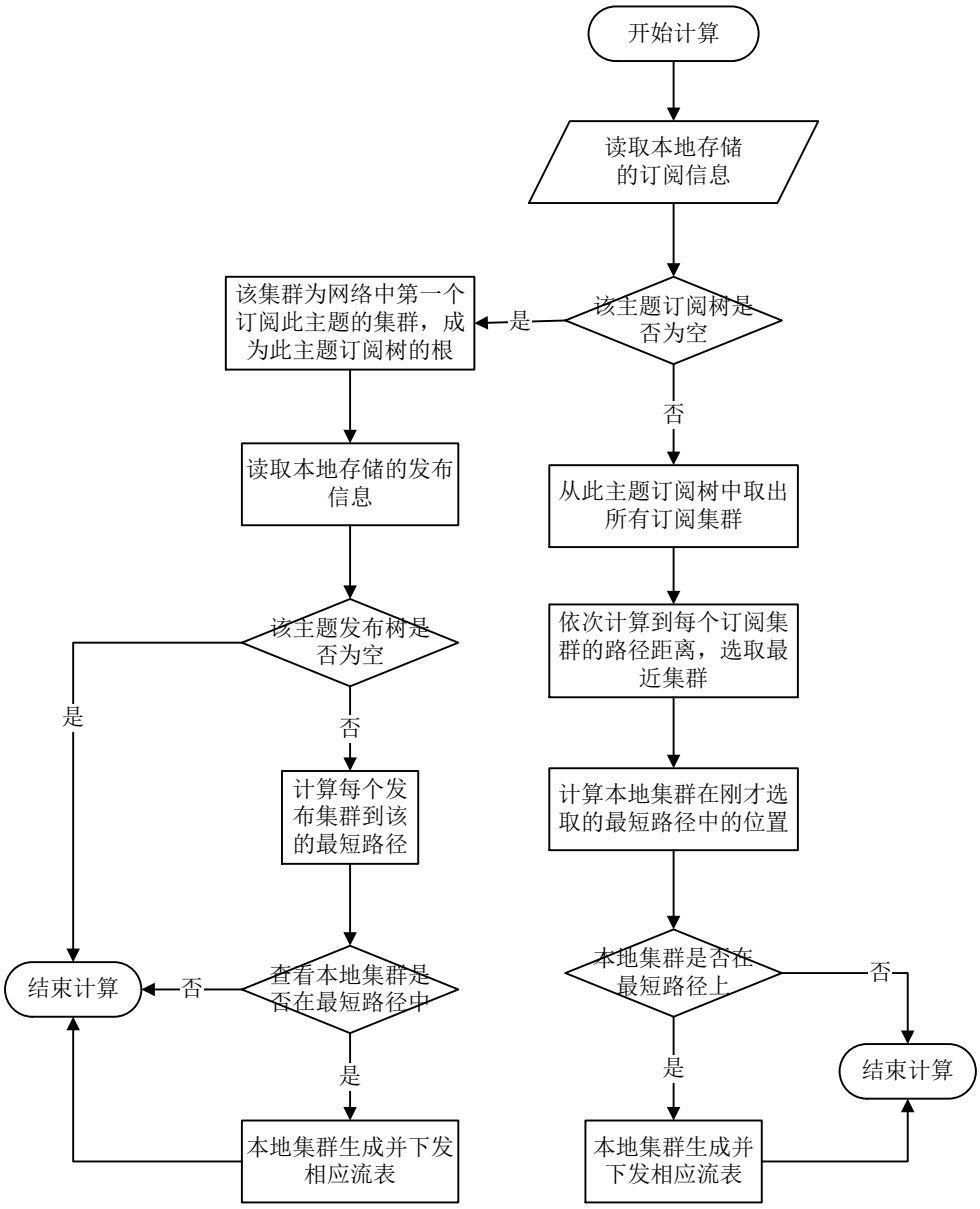

图 4-10 路由计算流程

路由计算时，节点首先会查看当前订阅表与发布表，订阅表与发布表由两部分组成，一部分是自身或者本集群内其他节点产生的发布和订阅信息，另一部分是节点收到的全网广播注册信息，或者其他集群通过 LSA 信息同步发送来的注册信息。如图 4-11 所示是全网集群拓扑简图，标明了集群间的连接情况和集群间链路的权值（综合考虑了网络带宽情况以及实时流量等数据得出的距离值）。

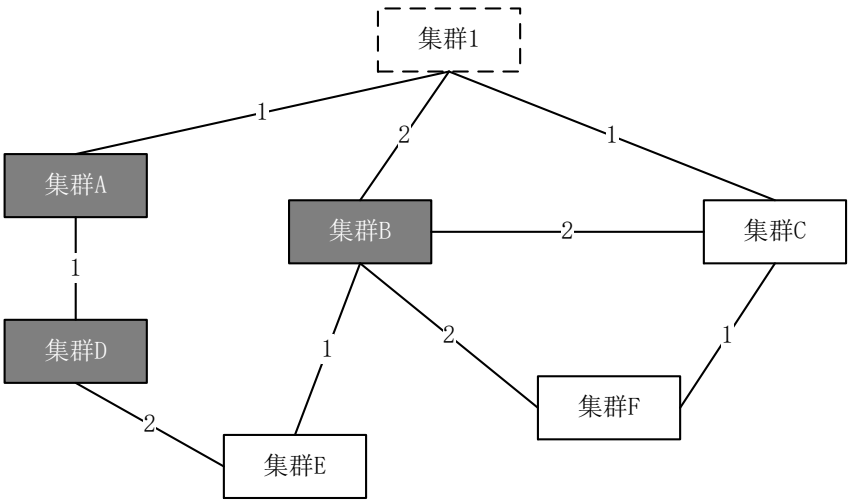

图 4-11 计算路由树

在收到新订阅集群 1 注册信息的时候，本地集群首先从本地找到对应主题的发布订阅树，可知集群 A、集群 B 以及集群 D 组成订阅树。而集群 1 到三者的最短距离分别为 1、2 和 2，因此选择集群 A 进行连接。再判断本地集群是否在这个路径上，因为不在，所以计算结束。

注意这里的连接是抽象的直连，在这路径上可能涉及既无订阅也无发布的过路集群，这些集群也需要进行计算并下发自己的流表。而如果是新发布产生，则要计算新发布集群到发布订阅树上所有集群的最短距离，向最近的节点下发单向流表。

在计算出当前主题的最短转发路径后，节点会查看当前主题及其父亲主题的流表项。这里不需要向控制器发起请求，因为在每次下发流表后，节点都会在集群内将下发过的流表项进行同步。这种机制保证了在集群控制器失效的时候，新控制器可以快速的恢复之前的流表状态。在获取到父亲主题的流表项后，从中得到父亲主题转发时对应的 OpenFlow 交换机的 ID 以及端口号，将其拼接在刚才计算出来的结果后面。如果涉及到多个 OpenFlow 交换机，则将其分置在多条流表项中。

下发相应主题的流表项时，还需要指定其流表编号以及其优先级，本文在 4.3.1 小节中已经对多流表设计进行了说明，系统将普通转发类消息对应的流表项，根据消息进端口不同，插入不同的第二级流表。而针对优先级的设计，则是将优先级与主题所在的层级对应起来，层级越多的主题对应的流表项优先级越高，而优先级越高的流表项对应的转发方向也越多，这样就实现了主题从 LDAP 中存储的树形结构到 OpenFlow 交换机流表中的线性结构的转换。

最终，在节点算出最短路径并生成了与本集群相关的流表项之后，就会将这

段路径对应的流表项进行下发，一般而言每次计算会产生多条流表项，每条流表项对应一台 OpenFlow 交换机设备。生成流表的流程如下图所示。

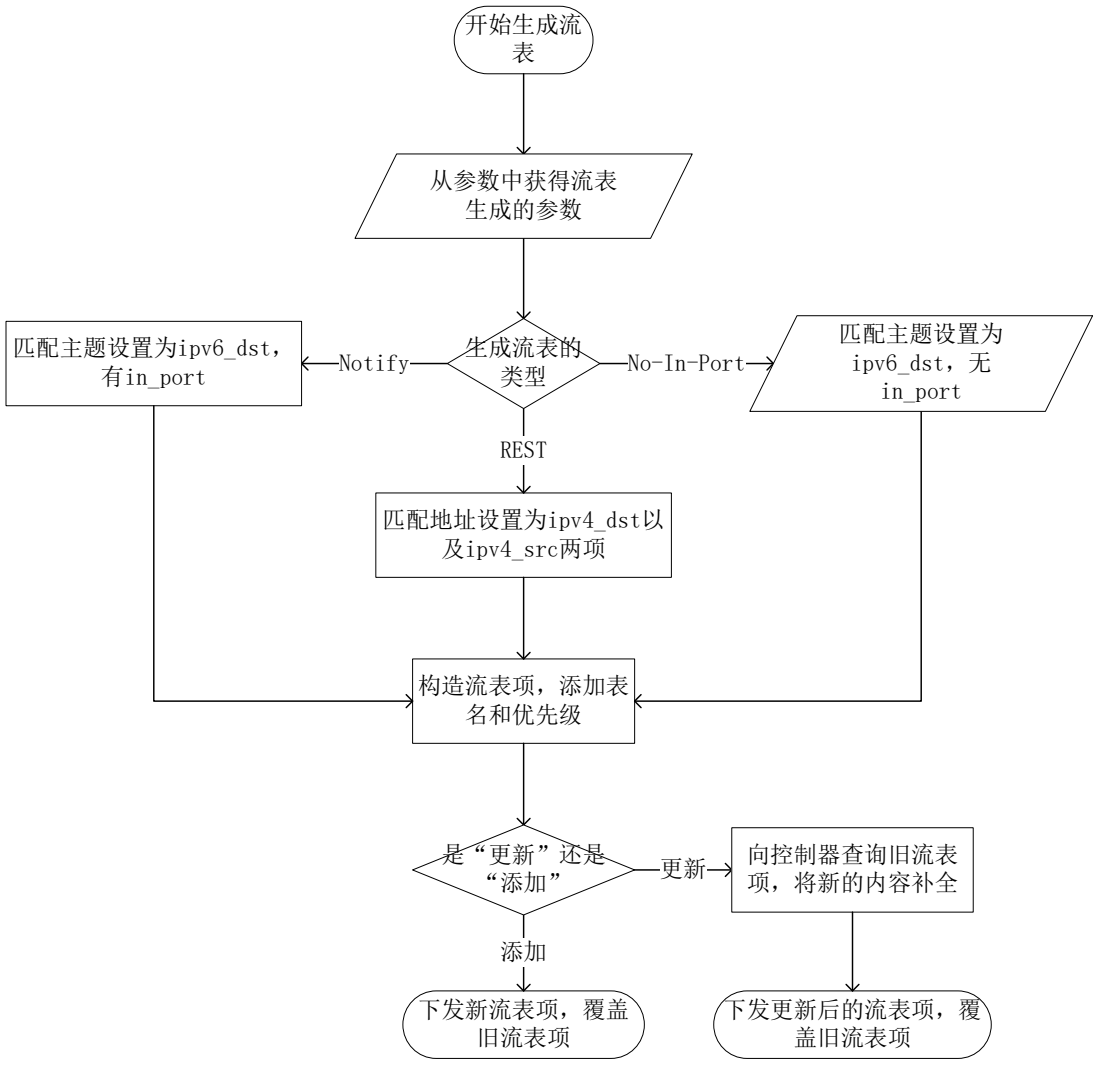

图 4-12 流表生成流程图

在路由计算完成后，节点还需要定时检查流表中涉及到的每个端口的实时匹配情况。一方面通过流量情况来决定是否需要将这条流表项对应的订阅主题分裂为若干孩子主题，另一方面也可以为流表动态调整提供依据。对路由的动态调整需要结合两部分数据共同决策，其一是随全网同步的 LSA 信息中的邻居距离，其二就是刚才提到的端口上的流表项实时匹配情况，若实时流量统计的结果超过路由更新的阈值，则会激活函数重新计算路由。结合这两部分来更新路由并下发新的流表项。路由计算配合流表下发的全过程如图 4-13 所示。

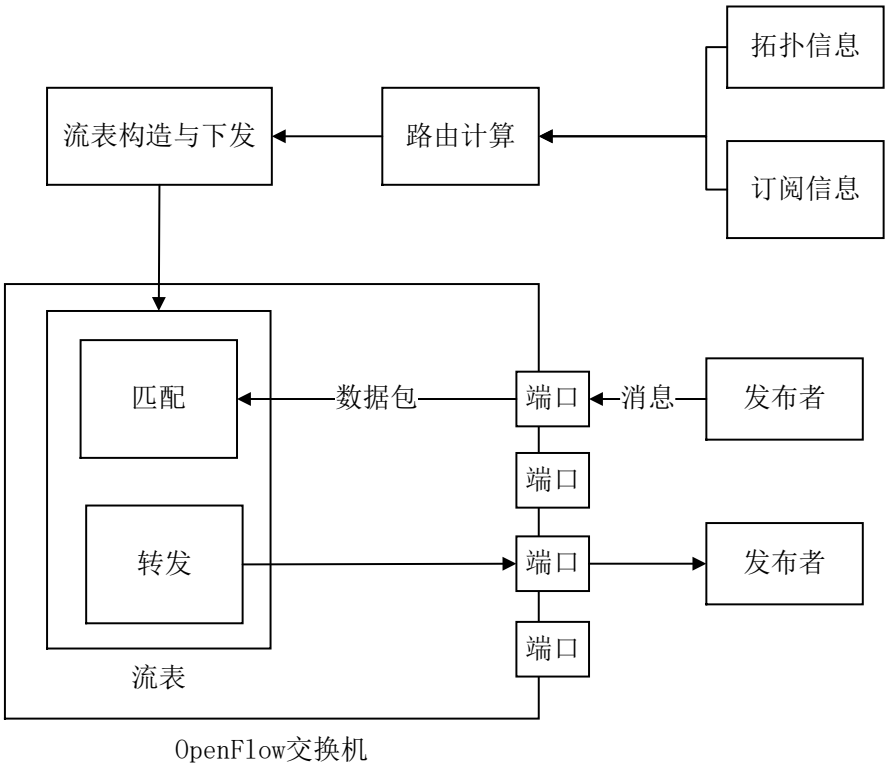

图 4-13 路由计算与流表下发全过程示意图

集群内的消息通知相对简单，只需要在集群内进行相关主题消息的组播推送既可，这样一来是减轻了节点的计算压力，只需要计算集群间转发路径，而不需要计算集群内的推送路径，二来也方便控制集群内 OpenFlow 交换机中流表的管理，如果集群内节点上有需要订阅的消息，那么节点会开启相关主题消息的监听，进行群内推送消息的接收。

4.6 本章总结

本章主要介绍了基于 SDN 网络的发布/订阅系统中的核心模块的架构设计，其中主要包括了新系统的拓扑管理、多级流表、订阅动态管理以及分布式路由算法的设计与优化。同时根据新系统中各模块的应用场景，分析了采取当前设计方案的原因和预期的效果。

第五章 路由及相关模块的实现

本章主要介绍基于 SDN 网络的发布/订阅系统的拓扑管理、订阅管理以及路由计算三个核心模块的具体实现，这三部分相互协作，保证了消息实时准确的传输。

5.1 拓扑构建与维护的实现

拓扑管理模块实现的功能主要包括集群内网络节点（节点程序所在的主机 Host 以及 OpenFlow 交换机）的加入删除，以及邻居集群的构建维护。它们能帮助系统在任意节点上获得并维护统一的全网拓扑信息，为管理员的展示以及路由的计算打好基础。下面就分别介绍集群内以及集群间拓扑管理的具体实现过程。

5.1.1 集群内节点的加入

集群内的网络拓扑主要通过查询本集群的集群控制器来进行获取。当节点启动时，节点首先会读取配置文件 RtConfig.properties 以及 DtConfig.properties，确定一些参数，具体配置项及其含义见下表。

表 5-1 配置文件说明表

配置项	说明
adminAddress	管理员的 IP
localSwtId	本地连接交换机的 OpenFlow ID
groupCtl	集群控制器的 IP
localGroupName	所在集群名称
localAddress	本地 IP
tPort	TCP 消息监听端口
notifyPort	普通转发类消息监听端口
helloTaskPeriod	邻居探测定时任务启动时间间隔
nbrGrpExpiration	邻居集群丢失时间判定阈值
refreshPeriod	集群内拓扑探测任务时间间隔
checkSplitPeriod	分裂任务执行时间间隔

在配置完成后，节点会调用 downRestFlow()函数下发一条流表项，其匹配项为 ipv4_dst，匹配值为集群控制器地址，通路为全网组播。通过这条流表项，节点能够顺利将 REST 请求发送到集群控制器。之后节点会启动对集群控制器的轮询任务，通过向 OpenDaylight 控制器所提供的 REST 接口 http://<controller:port>/restconf/operational/network-topology:network-topology/ 发送 GET 请求进行查询，得到名为 network-topology 的 JSON 对象。根据其含义不同，分类保存为 Switch 以及 Host，这两个类和它们的基类设计如下表所示.

表 5-2 DevInfo 基类设计

类名：DevInfo		
属性名	类型	说明
mac	String	设备 Mac 地址
neighbors	Map<String, DevInfo>	设备的邻居。key 是端口号，value 是设备
subMap	Map<String, Set<String>>	本集群的订阅信息。key 是订阅的主题名，value 是形如“swtId:port”的 String 集合
pubMap	Map<String, Set<String>>	本集群的发布信息。key 是发布的主题名，value 是形如“swtId:port”的 String 集合

表 5-3 Switch 类设计

类名：Switch		
属性名	类型	说明
portSet	Set<String>	LOCAL 端口和 outPort 的集合
id	String	交换机的 OpenFlow ID
queues	Map<String, List<Queue>>	该交换机每个端口上的队列。key 是端口号，value 是这个端口上的队列

表 5-4 Host 类设计

类名：Host		
属性名	类型	说明
ip	String	主机的 IP

表 5-4 Host 类设计（续上表）

属性名	类型	说明
swtId	String	主机所连接的 OpenFlow 交换机的 ID
port	String	OpenFlow 交换机上和这个主机连接的端口

从控制器获得的 JSON 对象中的主要内容结构如表 5-5 所示。

表 5-5 OpenDaylight 拓扑信息表

```
<topology>
  <topology-id>flow:1</topology-id>
  <node>
    <node-id>openflow:139329991887426</node-id> // Switch 中的 id
    <termination-point>
      <tp-id>openflow:139329991887426:1</tp-id> // 先放到 Switch 的 portSet 中，后面根据<link>取出再存到 neighbors 中
    </termination-point>
  </node>
  <link>// 保存到 Switch 的 neighbors 中，key 是<source-tp>的端口号，value 是<dest-node>中对应的 Switch
    <link-id>openflow:108132035398724:1</link-id>
    <source>
      <source-node>openflow:108132035398724</source-node>
      <source-tp>openflow:108132035398724:1</source-tp>
    </source>
    <destination>
      <dest-tp>openflow:20601728700235:1</dest-tp>
      <dest-node>openflow:20601728700235</dest-node>
    </destination>
  </link>
</topology>
```

根据各部分的内容不同，节点将所获得的数据进行抽象，保存到 hostMap、

switchMap、outSwitches（在内存中保存所有拥有 outPort 的 OpenFlow 交换机实例）等对象中，保存过程的时序图如下图所示。

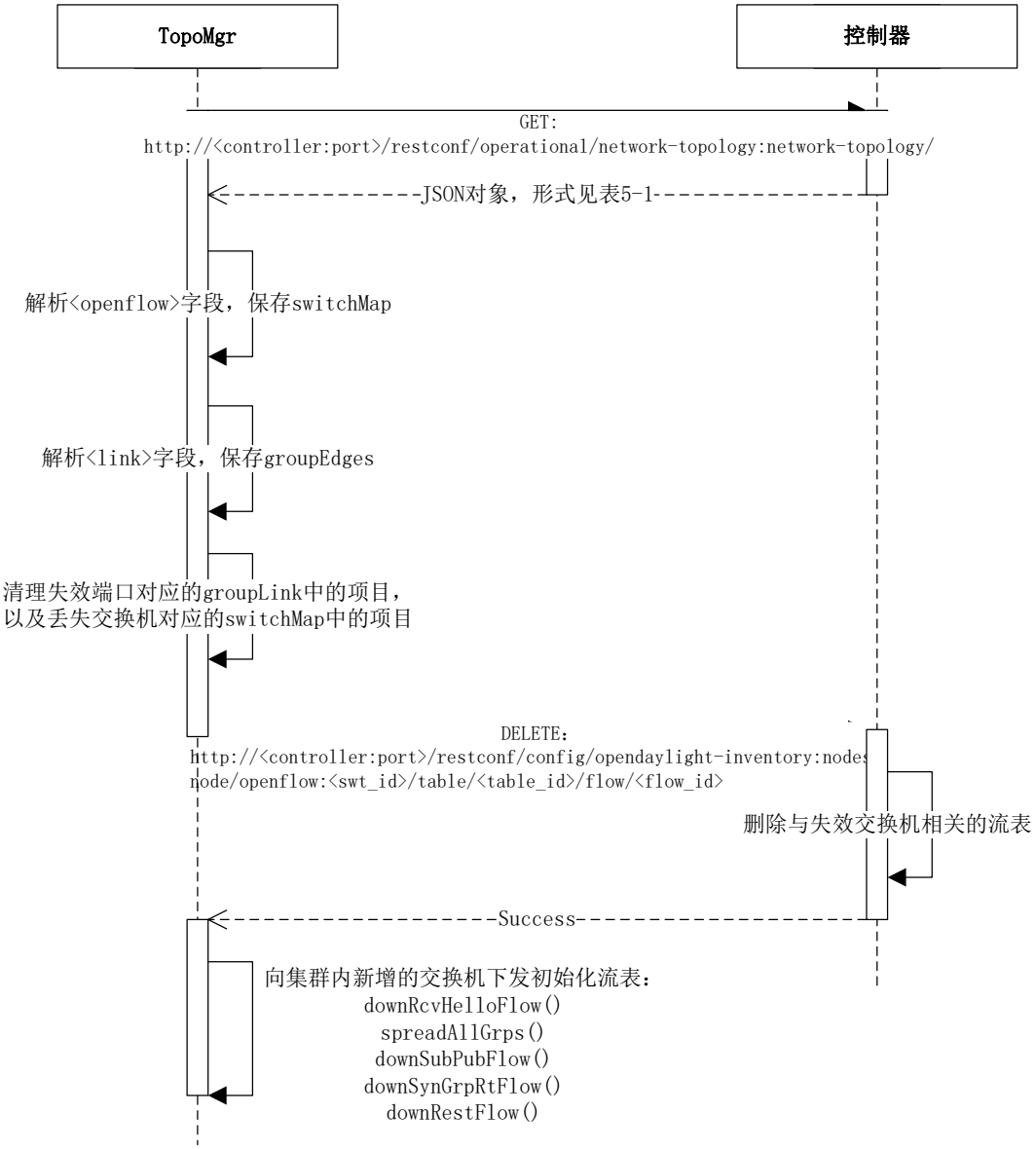

图 5-1 集群内拓扑获取时序图

节点会将每个 OpenFlow 交换机的端口情况进行分类，一类是连接集群内部其他 OpenFlow 交换机以及主机的端口，另一类是连接集群外其他交换机的端口（即 outPort）。之后节点会在所有拥有第一类端口的 OpenFlow 交换机上下发初始化流表项，初始化流表项所匹配的主题包括“admin”、“pub”、“sub”、“route”以及“lsa”几类，它们都属于系统管理类主题，在这里下发是为了方便后面订阅管理以及链路状态的传播。

5.1.2 集群内节点的删除

集群内节点删除所用到的类，与集群内节点增加的类基本相同。节点在运行中持续执行 RefreshGroup 任务轮询集群控制器，在 OpenDaylight 控制器运行过程中集群如果出现 OpenFlow 交换机掉线、主机掉线等情况，OpenDaylight 控制器会第一时间更新自己存储的信息。而节点也会通过 getGrpTopo()函数从控制器获取到包含集群内拓扑情况的 JSON 对象，解析后就可以对之前提到的一系列保存在内存中的对象进行覆盖更新即可。

同样，在更新过 GroupTopo 之后，RefreshGroup 任务也会重新下发流表，同时对比之前下发的流表，将失效的部分删除，执行这些动作的函数有下发接收 Hello 消息流表的 downRcvHelloFlow()、下发同步发布/订阅消息流表的 downSubPubFlow()、下发同步集群内计算得到的路径的流表的 downSynGrpRtFlow()以及下发支持 REST 请求的流表的 downRestFlow()。执行这一系列操作的 GroupUtil 类的类图如下图所示。

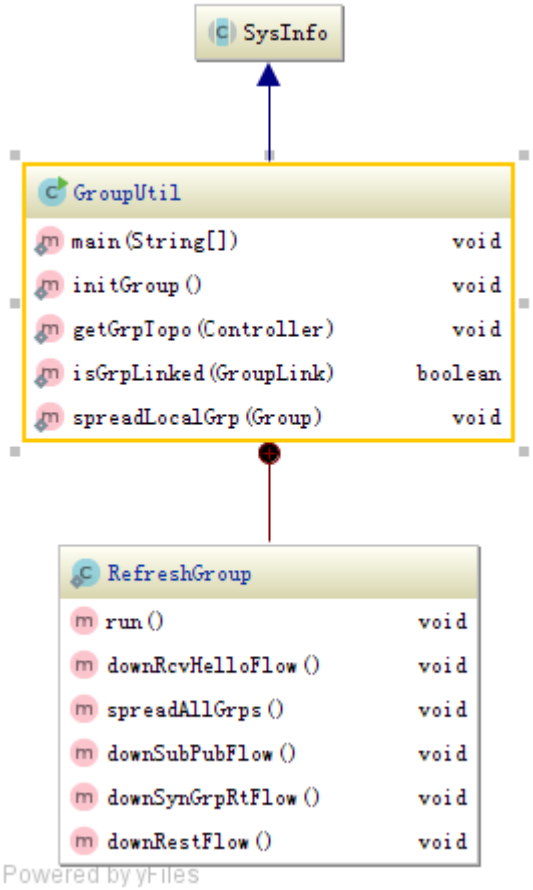

图 5-2 GroupUtil 类图

相比之前基于传统网络的发布/订阅系统，新系统依靠 OpenDaylight 控制器

对集群内的拓扑进行管理，在减少业务层级的同时，也使得处理逻辑更加清晰。

5.1.3 新增集群

普通转发类消息需要在几个集群之间进行传送，依赖于产生消息的节点对全网拓扑的掌握。对于全连通的网络，节点可以直接选择一条链路情况最优的路径调用 MultiHandler.v6Send()函数进行传送。然而对于不全连通的网络，就需要调用 RouteUtil 中的 calNetworkRoute()函数进行路由计算，并选择出一条需要经过若干非订阅集群跳转的路径来进行转发。

要做到这一点，首先就需要集群能够感知并与一个新集群建立邻居关系，这样后面才能够计算出消息在集群间的转发路径。相关类图如下三图所示。

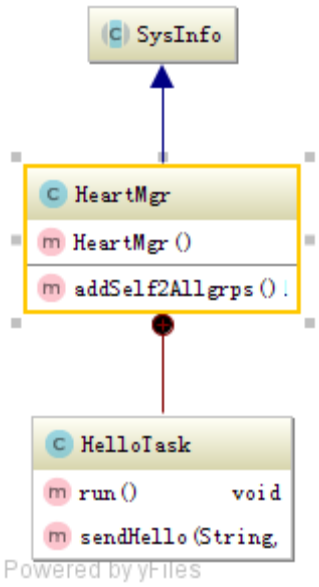

图 5-3 HeartMgr 类图

其中 sendHello()函数用来发送 Hello 消息到 outPort 上，而 addSelf2Allgrps 则是将本地集群的 LSA 更新 updateTime 属性后，覆盖 LSDB 中的旧数据。

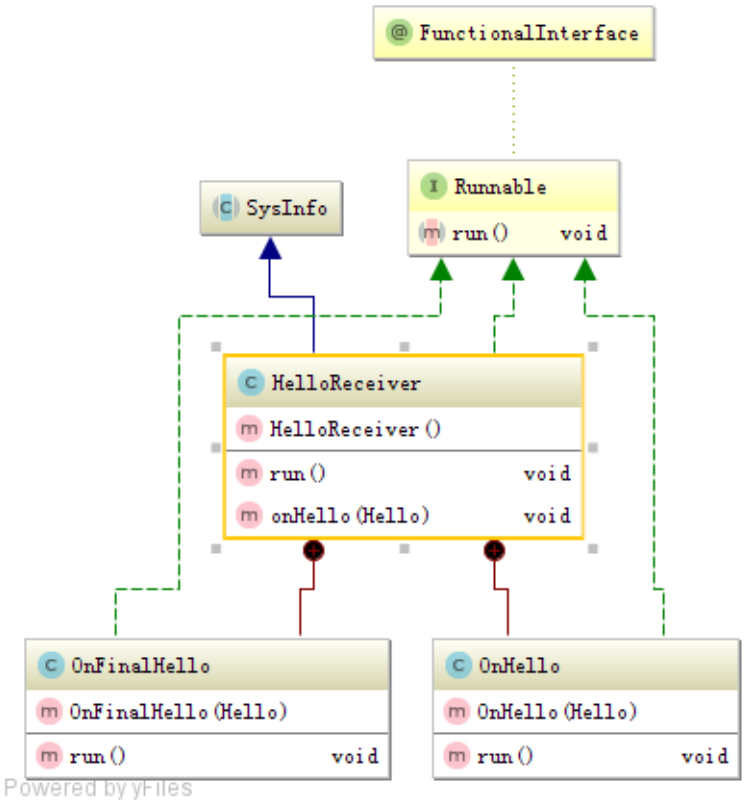

图 5-4 HelloReceiver 类图

在 HelloReceiver 线程收到 Hello 主题的消息后，会调用 onHello()函数，该函数会根据收到消息是否存有 endGroup 值来选择 OnFinalHello 和 OnHello 两个线程进行处理。OnHello 线程处理第一次握手 Hello 消息，OnFinalHello 线程处理第三次握手 FinalHello 消息。

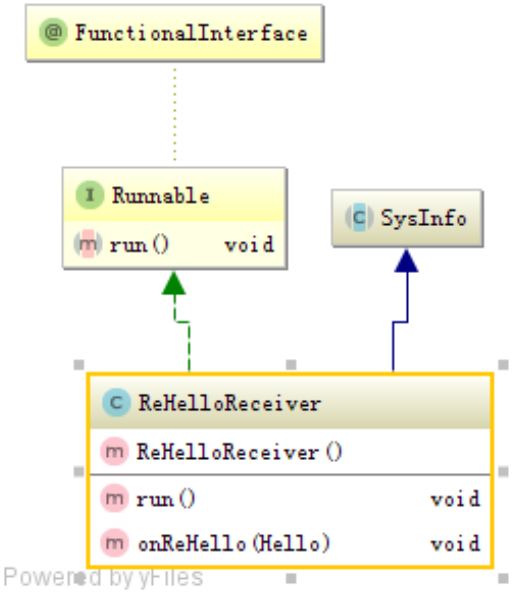

图 5-5 ReHelloReceiver 类图

ReHelloReceiver 收到 Rehello 主题的消息后，会调用 onReHello()函数进行处理。

由于不同集群的 OpenFlow 交换机由不同 OpenDaylight 控制器进行管理，因此节点无法直接确定某个 outPort 所连接的集群名称以及其邻居信息。所以在 HeartMgr 获取到本集群内所有 OpenFlow 交换机的 outPort 信息后，它会启动 HelloTask 依次从 outSwitches 中选择一个交换机实例（下称 outSwitch），调用 RouteUtil 中的 calRoute()函数，利用 Dijkstra 算法来计算从控制器节点所在的交换机到 outSwitch 之间的最短通路。之后对这个方向上沿途的 OpenFlow 交换机下发主题是“hello”的流表项，同时沿反方向对这些 OpenFlow 交换机下发“re_hello”主题的流表项。

在确定流表项下发成功后，HelloTask 调用 SendHello()开始向这个 outPort 发送 Hello 消息，Hello 消息类设计如下表所示。

表 5-6 Hello 类设计

类名：Hello		
属性名	类型	说明
startGroup	String	本地集群名称
endGroup	String	对面的集群名称
startBorderSwtId	String	消息发起方边界的 swtId
endBorderSwtId	String	本消息对标对面的边界 swtId
startOutPort	String	记录这条消息是从哪个端口发出去的
endOutPort	String	记录这条消息对标对面的哪个端口
reHelloPeriod	long	判定节点失效的时间间隔
allGroups	Map<String, Group>	LSDB

对面集群的控制器节点的 HelloReceiver 收到这条消息后，会调用 onHello()函数判断该消息是 Hello 消息还是 FinalHello 消息，确定是 Hello 消息后启动 OnHello 线程进行处理。由于同样无法判断该 Hello 消息是由本集群的哪一个 outPort 发进来的，因此 OnHello 任务会对本集群所有的 outPort 进行遍历，分别计算从自己与本地交换机相连的端口（下称 port2wsn）到选中的 outPort 的最短路径，并对这个端口发送 ReHello 消息。这个消息重新封装了之前收到的 Hello 消息的内容，同时添加了本地集群的名称，当前发送消息所用的 outSwitch 的 ID

以及 outPort 的端口号。

这样，在初始发送 Hello 消息的控制器节点中，ReHelloReceiver 会再次收到 ReHello 消息，就能确定一条本集群与邻居集群之间的链路。本集群会将这条成功创建的链路信息保存到本地，同时在 ReHelloReceiver 类中的 onReHello()函数中重新封装一份 FinalHello 消息发给邻居集群，确保邻居集群的 HelloReceiver 类也得到这条信息，同时在 OnFinalHello 任务中将链路信息保存。邻居构建的全过程如图 5-6 所示。

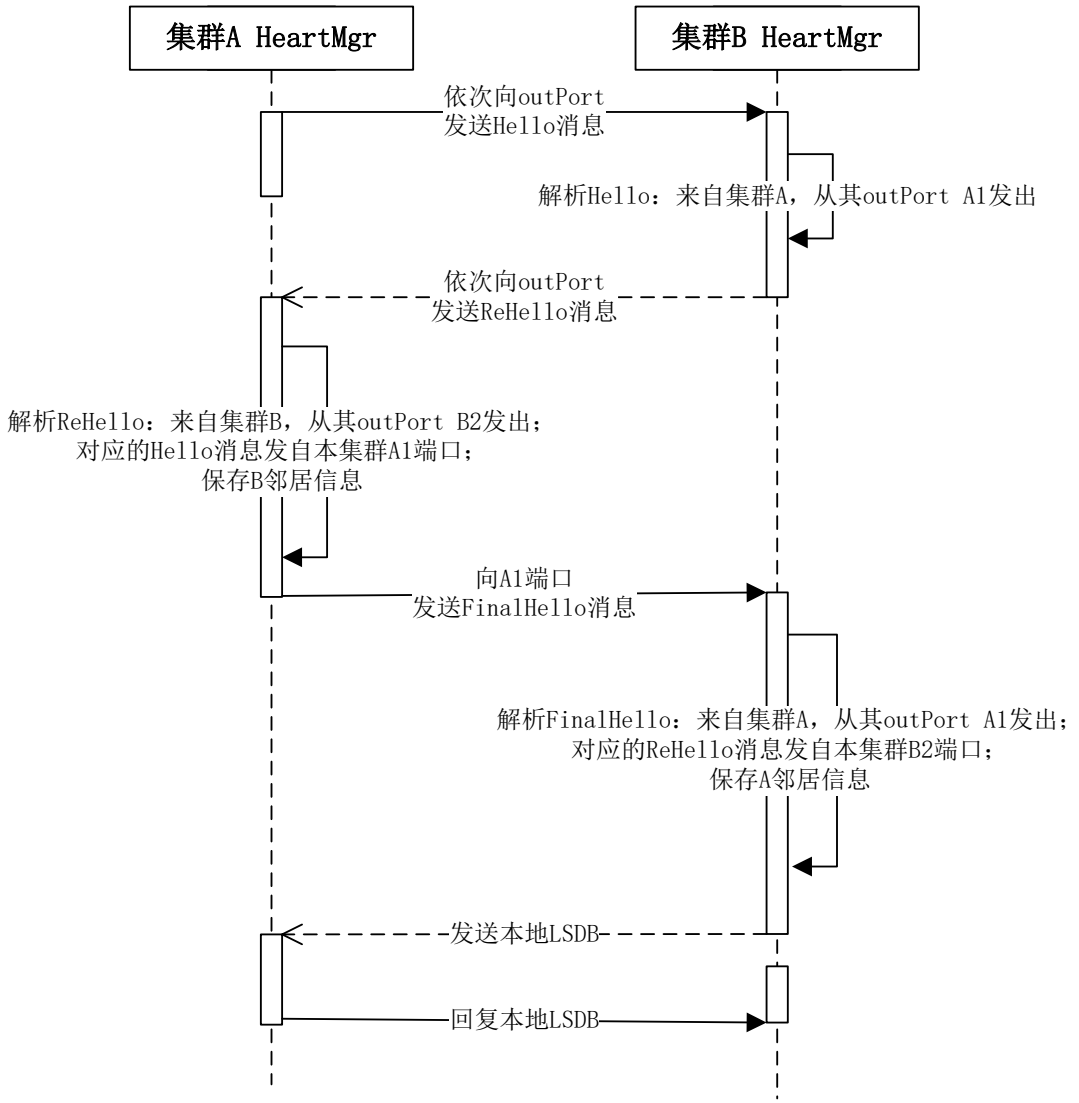

图 5-6 邻居关系构建原理时序图

至此邻居的感知工作就完成了，这之后两个集群还会交换本地存储的 LSDB，每条 LSA 中都包含了该信息的唯一序号和更新时间，确保集群保存了对方最新的链路状态信息。

5.1.4 邻居集群维护

集群在与邻接的集群建立起了邻居关系后，需要 HeartMgr 继续定时的发送 Hello 消息来检测邻居集群是否还在线，如果超过掉线判断时延的阈值，那么就判定这个邻居集群已经掉线，需要从本地保存的链路信息中将这个邻居集群删除。链路信息类表如下表所示。

表 5-7 GroupLink 类设计

类名：GroupLink		
属性名	类型	说明
srcGroupName	String	链路起始方的集群名
dstGroupName	String	链路终止方的集群名
srcBorderSwtId	String	链路中起始集群里的边界交换机
srcOutPort	String	起始集群的边界交换机使用的端口
dstBorderSwtId	String	链路中终止集群里的边界交换机
dstOutPort	String	终止集群的边界交换机使用的端口

这个检测过程中涉及到的发送 Hello 消息的定时任务所用的间隔时间，还有等待邻居集群回复的时间，都可以在配置文件 DtConfig.properties 中进行设置。在维护邻居集群的过程中，控制器节点以间隔 helloTaskPeriod 来运行 HeartMgr 类中的 HelloTask 定时任务，该任务的 run()函数会读取当前节点保存的全集群 outSwitch 信息，选择集群内最短的 Hello 路径来发送 Hello 消息。通过检测结果来更新全网集群信息 allGroups，之后再读取 allGroups 中每个集群最近一次更新的时间 updateTime，确认此集群更新时间与当前时间之差小于集群失效的时间差阈值 nbrGrpExpiration，否则就要将该集群及其对应的邻居链路从 allGroups 和 nbrGrpLinks 中删除掉。

5.2 订阅管理

基于 SDN 网络的新版发布/订阅系统针对订阅管理，主要做了三部分改进：其一是对 LDAP 主题树进行编码，将主题树从树形结构编码成二进制字符串，进而将主题编码编入自定义匹配字段 ipv6_dst 的 128bit 中；其二是在订阅与发布的产生与维护过程中，加入了注册与周期性同步的双重确认机制；其三是对订阅表进行动态处理，对满足条件的订阅主题进行分裂和聚合，提高订阅管理效率。

5.2.1 主题树的编码

由于在消息进行转发的时候需要进行流表项的匹配，因此需要对消息主题进行编码后拼接到 ipv6_dst 主题中，其核心代码如下表所示。

表 5-8 主题树编码核心代码

变量说明：

topicTree：LDAP 中获取的主题树的根元素

NotifyTopicCodeMap：节点中保存的普通转发类主题及其对应的编码

```
private void encodeTopicTree() {
    queue q;
    q.enqueue(newTopicTree);
    while (!q.isEmpty()) {
        t = q.poll();
        if (t .getChildrens().size() != 0)
            q.addAll(t.getChildrens());

        tp = "";
        spt = t.getTopicPath().split(",");
        for (i = spt.length - 3 --> 0)
            tp += ":" + spt[i]; //  将主题树中主题化成 all:a:b 的形式

        code = "1";
        if (t.getParent() != null) {
            // 父主题的编码 + t 序列中序号对应的二进制编码
            code = NotifyTopicCodeMap.get(t.fatherTopicCode)
                    + Integer.toString(t.getChildrens().indexOf(t) + 1);
        }
        NotifyTopicCodeMap.put(tp,code);
    }
}
```

在 NotifyTopicCodeMap 中保存的是主题树的编码，长度为 100bit，节点在收

到主题树编码后，还需要将其拼接上 28bit 的控制码，其逻辑如下表所示。

表 5-9 ipv6_dst 匹配域生成核心逻辑

变量说明： NotifyTopicCodeMap：节点中保存的普通转发类主题及其对应的编码 NotifyTopicMap：节点中保存的普通转发类主题及其对应的 IPv6 地址

public static void initNotifyTopicMap() {
for(topicCode in NotifyTopicCodeMap) {
String binStr = "11111111"//prefix ff 8bit
+ "0000"//flag 0 4bit
+ "1110"//global_scope e 4bit
+ "00"//event_type sys 00 2bit
+ "0000001"//topic_length 7 7bit
+ "000"//queue_NO 1 3bit
+ topic(value);//topic_code 100bit
NotifyTopicMap.put(topicName, topicAddr);
}
}

5.2.2 发布和订阅过程的实现

为计算出某一订阅主题正确的转发路径，节点需要对全网的订阅表进行同步，确保每个集群内、每个节点中保存的订阅表是一致的，这个过程主要是通过 LSA 信息的同步以及发布者注册者的注册来实现。新增发布者和新增订阅者在注册流程中是一致的，因此这里以本地新增订阅者为例进行说明。

如果有用户想要调用发布/订阅系统进行有主题消息的订阅时，首先需要通过节点暴露的 WebService 服务接口，向节点进行注册，通报自己想要注册的角色（SUB）。WebService 相关类图如下图所示。

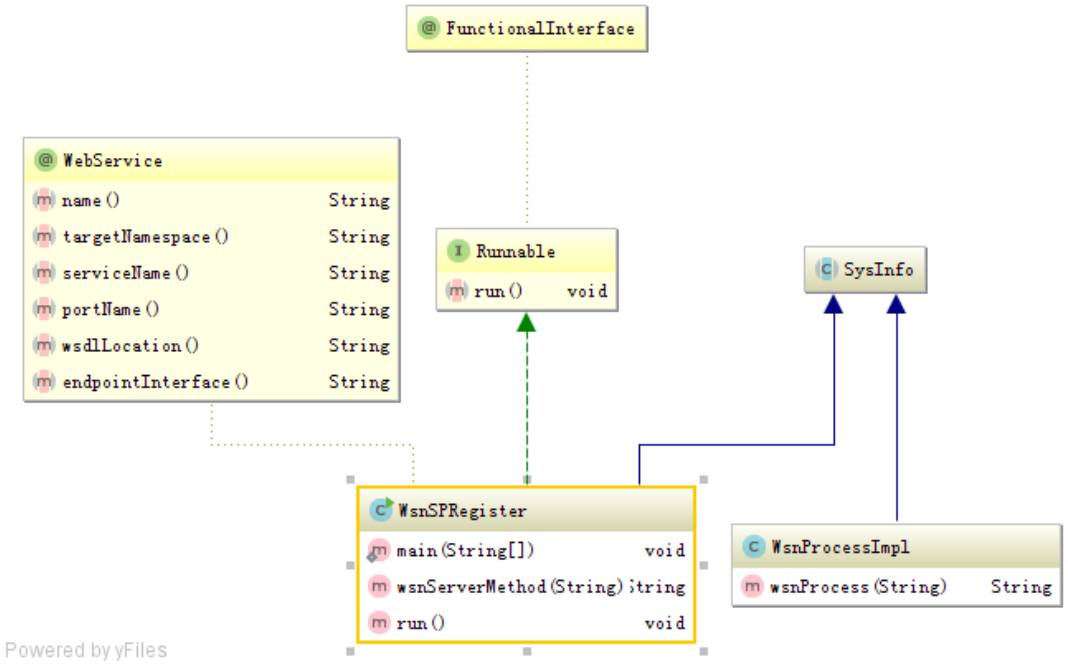

图 5-7 WebService 相关类图

在 WsnProcess()中调用 SubPubMgr.localSubscribe()函数通知节点，节点收到注册信息后，查看本地订阅表 localSubTopic 确认当前是否订阅了该主题本身或者该主题的父亲主题。如果没有，则利用 getTopicFather()和 totalDirectSons()两个方法取得该主题的直接父亲主题以及自己的所有兄弟主题，以此判断该主题是否需要聚合。SubPubMgr 相关类设计表如下表所示。

表 5-10 SubPubMgr 类设计

类名：SubPubMgr		
方法名	返回值类型	说明
localSubscribe	boolean	处理本地产生的新订阅
localUnsubscribe	boolean	处理本地产生的新取消订阅
localPublish	boolean	处理本地产生的新发布注册
localUnpublish	boolean	处理本地产生的新取消发布
needJoin	boolean	判定新订阅是否需要聚合
spreadSPInfo	void	本地订阅或发布情况变更，进行全网广播

无论节点新注册了什么服务，都要通过 Webservice 发到发布/订阅的消息域里，最终在 SubPubMgr 类中进行处理。SubPubMgr 类的类图如下图所示。

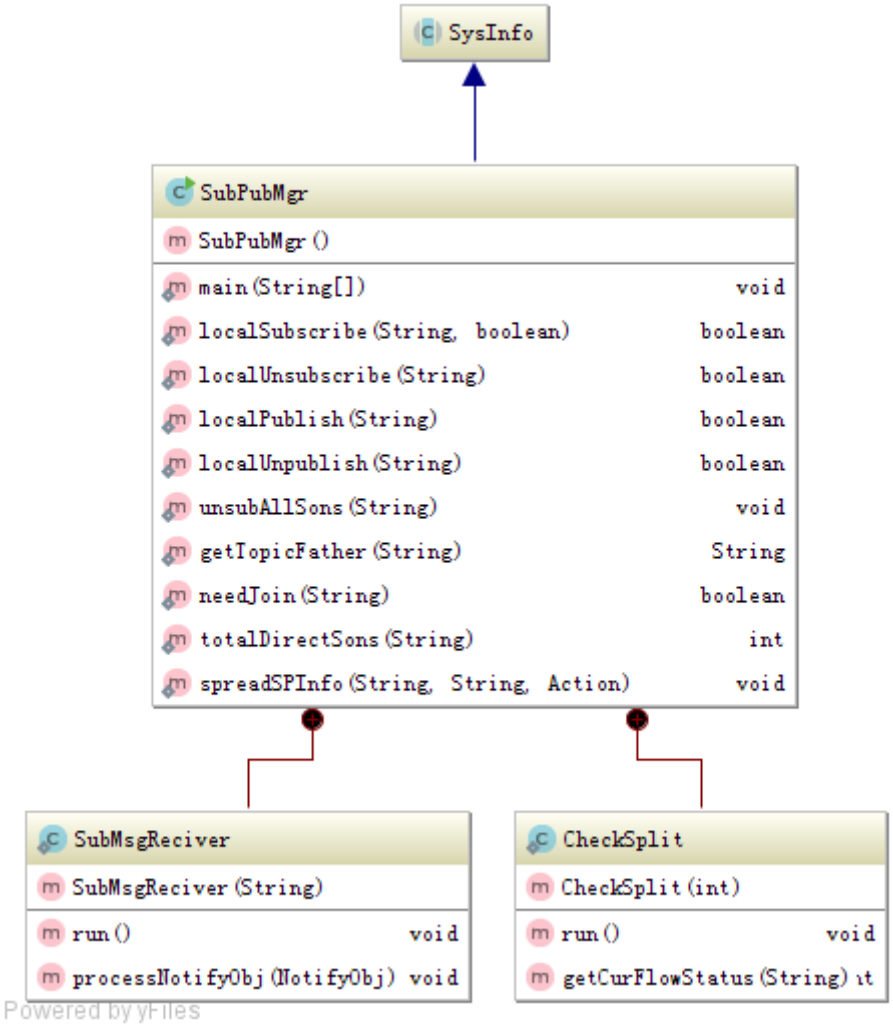

图 5-8 订阅管理器类图

如果需要聚合，则将该节点上该主题所有的兄弟主题的订阅都取消，转而订阅该主题的父亲主题。这个过程中需要将所有受影响的主题都分类保存到 joinedSubTopics（由聚合产生的订阅）和 joinedUnsubTopics（由聚合而取消的订阅）里，后面订阅表在执行分裂操作时需要读取这两项。

如果不需要聚合，则进入到广播通知阶段。节点将本次订阅的变化生成一条集群状态变化消息 LSA，向全网的节点进行广播，收到该消息的节点则会更新本地的集群信息库 allGroups。具体流程如图 5-9 所示。

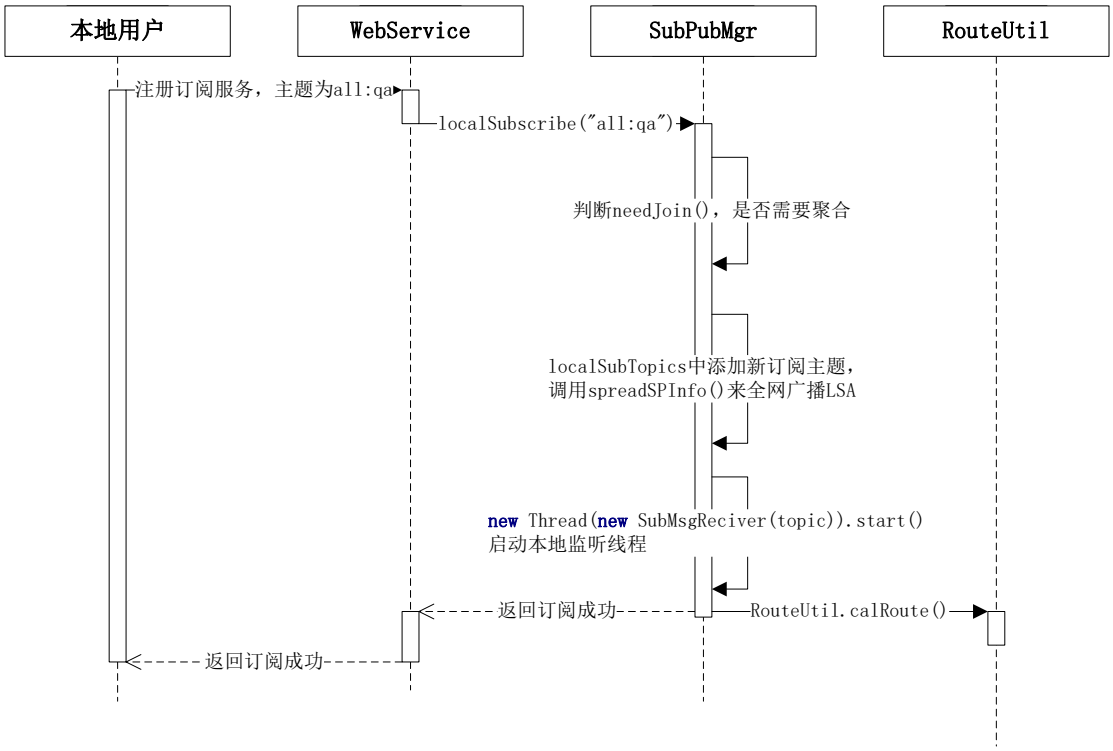

图 5-9 本地新增订阅者注册时序图

之后所有控制器节点开始计算相关主题的转发路径，将与自己相关的部分的流表下发。流表分成了两级，#0 流表已经在节点启动之初完成初始化，因此只需要根据计算结果，选择第二级流表中对应端口号的那张流表，下发或更新流表项的 Action 即可。

5.2.3 分裂与聚合机制的实现

订阅表的聚合主要是为了精简流表的尺寸。在节点数量增多以及主题树十分庞大的前提下，如果任由订阅表膨胀，很快 OpenFlow 交换机就会无法顺利匹配，毕竟最终流表是按照流表项的优先级线性匹配的。因此，节点在进行订阅的同时也要进行订阅主题的整合，适当的将重复的订阅整合在一起统一进行订阅，节约流表中的空间，提升匹配的效率。订阅表聚合的核心部分代码如下表所示。

表 5-11 订阅表聚合核心代码

变量说明：

topic：需要判断是否需要聚合的订阅主题

subBros：该主题的兄弟主题中，被订阅的主题个数

topicPath：将主题用“:”分隔形成的数组，表示主题每层的名字

notifyTopicAddrMap：保存着所有 NotifyTopic 及其对应的编码

表 5-11 订阅表聚合核心代码（续上表）

变量说明：

topicLevel：当前主题所在的层数

totalDirectSons：该主题的兄弟主题的总数

localSubTopics：本地订阅表

```
private static boolean needJoin(String topic) {
    int level = topic.split(":").length;

    for i : 1 → topicPath.length – 1 // 前面名字相同，层级比自己少一层的主题，就是父主题
        fatherTopic += ":" + topicPath[i];

    for t in notifyTopicAddrMap
        int topicLevel = t.split(":").length;
        if (t.contains(father) // 主题树中父主题与自己相同
            && topicLevel == fatherLevel + 1) // 且层级比父主题多一级
            totalDirectSons ++; // 都是兄弟主题（或者 topic 自己）

    for lst in localSubTopics
        if (lst.contains(father)
            && lst.split(":").length == level // 本地订阅表中的兄弟主题
            && !lst.equals(topic)) // 且不是 topic 自己
            subBros++;

    return subBros > totalDirectSons / 2;
}
```

在判断是否需要聚合订阅时，具体决定条件是该节点订阅了新主题的多数兄弟主题，通过主题的层级关系找到 totalDirectSons，通过 localSubTopics 得到 subBros，如果需要聚合，那么就重新调用 localSubscribe()，将父主题进行订阅，同时取消之前订阅的所有兄弟主题。这个过程中将父主题保存一份到

joinedSubTopics 中，表示该主题并非真实订阅，只是由聚合产生的。同理在取消兄弟主题订阅时也要保存一份到集合 joinedUnsubTopics，方便后面在分裂订阅表时进行查阅。

最后无论新订阅是否需要聚合，节点都要将这一变化保存到本集群的订阅表 groupSubMap 以及 LSDB 中，通过 GroupUtil.spreadLocalGrp()函数将这一变化以 LSA 广播的形式通知全网。最后，节点会进行路由的计算，将本集群作为新的订阅集群，以最小代价添加到网络中该主题的发布订阅树中。Group 类同样继承了 DevInfo 类，它是 LSA 消息的具体形式，因此也具备 LSA 消息要求的内容，其设计见下表。

表 5-12 Group 类设计

类名：Group		
属性名	类型	说明
id	String	LSA 消息的唯一编号
updateTime	long	该集群内容最后一次更新的时间
groupName	String	该集群名字
dist2NbrGrps	Map<String, Integer>	该集群与其邻居集群间的距离。key 是邻居 group 的名字，value 是二者间链路的距离

如果说订阅表的聚合可以减少 OpenFlow 交换机中的流表项数，那么订阅表的分裂则会保证链路中的流量处于一个相对经济的水平。在发布订阅管理器 SubPubMgr 启动的时候，就会启动一个定时任务，通过定期向集群控制器请求当前每个 OpenFlow 交换机中流表项的匹配情况，如果（聚合后主题匹配成功数）/（聚合前主题匹配成功总数）低于阈值，那么就认为该主题应当被分裂，节点会取消该主题的订阅，同时将 joinedUnsubTopics 中该主题的孩子主题重新订阅。具体流程如图 5-10 所示。

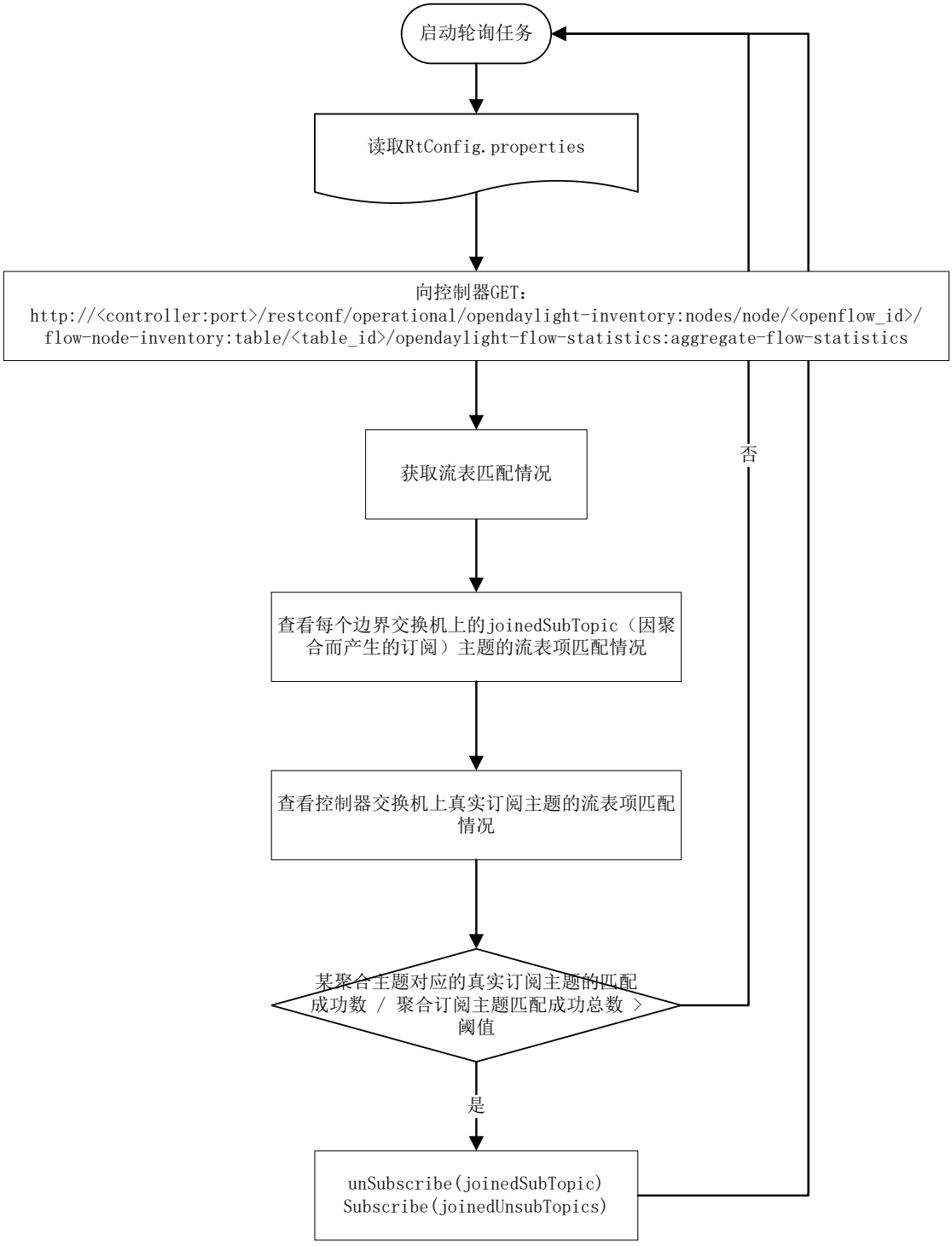

图 5-10 订阅表分裂流程图

5.3 路由模块的实现

RouteUtil 是负责计算路由的核心模块，其内部结构如表 5-13 所示。

表 5-13 RouteUtil 类设计

类名：RouteUtil		
方法名	返回值类型	说明
calRoute	List<String>	计算集群内交换机之间的最短路径。返回值是 OpenFlow ID 组成的序列
calNetworkRoute	List<String>	计算集群间的最短路径。返回值是集群名组成的序列
downInGrpRtFlows	void	下发集群内最短路径对应的流表项
downBridgeFlows	void	下发集群间最短路径中与本集群相关的流表项
updateInGrpChange	void	更新集群内发布订阅状态
updateNbrChange	void	更新集群间发布订阅状态

其中 RouteSyncMsgReceiver 用来在集群内同步计算好的路径，方便其他节点查询。RouteUtil 是计算路由的核心，通过查询 SysInfo 中保存的全网订阅树，找到相应主题后进行路由的计算。如果新注册的请求来自集群内，就调用 updateInGrpChange()；如果新注册的是网络中的其他集群，那么就调用 updateNbrChange()。

在函数通过 SPF 算法计算出每条路径的最短距离后，选择最优的路径作为函数返回值，并将其作为参数传给 downBridgeFlow()。图 5-11 为路由计算的相关类图，

downBridgeFlow()函数会按照刚才算出来的最短路径，将与自己相关的流表进行下发。这里下发流表的情况一共有三种，即：本集群处在路径的起点（i＝0，本集群内部产生的新订阅）、路径的中间（1 <= i <= route.size()-2 过路集群）、路径的结尾（i＝route.size()-1，本集群是被选中的那个最近的订阅集群）。控制器节点会根据位置的不同，下发相应的流表。在新订阅者加入到发布订阅树的时候，它和原树之间的流表是双向的，这是为了确保后面注册订阅的集群只要能够连接到该订阅树中任意一个集群，就可以收到这个发布/订阅消息子域内的全部消息。

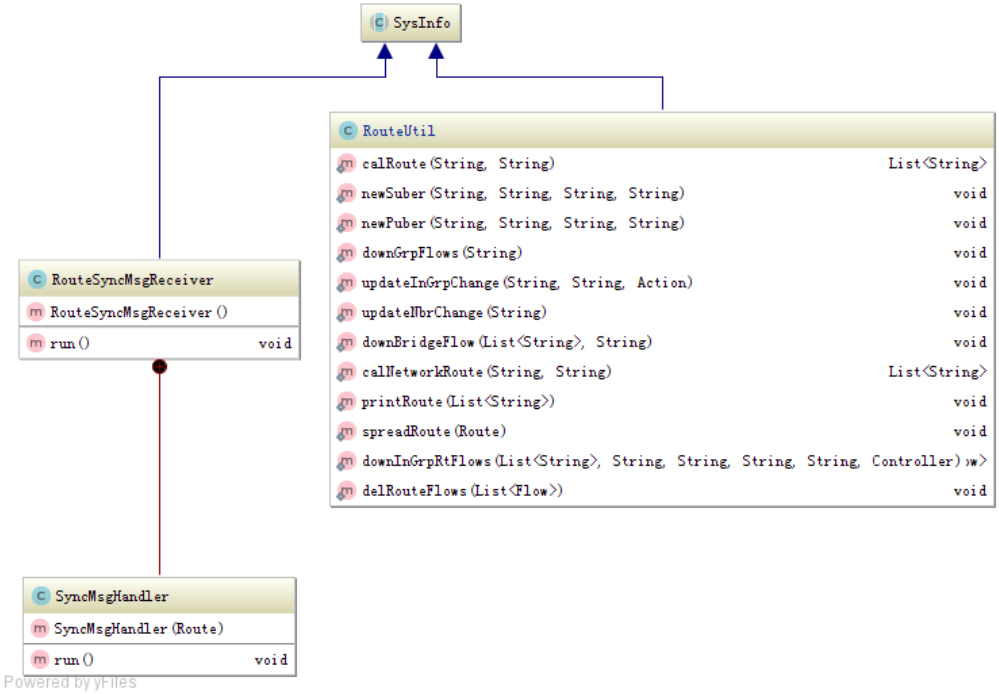

图 5-11 路由计算相关类图

最后节点调用 FlowUtil.generateFlow()函数生成流表，再调用 downFlow()函数下发流表项。流表项的相关类设计如下表所示。

表 5-14 Flow 类设计

类名：Flow		
属性名	类型	说明
flow_id	String	流表项唯一标号
table_id	int	流表项所在的流表的标号
priority	int	流表项的优先级
topic	String	流表项匹配的主题
swtId	String	流表项所下发到的交换机的 OpenFlow ID
in	String	流表项匹配的数据包的进端口
out	String	流表项匹配的数据包的出端口
nv_src	String	流表项匹配的 ipv4_src 字段的值
nw_dst	String	流表项匹配的 ipv4_dst 字段的值
ipv6_dst	String	流表项匹配的主题对应的 ipv6 字段的值

流表项中包括流表名（table_id）、优先级（priority）、主题名（topic）以及对

应的匹配项值（ipv6_dst）等参数，节点可以通过这些参数在 FlowUtil 中对已经下发过的流表项进行查询，同时 FlowUtil 也可以直接向 OpenDaylight 控制器发送请求来查找特定流表项。FlowUtil 类设计表如下表所示。

表 5-15 FlowUtil 类图

类名：Flow		
方法名	返回值类型	说明
generateFlow	Flow	生成普通转发类消息的流表项
generateNoInPortFlow	Flow	生成无需匹配入端口的消息的流表项
generateRestFlow	Flow	生成 REST 支持类流表项
downFlow	void	将生成的 Flow 发送至控制器，请求控制器下发该流表项
deleteFlow	void	向控制器请求删除特定的流表项

根据所给参数和所需功能的不同，节点在 FlowUtil 中会调用三个不同的函数，分别是生成普通转发类消息的流表项的 generateFlow()，参数包括 in_port、ipv6_dst 、 out_put 等；生成无需匹配入端口的消息的流表项的 generateNoInPortFlow()，参数中没有 in_port，表示只要满足匹配主题 ipv6_dst 就转发到特定端口或加入队列；以及生成 REST 支持类流表项的 generateRestFlow()，匹配的是 IPv4 的源地址 nw_src 和目的地址 nw_dst，因为发给控制器的请求只指定了 IPv4 地址。FlowUtil 类相关类图如下图所示。

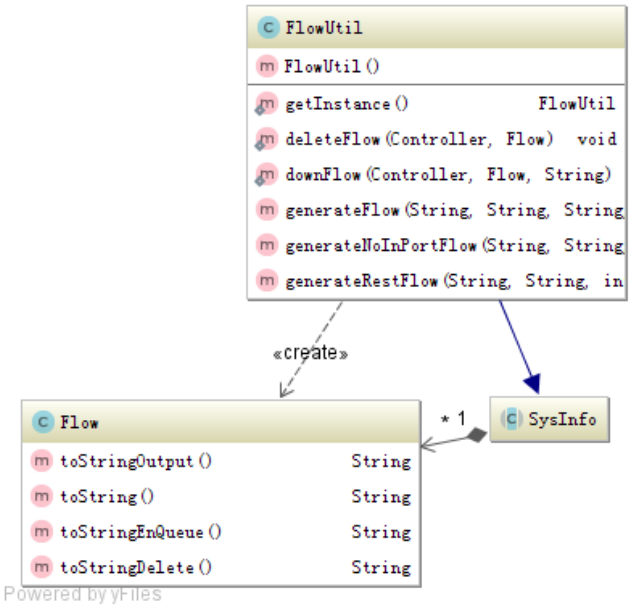

图 5-12 FlowUtil 相关类图

5.4 本章总结

本章主要介绍了在第四章中提到的几个具有代表性的模块的具体实现。其中主要有拓扑的构建和维护，订阅的动态管理以及路由模块的实现和优化。在拓扑维护模块以及路由计算模块中，针对具体的设计给出了设计的目的以及实现中的特点。

第六章 系统测试

本系统的测试主要分为功能测试和性能测试，测试内容主要包括拓扑维护和路由计算等功能模块的基本功能以及运行效率。

6.1 测试目标

为保证基于 SDN 网络的发布/订阅系统的各个模块能够满足各项基本需求，我们对各个功能模块进行了单元测试和集成测试。本文主要针对路由模块方面的优化与实现，因此对该模块本身以及与其功能相关的路由维护模块以及订阅管理模块进行详细的测试。功能测试的对象主要是在设计与实现部分列出的各模块的主要功能，即验证拓扑构建与维护功能，订阅管理的有效性，路由计算以及流表下发的有效性和可靠性。

6.2 测试环境

测试环境使用 OpenvSwitch 创建虚拟网桥，利用 KVM 和 Libvirt 创建并管理虚拟机，每个服务器中安装多个虚拟机和虚拟网桥，连接成测试网络，同时在虚拟网桥上设置控制器，将虚拟网桥的控制器设置为相应集群的集群控制器，形成多集群测试环境。具体环境如表 6-1 所示：

表 6-1 测试环境

硬件环境	
IBM 服务器	3 台 System x3850 X5 服务器
操作系统	Ubuntu 14.04.1 Desktop
CPU	Intel(R) Xeon(R) CPU E7520 1.87GHz 16 核
服务器内存	32G（大集群中每个虚拟机分配 4G，小集群每个虚拟机分配 8G）
服务器硬盘	1.5TB（大集群中每个虚拟机分配 10G，小集群每个虚拟机分配 40G）
网卡	双网卡

表 6-1 测试环境（续上表）

软件环境	
OpenvSwitch	2.4.0（支持 OF 1.3）
OpenFlow 协议	1.3（支持 IPv6 地址作为匹配项）
KVM	1.0（模拟硬件功能）
Libvirt	1.3.0（进行虚拟机的管理）
JDK	1.7.0_79

实验拓扑示意：

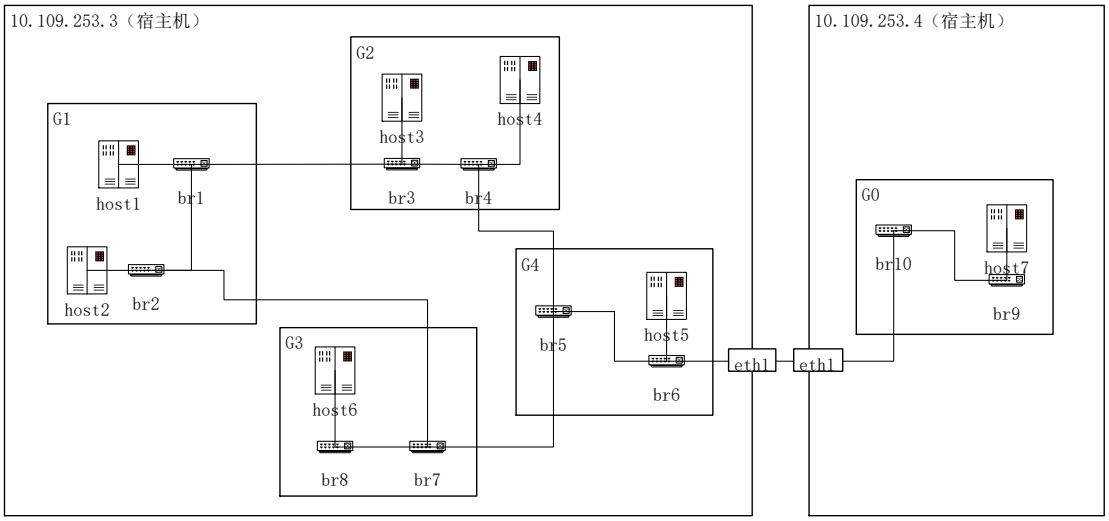

图 6-1 实验环境示意图

其中服务器内部创建的 OpenvSwitch 网桥（brX），可以起到虚拟交换机的作用，同时由于 OpenvSwitch 中网桥的特殊设计，使得它可以支持 OpenFlow 协议 1.3 版，这版协议标准提供了对 IPv6 目的地址和 IPv6 源地址的匹配项支持，使得实现自定义的匹配项成为可能。

在两个网桥之间建立连接的方案有 Patch Peer 和 Veth Pair 两种，在权衡性能与功能上的差异后，由于通过 Patch Peer 进行连接时无法在这些端口上添加消息队列，本文最终选择了 Veth Pair 这种方案进行 OpenvSwitch 虚拟网桥的连接，通过在虚拟网桥间建立连接，就可以在服务器上构造复杂的拓扑环境。

在物理服务器上有两块网卡，将其中一块网卡接入校园网，方便远程连接服务器进行虚拟机的配置和拓扑的调整。另一块网卡添加到虚拟网桥上，成为虚拟网桥的一个端口，通过集线器将几个服务器连接在一起，实现服务器所构筑的集群的连接。

在划分集群时，本文默认将同一控制器控制的 OpenFlow 交换机以及它们上

面连接的主机划分为同一集群。因此需要在每个集群中都运行了一个OpenDaylight 控制器的实例，作为集群控制器。

6.3 系统功能测试

本小节主要描述系统中路由计算及其相关模块的功能测试，预期结果应与设计与实现中描述的一致。

6.3.1 拓扑维护模块测试

1）集群内拓扑获取测试

测试项目	新节点启动并完成初始化操作的测试。
测试目的	验证新节点启动后能否成功检查配置文件进行本地设置，向集群控制器请求当前集群的拓扑；若本地 IP 是集群控制器 IP，则还需验证控制器节点能否正常对集群内的 OpenFlow 交换机下发初始化流表。
测试步骤	首先在虚拟机中构造集群，在集群内添加虚拟网桥，预先设置好网桥间的连接情况以及控制器 IP，然后启动部署在虚拟机中的节点程序，虚拟机的网卡与集群中 br1 相连，也就确保了这个虚拟机构成了集群上的一个主机。
预期结果	节点在启动过程中不报错，正确打印获取到的配置信息，同时启动相应的服务（如轮询等），打印初始化流表下发情况。
实际结果	与预期相同，在程序运行时打印的信息如下图所示。可以看到配置文件被正确读取，同时连接控制器成功，获取到了集群内的拓扑信息，同时初始化流表也已经下发。

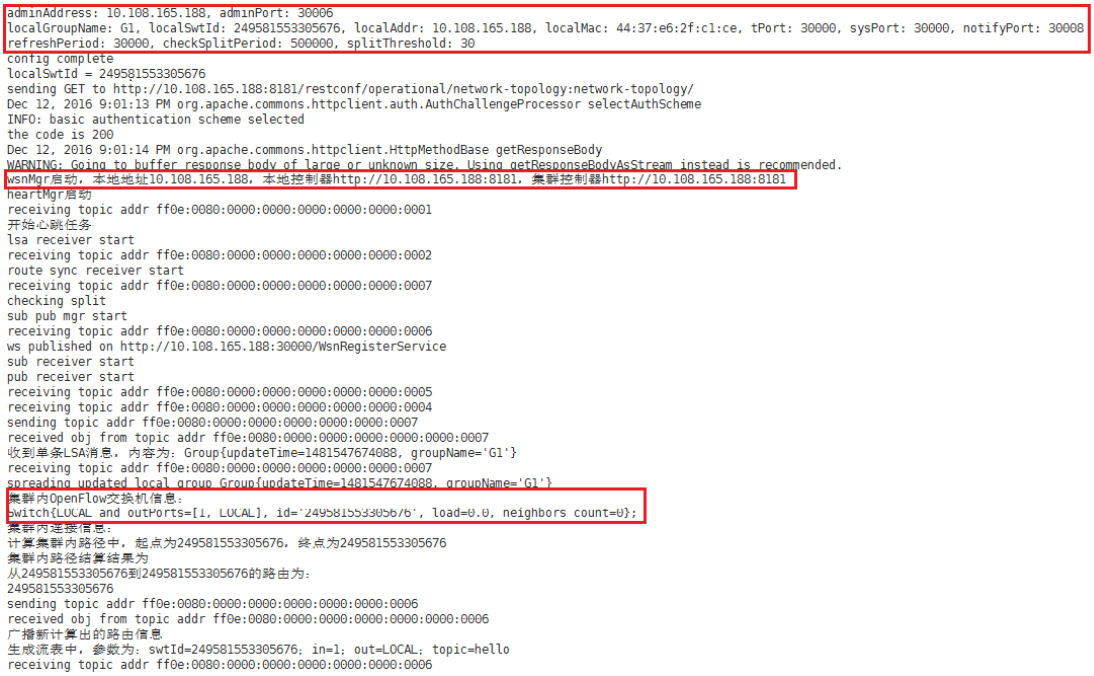

图 6-2 节点启动获取集群内拓扑

2）集群内节点丢失测试

测试项目	新节点启动后感知集群内节点丢失的测试。
测试目的	验证新节点启动后能否持续向集群控制器请求当前集群的拓扑；并在集群内节点丢失后成功获知，从而更新节点内的集群信息。
测试步骤	首先在虚拟机中构造集群，在集群内添加多个虚拟网桥，预先设置好多个网桥间的连接情况以及控制器 IP，然后启动部署在虚拟机中的节点程序，在节点已经可以从控制器获得集群内拓扑后，将集群内的 br1 删除，观察节点打印情况。
预期结果	节点在轮询时打印获取到的集群内拓扑情况，包括 OpenFlow 交换机的个数以及其端口数量等，在删除 br1 后，发现节点在打印时已经将 br1 对应的交换机从保存的结果中删除。
实际结果	与预期相同。可以看到节点在轮询的过程中不断打印当前节点保存的集群内拓扑情况，在删除 br1 后，打印的内容中就不再出现 br1 对应的交换机。

3）集群间邻居构建测试

测试项目	新集群中控制器节点能否正确建立邻居关系。
测试目的	心跳管理员会随新集群的控制器节点一起启动，而心跳管理员会向邻居集群发送 Hello 消息，如果 Hello 消息收到回复，就意味着

	能够建立邻居关系，也就证明可以在邻居集群间建立连接。
测试步骤	在不同的集群中分别部署两套节点程序，首先启动 G1 中节点程序，待其启动完成后，启动 G2 中节点程序，二者在互相试探后会完成 Hello 消息、ReHello 消息以及 FinalHello 消息的传输，建立起邻居关系。
预期结果	节点启动无异常，正确打印三种消息的内容，包括来源集群以及目的集群的名称和它们边界交换机的端口。最后在建立邻居后会打印当前全部邻居的信息。
实际结果	与预期相同，程序运行时打印的信息如下图所示。可以看到节点对自己多个 outPort 进行轮询，因此在启动时不确定是由谁发起的 Hello 消息，但之后两个节点会相继按顺序打印自己收到的消息内容，最后打印出本地集群的所有邻居，可以发现二者已经成为了对方的邻居。

```
向交换机139329991887426通过1端口发送Hello消息
收到ipv6地址为ff0e:0080:0000:0000:0000:0000:0000:0001的消息
正在监听ipv6地址ff0e:0080:0000:0000:0000:0000:0000:0001
收到ipv6地址为ff0e:0080:0000:0000:0000:0000:0000:0007的消息
收到ipv6地址为ff0e:0080:0000:0000:0000:0000:0000:0002的消息
收到单条LSA消息，内容为：Group{updateTime=1481597333305, groupName='G1'}
正在监听ipv6地址ff0e:0080:0000:0000:0000:0000:0000:0007
从G1集群收到ReHello消息，我方边界交换机为139329991887426，对外端口为1
收到ipv6地址为ff0e:0080:0000:0000:0000:0000:0000:0001的消息
正在发送ipv6地址为ff0e:0080:0000:0000:0000:0000:0000:0007的消息
spreading updated local group Group{updateTime=1481597320281, groupName='G2'}
收到ipv6地址为ff0e:0080:0000:0000:0000:0000:0000:0007的消息
收到单条LSA消息，内容为：Group{updateTime=1481597320281, groupName='G2'}
正在监听ipv6地址ff0e:0080:0000:0000:0000:0000:0000:0007
正在发送ipv6地址为ff0e:0080:0000:0000:0000:0000:0000:0001的消息
向集群G1回复FinalHello消息
```

图 6-3 G1 构建邻居

```
收到来自G2的Hello消息
生成流表中，参数为：swtId=249581553305676；in=LOCAL；out=1；topic=re_hello
update flow "table=0,priority=20,dl_type=0x86DD,in_port=LOCAL,ipv6_dst=ff0e:80::2/128,action=output:1" complete
下发从本地交换机到249581553305676交换机的1端口的ReHello消息流表
向集群G2回复ReHello消息，我方边界交换机为139329991887426，对外端口为1
正在发送ipv6地址为ff0e:0080:0000:0000:0000:0000:0000:0007的消息
收到ipv6地址为ff0e:0080:0000:0000:0000:0000:0000:0007的消息
正在发送ipv6地址为ff0e:0080:0000:0000:0000:0000:0000:0002的消息
spreading updated local group Group{updateTime=1481597333305, groupName='G1'}
收到单条LSA消息，内容为：Group{updateTime=1481597333305, groupName='G1'}
正在监听ipv6地址ff0e:0080:0000:0000:0000:0000:0000:0007
正在发送ipv6地址为ff0e:0080:0000:0000:0000:0000:0000:0001的消息
收到来自集群G2的FinalHello消息
从G2集群获得了FinallHello消息，此连接中我方边界交换机为139329991887426，对外端口为1
收到ipv6地址为ff0e:0080:0000:0000:0000:0000:0000:0001的消息
收到ipv6地址为ff0e:0080:0000:0000:0000:0000:0000:0007的消息
收到单条LSA消息，内容为：Group{updateTime=1481597320281, groupName='G2'}
正在监听ipv6地址ff0e:0080:0000:0000:0000:0000:0000:0007
```

图 6-4 G2 构建邻居

4）集群间邻居丢失测试

测试项目	集群对自己邻居关系进行检查和维护的测试。
测试目的	验证在已经建立邻居关系的集群间发送心跳消息，可以有效保证集群间邻居关系的有效性。同时验证集群在邻居丢失的时候能够感知到这种变化，并更新本地的数据。
测试步骤	首先启动两个集群中的节点程序，让二者建立邻居关系，确定二者已经开始正常发送心跳消息后，将 G2 中的节点程序退出，由此形成一个集群掉线的状况，观察 G1 的节点程序是否能够检测到这种掉线情况。
预期结果	在 G2 的所有节点程序都停止运行后，G1 首先会继续发送心跳消息，在多次发送没有回复后，G1 会将 G2 从本地存储的邻居信息中删除，并且打印剩余的邻居信息。
实际结果	实际结果：与预期相同。可以看到在 G2 掉线后，G1 的控制器节点又打印了几次心跳包的信息，但是并没有收到回复，之后 G1 就打印了新的邻居信息，发现 G2 已经不在其中了。

6.3.2 订阅管理模块测试

1）新订阅处理测试

测试项目	本地订阅能否正确提交到控制器节点，而节点收到订阅后能否正确处理。
测试目的	在用户通过 WebService 进行了主题的订阅后，节点应当可以接收到新订阅，同时在收到后进行包括聚合在内的一系列处理，最终将该订阅保存。
测试步骤	首先启动节点程序，之后通过 WebService 地址向节点发送新订阅请求，订阅 All:a 主题，观察节点打印情况。
预期结果	在节点收到新订阅消息后，应该首先打印收到的订阅信息，而后打印当前节点上关于该主题的订阅情况，而后打印聚合的判定情况，如果该订阅有效，无论是聚合与否，都会打印节点新的订阅情况。
实际结果	程序运行时打印的信息如下图所示。可以看到节点在接收到订阅信息后，首先查看了本地的订阅信息，发现允许订阅后又进行了聚合判定，发现 All 主题下只有新订阅 All:a，因此不用聚合，最终节点更新并广播了本地的新订阅消息，同时触发了路由计算模块。

```
new webservice register: SUB#all:a#123
判定是否为聚合订阅：false
搜索当前订阅表中有无订阅过all:a主题的父亲主题
更新本地订阅表localSubTopics
判定新订阅是否需要聚合
新订阅不需要聚合，判定新订阅是否为之前聚合而成的订阅
新订阅是全新订阅，更新本集群订阅信息groupSubMap
正在发送ipv6地址为ff0e:0080:0000:0000:0000:0000:0000:0005的消息
收到ipv6地址为ff0e:0080:0000:0000:0000:0000:0000:0005的消息
new suber in group, sub topic is all:a
全网广播集群内订阅情况变更，变更类型为SUB，相关主题是all:a
正在监听ipv6地址ff0e:0080:0000:0000:0000:0000:0000:0005
正在发送ipv6地址为ff0e:0080:0000:0000:0000:0000:0000:0007的消息
广播当前集群LSAGroup{updateTime=1481599948358, groupName='G1'}
收到ipv6地址为ff0e:0080:0000:0000:0000:0000:0000:0007的消息
收到单条LSA消息，内容为：Group{updateTime=1481599948358, groupName='G1'}
正在监听ipv6地址ff0e:0080:0000:0000:0000:0000:0000:0007
新监听启动中，监听主题为all:a
新增订阅主题all:a，新订阅集群为G1，订阅者所在OpenFlow交换机为249581553305676
正在监听ipv6地址ff0e:0080:0000:0000:0000:0000:2222:0006
update flow "table=0,priority=20,dl_type=0x86DD,ipv6_dst=ff0e:0080:0000:0000:0000:0000:2222:0006/128,action=
集群内订阅状态发生变化，正在重新计算路由
路由重新计算完毕
```

图 6-5 集群新增订阅测试

```
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=569190.831s, table=0, n_packets=0, n_bytes=0, idle_age=65534, hard_age=65534, priority=50,ipv6,ipv6_dst=ff0e:80::6 a
 cookie=0x0, duration=569.438s, table=0, n_packets=0, n_bytes=0, idle_age=569, priority=20,ipv6,ipv6_dst=ff0e:80::2222:6 actions=LOCAL
 cookie=0x0, duration=547.780s, table=0, n_packets=0, n_bytes=0, idle_age=547, priority=50,ipv6,ipv6_dst=ff0e:80::7 actions=FLOOD
 cookie=0x0, duration=547.770s, table=0, n_packets=0, n_bytes=0, idle_age=547, priority=50,ipv6,ipv6_dst=ff0e:80::5 actions=FLOOD
 cookie=0x0, duration=547.766s, table=0, n_packets=0, n_bytes=0, idle_age=547, priority=50,ipv6,ipv6_dst=ff0e:80::4 actions=FLOOD
 cookie=0x0, duration=547.760s, table=0, n_packets=0, n_bytes=0, idle_age=547, priority=50,ip,nw_src=10.108.165.188 actions=FLOOD
 cookie=0x0, duration=547.759s, table=0, n_packets=0, n_bytes=0, idle_age=547, priority=50,ip,nw_dst=10.108.165.188 actions=FLOOD
 cookie=0x0, duration=547.784s, table=0, n_packets=0, n_bytes=0, idle_age=547, priority=20,ipv6,in_port=1,ipv6_dst=ff0e:80::1 actions=LOCA
 cookie=0x0, duration=547.763s, table=0, n_packets=0, n_bytes=0, idle_age=547, priority=50,ipv6,in_port=LOCAL,ipv6_dst=ff0e:80::6 actions=
 cookie=0x2b0000000000002, duration=54259.680s, table=0, n_packets=86736, n_bytes=9800400, idle_age=4, priority=2,in_port=1 actions=drop
```

图 6-6 集群新增订阅后流表新增项目

2）订阅表聚合测试

测试项目	测试聚合功能工作情况。
测试目的	检查订阅表聚合功能能否正常运行。
测试步骤	在管理员处更改 NotifyTopicCodeMap，确保节点能够收到的主题只有 All:a、All:b 以及 All:c 三个主题。之后启动节点，通过 WebService 提交 All:a 主题订阅，在打印成功订阅后，再向其提交 All:b 主题的订阅，观察节点打印情况。
预期结果	节点将会首先检查该主题是否需要聚合，由于该主题的兄弟主题中超过一半（All:a 以及 All:b）已经被订阅，因此该主题需要被聚合，可以看到节点会打印取消订阅 All:a 主题的信息，接下来会打印新增订阅 All 主题的信息。
实际结果	与预期相同。首先打印的是判定信息，判定结果是需要聚合，而后打印取消订阅的信息，可以看到取消后订阅表为空，之后又新增了主题 All 的订阅。

3）订阅表分裂任务测试

测试项目	测试订阅表分裂任务工作情况。

测试目的	检查订阅表分裂任务能否正常运行。
测试步骤	在管理员处更改 NotifyTopicCodeMap，确保节点能够收到的主题只有 All:a、All:b 以及 All:c 三个主题。之后将 All:a 和 All:b 两个主题订阅，使主题聚合为 All，之后只发送 All:c 主题的消息，观察节点打印。
预期结果	前期打印与测试订阅表聚合时内容相同，在增大主题 All:c 消息的流量后，从控制器可以获取 All:a 主题和 All:b 主题匹配成功总次数 /All 主题匹配成功次数的比值过小，超过阈值，因此应当打印 All 主题取消订阅，同时打印重新订阅 All:a 主题和 All:b 主题。
实际结果	与预期相同。打印了 All 主题的取消订阅信息，然后打印了订阅 All:a 主题和 All:b 主题的订阅信息。

6.3.3 路由计算模块测试

1）路由计算测试

测试项目	路由计算模块能够正确计算出新订阅的路由。
测试目的	确保节点在触发路由计算后，路由计算模块能够正确获取拓扑和订阅信息，然后按照算法计算出最优接入路径，并将该结果返回。
测试步骤	搭建由 7 集群组成的测试网络，G5 为订阅集群，向其节点的 WebService 提交新订阅后待其完成初始化，再向 G0 中提交新订阅，成功触发中节点的路由计算模块后，观察其打印信息。
预期结果	其他集群在获取并打印得到的全网拓扑及其订阅情况后发现自己与最短路径无关，会停止计算。而 G0、G2 以及 G5 的控制器节点在查找并计算最短接入距离后发现当前路径与本集群有关，因此打印该路径并下发相应流表项。
实际结果	与预期相同，程序运行时打印的信息如下图所示。G0、G2 以及 G5 打印了计算出来的最短接入路径，以及下发流表时的信息。

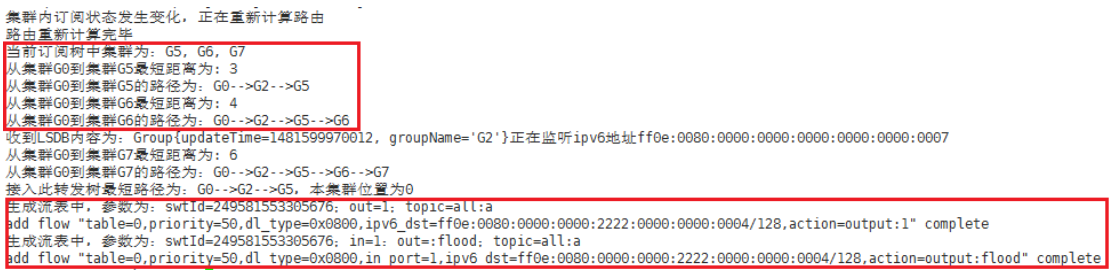

图 6-7 G0 路由计算测试

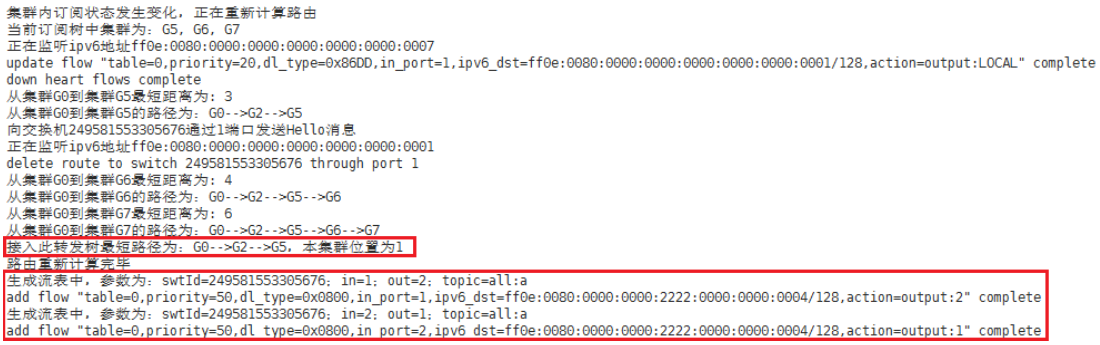

图 6-8 G2 路由计算测试

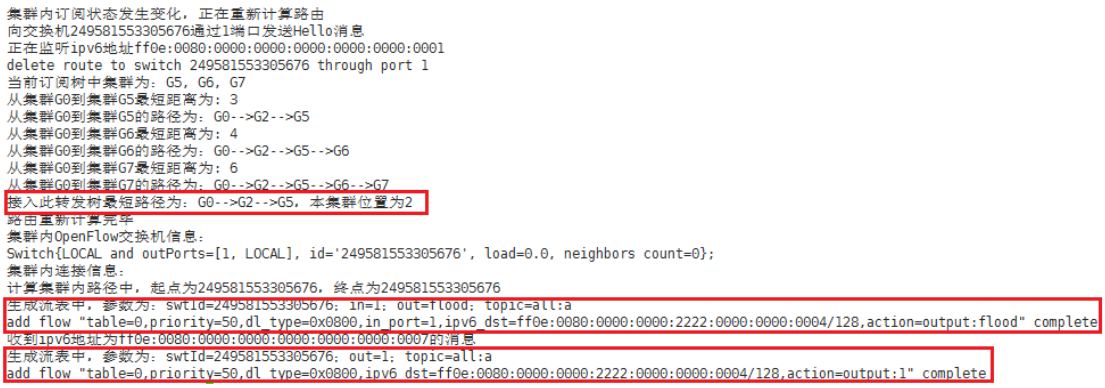

图 6-9 G5 路由计算测试

2）流表下发测试

测试项目	测试节点中的流表组件下发流表的相关功能。
测试目的	确认流表组件能够正常生成并下发流表，数据包在经过多级流表时能够逐个按层级和跳转规则进行匹配，同时流表项中的定长匹配位也能够正常发挥作用。
测试步骤	首先启动两个节点，连接在两个 OpenFlow 交换机上，再分别在两个相连的交换机上下发流表，同时匹配内容为 ipv6_dst = "FF0E:0080:0000:0000:0000:0000:FFFF:FFFF/64"，在其中一个节点上发送主题为 ipv6_dst = "FF0E:0080:0000:0000:0000:0000:ABCD:EFGH"的数据包，另一个

	节点上进行监听，观察第二个节点能否收到。
预期结果	由于交换机上匹配项是只匹配前 64bit，因此即使两个 ipv6_dst 并不完全相同，交换机也能够正常进行转发。
实际结果	与预期相同，接收节点能够正常进行打印发送节点发送的消息，丢包率符合预期。

6.4 系统性能测试

本文所述系统的性能指标主要有路由计算的速度以及消息转发的效率，因此性能测试也主要从这两方面来进行测试，检验本文所述系统设计的有效性。

6.4.1 拓扑收敛速度

在发布/订阅系统运行的过程中，新集群是动态添加到系统网络中的，因此新加入的集群构建邻居的速度，一定程度上也就决定了新发布/订阅系统进行动态扩展的效率。由于新增集群会向本集群的所有 outPort 依次发送 Hello 消息，因此一个集群构建邻居的速度主要取决于该集群 outPort 的总数。为此需要记录在不同的网络环境中进行的对比实验数据，新加入集群连接的物理邻居从 1 个依次增至 5 个，同时新加入集群的一个边界交换机始终保留一个未连接其他集群的空 outPort，多次实验并记录时间求平均值，结果如下图所示。

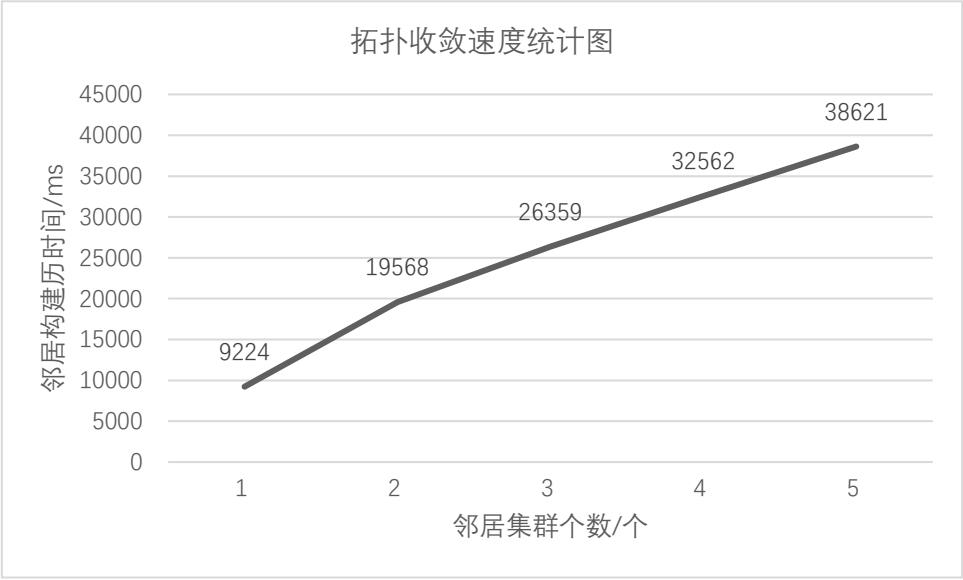

图 6-10 拓扑收敛速度统计图

如图可见，随着实际连接集群的增多，新加入集群在进行邻居构建时耗时显著上升，这主要是因为构建过程中需要频繁下发及删除流表，因此节点的计算速

度和反应的及时性都受到了影响，但是考虑到 OpenFlow 交换机一般在添加后不会频繁插拔，当前拓扑收敛的开销还是可以接受的。

在邻居信息、发布信息或者订阅信息发生变动后，集群会在全网进行 LSA 消息的广播，将更新后的链路状态信息在全网进行同步，由于路由计算的准确性依赖于 LSA 消息的准确高效传播，因此 LSA 消息在全网完成一次覆盖传播的时延也成为了衡量发布/订阅消息中间件性能的重要指标。分别用 3、4、6、8 个集群构成实验网络，其中一个集群进行 LSA 消息的发送，其余几个集群分别接收并打印收到该消息的时间，记录最后一个收到 LSA 消息的集群打印的时间，计算差值就是 LSA 消息全网覆盖传播的时延。其结果如下图所示。

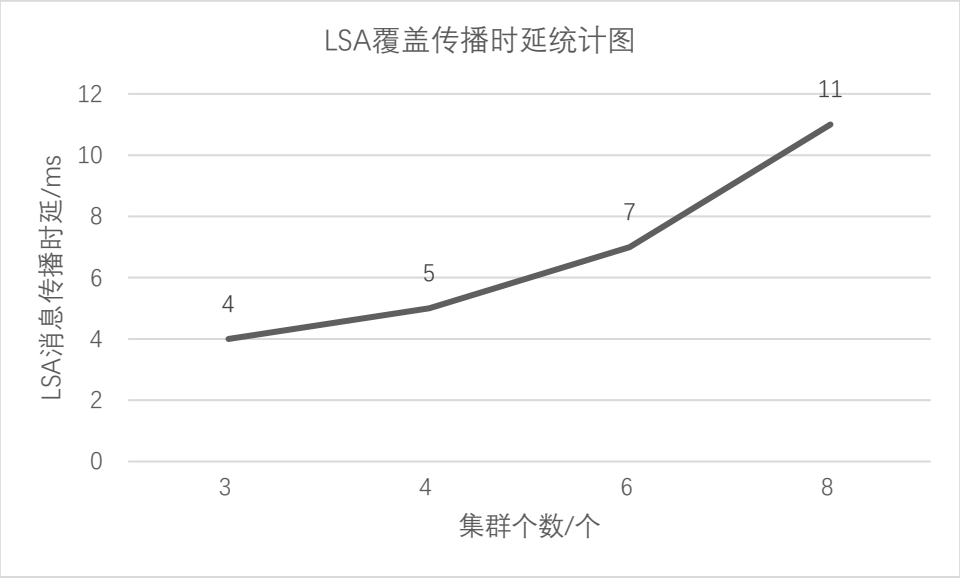

图 6-11 LSA 覆盖传播时延统计图

由图可以看出，随着网络中集群数量的增加，LSA 消息的覆盖传播到每一个集群的时延在逐渐变长，但总体上依旧在一个较低的范围。

6.4.2 路由计算时间测试

随着集群增多，整个网络也会逐渐膨胀，路由计算的负担也会越来越重，因此从整体上看路由计算的时间应当与集群的数目呈正相关，而路由计算功能是消息中间件的核心功能，因此路由计算的速度也就成为了衡量消息中间件性能的重要指标。路由计算的速度与路由模块所使用的路由计算算法有非常大的关系，同时网络中集群的总数以及订阅集群在其中的占比也直接影响了计算的效率和准确度。

因此为了更好的获取集群总数量与路由计算时间之间的关系，首先必须规定网络中订阅树的尺寸（2 个集群）以及新加入发布/订阅消息域的集群总数（2 个

集群）。之后启动每个集群中控制器节点上的程序，触发路由计算模块计算新增订阅节点到订阅树的最短路径，记录每次计算时间，多次计算后取平均值。

实验结果如图 6-12 所示，图中横轴为网络中集群总数量，纵轴为完成动作使用的平均时间。当集群总数分别为 4 个和 8 个时，路由计算与流表下发的总时间大致在 3000ms 以内，其中路由计算的时间仅有约 10ms，下发流表及确认下发成功占去了约 99%的时间。其结果如下图所示。

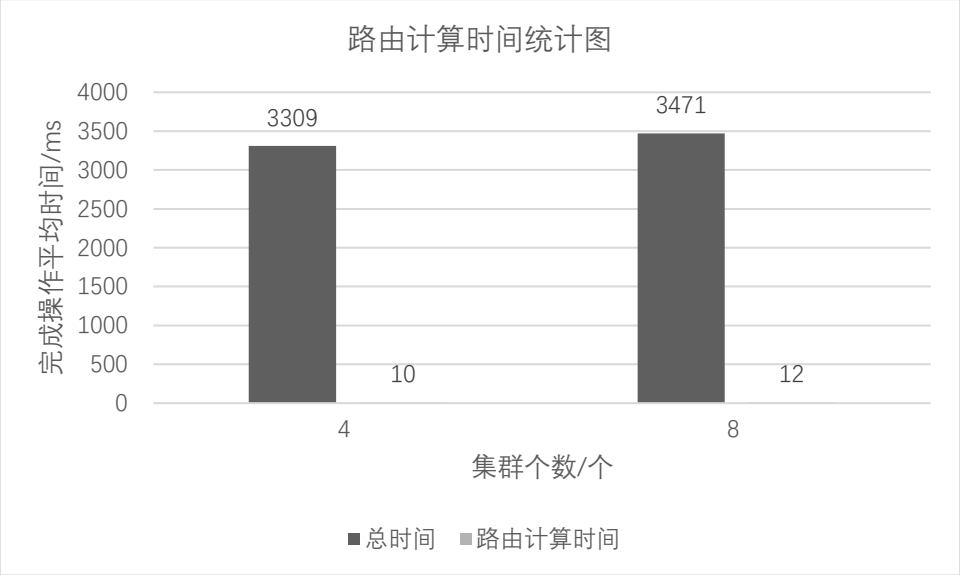

图 6-12 路由计算时间统计图

如图可以看出，由于集群个数较少，因此路由计算速度非常快，但是由于测试使用的虚拟环境在性能上存在局限性，以及 OpenDaylight 控制器串行下发流表时存在性能瓶颈，因此路径相关流表下发的总时长才远远超过路由计算的时长。如果将这个过程中下发流表的时延替换成手动下发流表时延的理想值，则会得到如图 6-13 所示的统计图。

由此图可见，总体上路由计算的速度与网络中集群数量的增长呈正相关，但并不明显。同时流表下发是相当耗费时间的操作，如果某集群需要下发多条流表，那么很有可能在此集群产生卡顿。

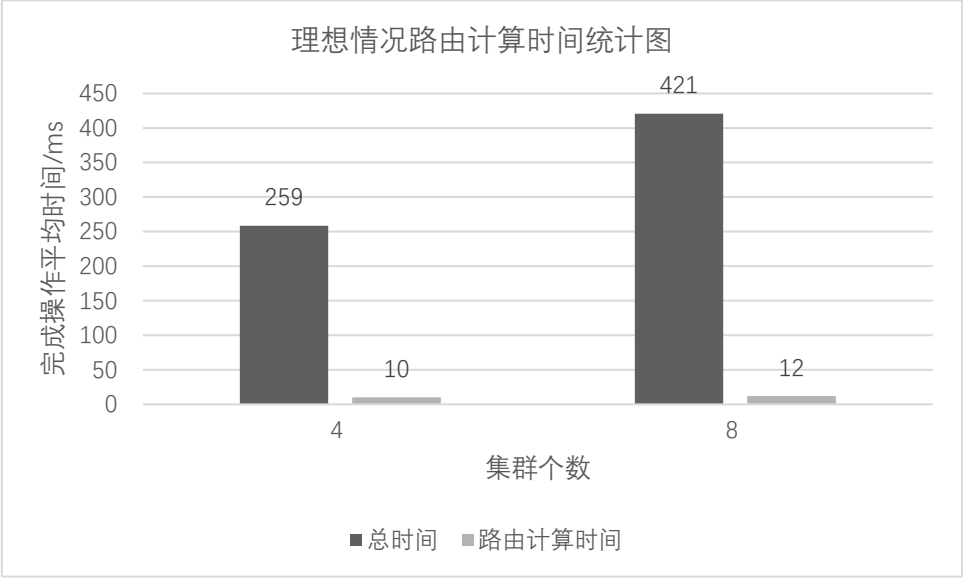

图 6-13 理想情况路由计算时间统计图

6.4.3 网络性能测试

1）吞吐量测试

网络吞吐量以及平均时延标志着发布/订阅消息中间件的性能高低，吞吐量越大，消息中间件在单位时间内能够处理的消息包数就越多，能够承载的业务量也就越大，因此本文针对不同集群个数的网络拓扑进行了测试。首先配置一个包含 8 个集群的测试网络，其中两个集群负责发送消息，其他两个集群负责接收，其余集群在网络中作为中转集群。测试网络的结构示意如下图所示。

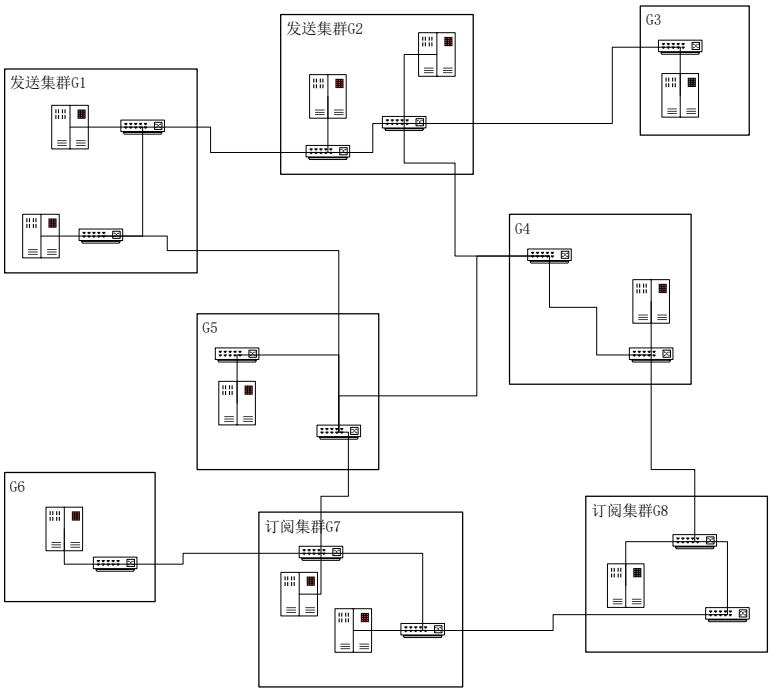

图 6-14 8 集群测试网络拓扑示意图

在集群产生并进行发布和订阅的注册时，系统会计算出相应的路由与流表项，并将这些流表项下发到沿途的 OpenFlow 交换机上，而后每个发送集群会同时连续发送指定数量的消息，而订阅集群则会打印接收到第一条和最后一条消息的时间，将两个时间取差值，就会得到系统中传输指定条数消息所需要的总时长。要注意，因为本次实验共有两个发送集群和两个订阅集群，因此系统中的消息总量应当是“单个集群发送消息数量 * 2 * 2”条。测试结果如下图所示。

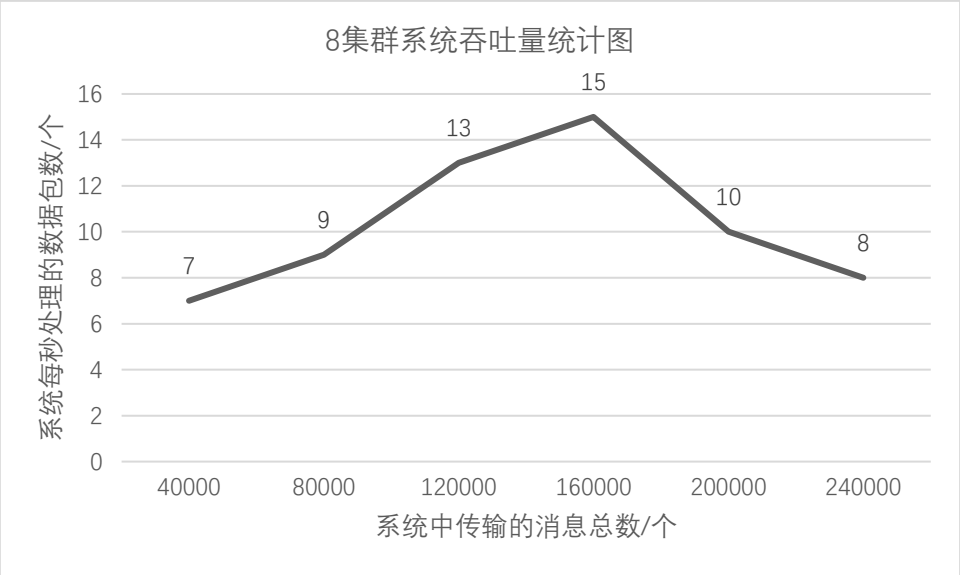

图 6-15 8 集群系统的吞吐量统计图

由图可见，当发送集群单次发送消息数逐渐增加时，系统每秒处理的消息数会逐渐上升，在达到每秒处理消息数的峰值后，随着消息数增加，系统每秒处理的消息数又会再次下降。因此系统的吞吐量大约是 10，即在系统中传输的消息总数为 200000 条时的处理效率。

这里丢包严重，主要原因还是在于实验环境是虚拟环境，同时发包速度过快，也一定程度上加重了丢包的情况。同时 WebService 接口也影响了发包的速率，也限制了系统的吞吐量。

2）时延测试

时延是指从发布集群发送第一条消息算起，到订阅集群接收到最后一条消息为止所需要的时间，这个时间会由于发布集群和订阅集群数量的不同而发生一定的变化，这种变化也反映了集群在进行消息处理时的基本性能。

实验系统拓扑依旧如图 6-14 所示，不同的是本次实验将从中随机选取若干集群作为发布集群和订阅集群，指定每个发布集群发送 100000 条消息，而后分别记录每个订阅集群收到这些信息所用的总时长，其平均值便是系统在这种情况

下的时延。订阅集群同时也会记录最终收到的消息总数，以此计算得到本次传输的丢包情况。实验结果如下图所示。

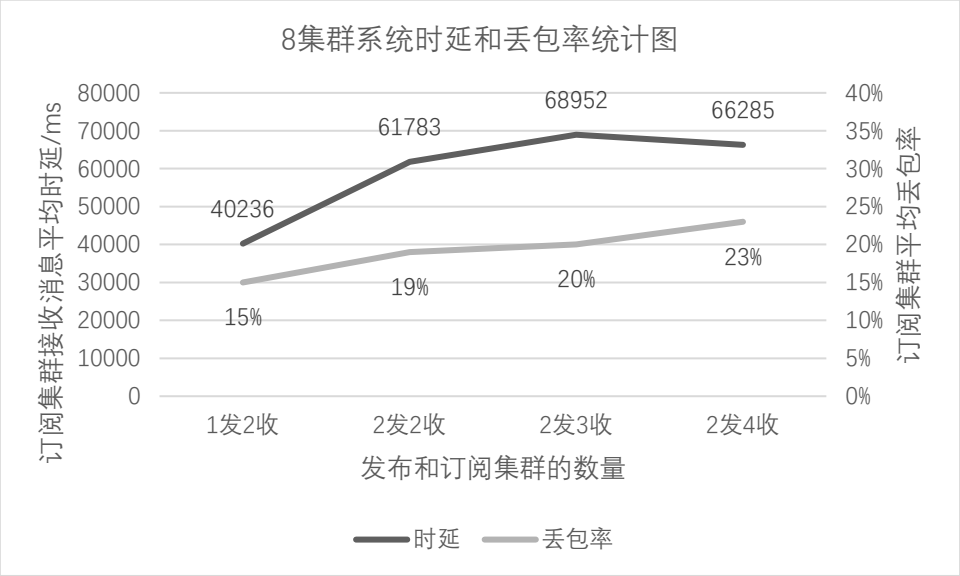

图 6-16 8 集群系统时延和丢包率统计图

随着系统中传输的消息数量增多，每个集群接收消息的时延也会随之提高。同时系统的丢包率也会升高，而升高的丢包率一定程度上又减小了系统时延。这样相互影响，最终构成了如上图所示的曲线。从总体上看，在发送消息数量比较大的时候，系统的时延变化趋于平缓，表明性能是相对稳定的。

6.4.3 集群断线重连性能测试

集群断线重连的性能同样是标志发布/订阅消息中间件性能的重要指标。由于在 SDN 网络环境中，OpenFlow 交换机有可能断掉连接，集群也有可能因为故障而失效，因此发布/订阅系统应对这种问题时的效率就十分重要了。本文首先安排了包含 4 个集群的实验网络，其拓扑如下图所示。

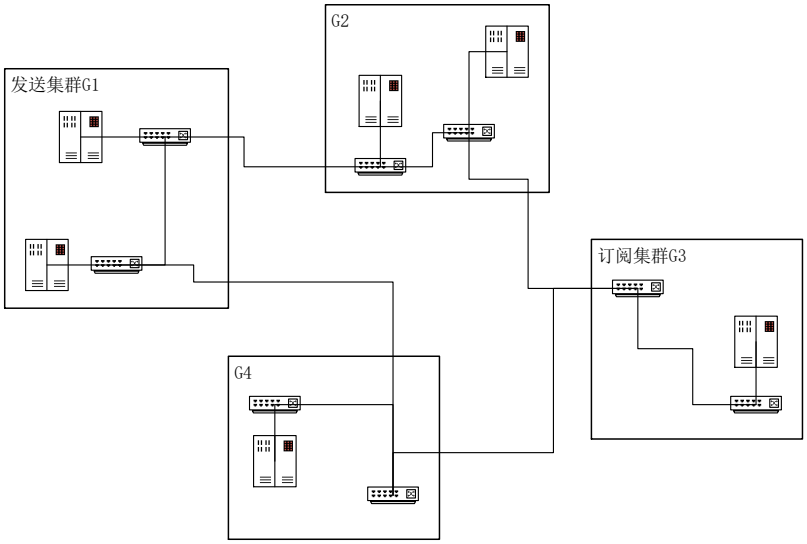

图 6-17 4 集群测试网络拓扑示意图

测试时需要首先将所有集群中的节点都启动，待其完成建立邻居的过程进入稳态后，在订阅集群和发布集群上分别启动订阅和发布程序，在发布程序所发送的数据包中，添加当前时间作为时间戳。当订阅程序能够顺利打印出接收信息后，将其计算出的转发路径上的某一集群下线，记录订阅程序用时多久能够重新接收到信息，多次记录后取平均值。

为了方便进行对比，本文之后还配置了拥有 6 个和 8 个集群的测试网络进行对比测试，这两个测试网络是在上述网络的基础上，新并入若干集群以形成更为复杂的实验环境，其中 8 集群测试网络拓扑如图 6-14 所示。

最终的测试结果如下图所示。

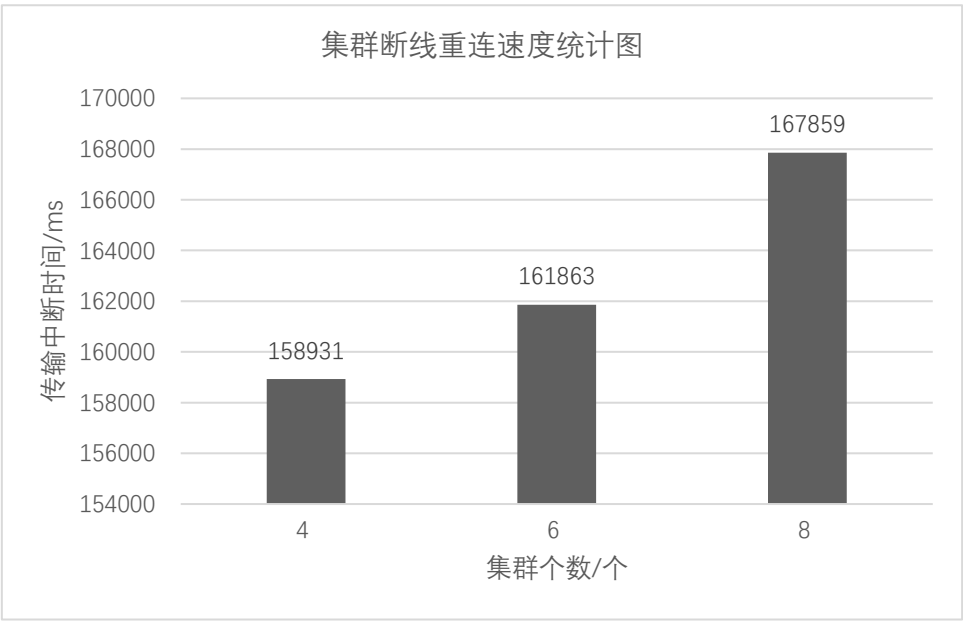

图 6-18 集群断线重连性能测试结果图

如图所示，在订阅程序等待了约 170s 后，订阅集群重新打印了新收到的 50 个数据包所携带的时间戳，标志着发布和订阅集群重新建立了连接。这 170s 包括了自定义配置中 120s 的集群失效阈值、LSA 消息的广播耗时以及计算并下发新的转发路径所需的时间。本文同时也单独记录了在收到 LSA 消息之后节点计算以及下发新路径的平均耗时，其结果如下图所示。

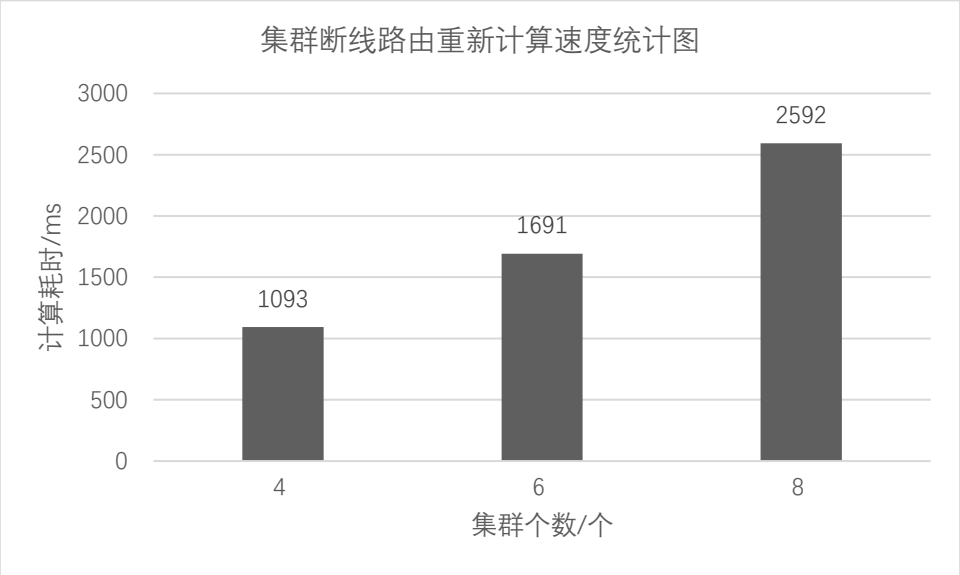

图 6-19 集群断线路由重新计算速度统计图

如图可以看到，随着网络中集群个数增多，路由重新计算的时间也在增加，这是因为新计算的路径涉及到的集群数量增多了，所以需要删除旧流表、下发新流表的集群就变多了，因此拖慢了整体的流程。

6.4.4 订阅表聚合性能测试

在对订阅管理模块进行设计与实现的时候，为了减少 OpenFlow 交换机中流表的项数，采取了在产生新订阅的时候对流表进行聚合的方案，通过合并父主题下多个子主题的订阅，减少在 OpenFlow 交换机中的流表匹配次数，以此来提高数据转发的速率。测试网络如图 6-18 所示，通过路由计算得知集群 G2 是中间的一跳，因此分情况对其下发聚合流表项或者普通流表项，以此实现对聚合结果的测定。同时在订阅集群 G3 中添加了多个主题的订阅，具体订阅情况如下图所示。

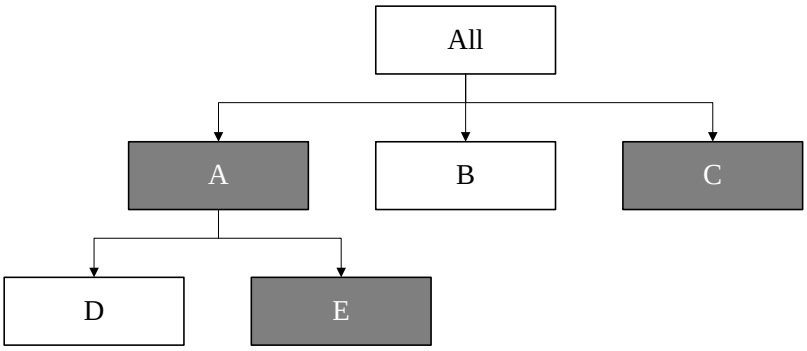

图 6-20 测试所需订阅情况示意图

其中灰色底色的 A、C 和 E 三个主题为已订阅的主题，在新增 All:A:D 主题的订阅之后，先进行无聚合的数据包发送测试，由发送集群依次发送相应主题的消息，总包数分别为 1000 包、5000 包、10000 包、15000 包、20000 包，记录下数据传输的时延。之后系统重新下发各集群的边界交换机上的流表，将其替换为聚合后的流表，再次进行数据包发送测试，最终实验结果与对比图如下图所示。

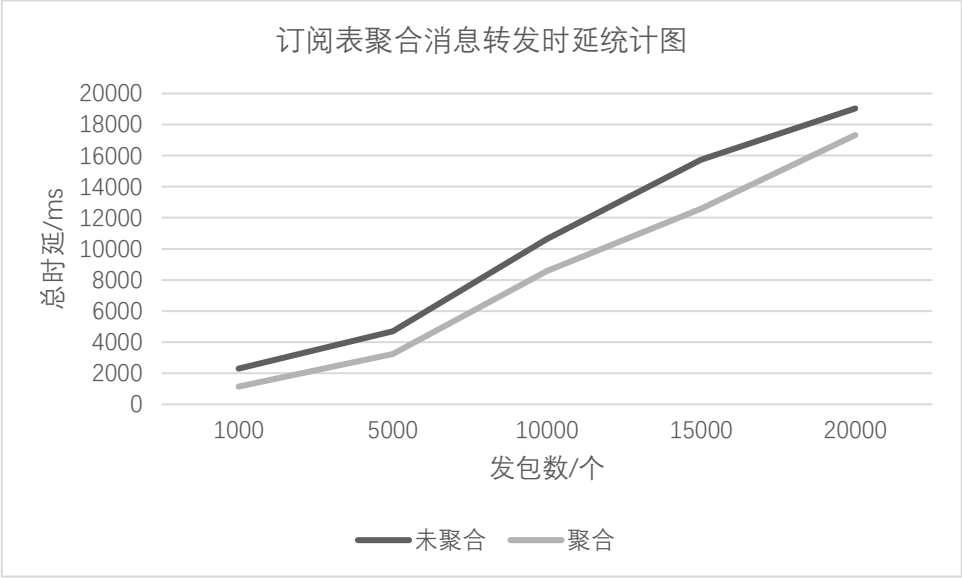

图 6-21 订阅表聚合消息转发时延测试统计图

如图可见，在没有聚合时，消息传输的时延与前一个测试的结果基本相同，可见参与传输的消息主题个数并不会影响到消息传输的速率，而在将消息传输路径上的边界交换机中的流表更换为聚合后的流表后，消息传输的速率得到了显著的提高，在发包数增加的过程中，时延相比聚合前都有了一定的下降，可见订阅表数目的减少对于消息传输性能的提高有很大的促进作用。

同时虽然消息传输的速率变快了，但是丢包率并没有增加，对比图如下图所示。

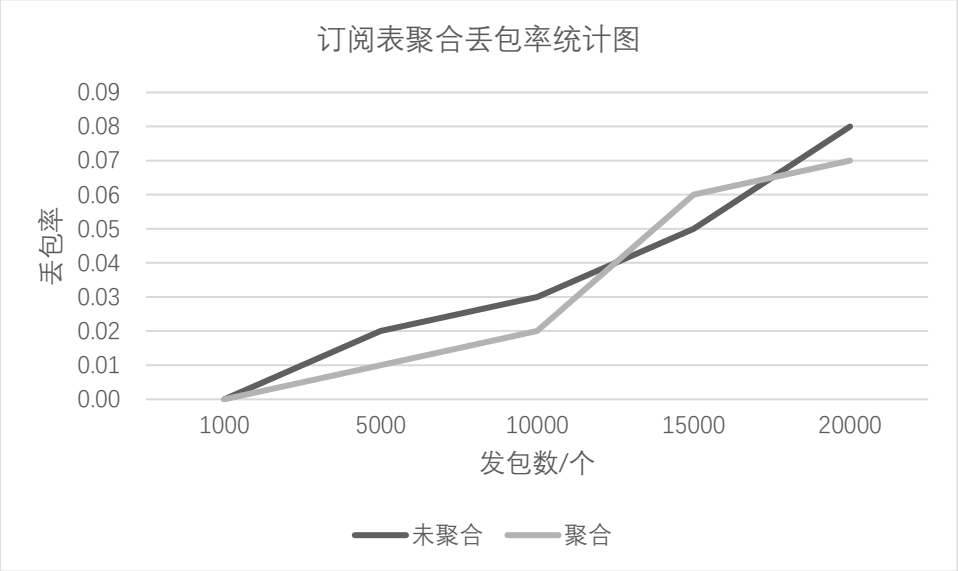

图 6-22 订阅表聚合丢包率统计图

由图可见丢包率依旧稳定在 8%左右，并没有随着转发效率升高而增大，可见系统性能是稳定的。

6.5 本章总结

本章对文中提出的设计和实现方案进行了测试，测试分为两部分，首先是针对各模块的功能进行了分块的测试，包括集群内拓扑的获取、集群间邻居关系的构建、订阅功能以及路由计算功能，保证了系统的正常可用。在此基础上针对系统的性能进行了性能测试，包括路由算法的性能以及消息在 SDN 网络中的转发效率，确保在集群数量增多、传输的数据量也增大的情况下，系统的性能依旧保持在相对稳定的水平上。实验表明基于 SDN 网络的发布/订阅系统的有效性，同时也验证了基于 Steiner 树思想的路由算法的性能。

第七章 总结和展望

本章对本文提出的基于 SDN 网络的发布/订阅系统中拓扑维护、订阅管理以及路由计算等几个模块的优化与实现工作进行了总结，同时对未来的工作进行了展望。希望通过已完成的工作，能够对未来的工作提供一些经验和帮助。

7.1 工作总结

本文所述的工作开始于 2015 年 12 月，最初一版基于传统网络的发布/订阅系统已经比较完善，但是在向 SDN 网络上迁移的时候，由于整体思路发生了变化，因此在修改的过程中走了很多弯路。最终决定在保留原先发布/订阅系统中核心功能的基础上，抛弃原有的代表/代理双层架构，转而设置控制器节点/普通节点双层架构，这样的划分使得节点的功能更偏向控制平面，而将转发平面的工作交给了 OpenFlow 交换机。同时针对包括 OSPF 协议、SDN 网络的基本特点、OpenFlow 协议的标准以及 OpenDaylight 控制器的应用在内的一系列内容，都进行了认真调研和学习。学习与设计的工作用去了开题后最初的几个月，期间在原定的方向上多次修改实现的具体方案，如从最初将单个交换机作为一个集群，到后面确定将是否归属同一个控制器管理作为划分同一集群的标准，以及在集群邻居间如何实现感知和构建等问题，都是在这期间经过大量调研学习后才做出的最终方案。

在拓扑维护方面，由于控制器可以直接检测到集群内的拓扑变化，因此代理节点无须在上线后注册，而只需由控制器节点定时向控制器进行轮询既可；而集群间的拓扑则是按照 OSPF 协议提出的 Hello 包机制进行探测，在建立邻接关系后进行 LSDB 的同步。针对订阅管理，由于转发平面依赖 OpenFlow 交换机以及它内置的流表，因此在针对订阅管理部分做优化时就必须兼顾流表的尺寸与匹配的准确率，为此对流表进行了分级设计，提出了多级流表匹配的方案，同时动态地对订阅表进行分裂和聚合操作，控制流表大小的同时也能够兼顾匹配命中率。在路由计算部分则是借鉴了 Steiner 树的思想，但并不每次都重新计算整棵树，而是在新增树枝时选取最优路径将树枝添加进去，最终获得一个次优的路由方案，之后根据路由方案生成流表并下发。

7.2 未来展望

本文主要论述了基于SDN网络的发布/订阅系统路由模块及其相关模块的优化与实现，在本系统给用户提供了在SDN网络上的松耦合、低时延、高可控的异步消息服务，但是本文系统也有可以改进和扩展的地方，主要有：

1. 完善策略控制模块。在目前的系统中，消息转发的规则全部基于订阅关系，但是在实际应用中，为了防止不同集群中消息互相干扰，同时预防恶意消息的破坏，因此在接下来的工作中可以进一步提高系统的可配置性和安全性，在防攻击方面做进一步的设计和实现。
2. 在消息接收过程中添加 Schema 校验。当前系统中传输的消息都是简单的字符串或者自定义的对象，而在未来为了能够适应更复杂的发布和订阅环境，添加Schema校验可以有效确保消息域中的消息格式正确。
3. 在真实环境中进行测试。由于现在的实验环境还是通过 OpenvSwitch 模拟 OpenFlow 交换机进行测验，尚未部署到实体的 OpenFlow 交换机上，因此可能存在很多兼容性的问题，未来应当在实体交换机上进行部署测试，以期获得更有说服力的数据，指导未来的优化工作。

参考文献

[1] 王丽君, 刘永强, 张健. 基于 OpenFlow 的未来互联网试验技术研究[J]. 电信网技术, 2011, 8(6): 1-4.

[2] Ng E. Maestro: A system for scalable openflow control[J]. Rice University, 2010.

[3] 何国锋. OpenFlow 在下一代数据中心网络的应用研究[J]. 互联网天地, 2013, 3(3): 71-74.

[4] Canini M, Venzano D, Perešíni P, et al. A NICE way to test OpenFlow applications[A]. // Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)[C]. 2012: 127-140.

[5] Gajic B, Riihijärvi J, Mähönen P. Intra-domain topology manager for publish-subscribe networks[A]. // Telecommunications (ICT)[C], 2011 18th International Conference on. IEEE, 2011: 394-399.

[6] Zhou L, Rodrigues J J P C. Service-oriented middleware for smart grid: Principle, infrastructure, and application[J]. IEEE Communications Magazine, 2013, 51(1): 84-89.

[7] 殷波, 张云勇, 王志军, 等. 基于 SDN 的数据中心网络技术研究[J]. 信息通信技术, 2015, 1: 007.

[8] 左青云, 陈鸣, 赵广松, 等. 基于 OpenFlow 的 SDN 技术研究[J]. 软件学报, 2013, 24(5): 1078-1097.

[9] 李程程, 王晓云. OpenFlow 技术与商业价值[J]. 软件, 2013, 34(12): 186-189.

[10] Pakzad F, Portmann M, Tan W L, et al. Efficient topology discovery in OpenFlow-based Software Defined Networks[J]. Computer Communications, 2016, 77: 52-61.

[11] Loukili A, Saucez D, Turletti T, et al. Content Distribution and OpenFlow: a Reality Check[A]. // IEEE Conference on Network Function Virtualization and Software Defined Networks[C], 2016: 54-75.

[12] Mueller P. OpenFlow: A new Paradigm for Campus Networks[J]. OpenFlow Technical Report ICSY, 2016: 25-59.

[13] Suh D, Jang S, Han S, et al. On performance of OpenDaylight clustering[A]. // NetSoft Conference and Workshops (NetSoft)[C], 2016 IEEE. IEEE, 2016: 407-410.

[14] Vu A V, Kim Y H. An implementation of hierarchical service function chaining using OpenDaylight platform[A]. // NetSoft Conference and Workshops (NetSoft)[C], 2016 IEEE. IEEE, 2016: 411-416.
[15] Pfaff B, Pettit J, Koponen T, et al. The design and implementation of OpenvSwitch[A]. // 12th USENIX symposium on networked systems design and implementation (NSDI 15)[C]. 2015: 117-130.
[16] Huang T, Yan S, Yang F, et al. Building SDN-Based Agricultural Vehicular Sensor Networks Based on Extended OpenvSwitch[J]. Sensors, 2016, 16(1): 108.
[17] Pan H Y, Wang S Y. Optimizing the SDN control-plane performance of the OpenvSwitch software switch[A]. // 2015 IEEE Symposium on Computers and Communication (ISCC)[C]. IEEE, 2015: 403-408.
[18] Vahdat A, Clark D, Rexford J. A Purpose-built Global Network: Google's Move to SDN[J]. Queue, 2015, 13(8): 100.
[19] Lara A, Kolasani A, Ramamurthy B. Network innovation using Openflow: A survey[J]. IEEE Communications Surveys & Tutorials, 2014, 16(1): 493-512.
[20] Kobayashi M, Seetharaman S, Parulkar G, et al. Maturing of OpenFlow and Software-defined Networking through deployments[J]. Computer Networks, 2014, 61: 151-175.

致谢

时间如白驹过隙，转瞬即逝，我在网络服务基础研究中心也已经度过了两年半的时光。在此，我想对所有在这两年半之中关心、鼓励、支持、陪伴和指导我的所有人表示最诚挚的感谢。

感谢我的导师陈俊亮院士。陈院士渊博的专业知识，踏实的科研态度，让实验室的老师与同学们都深受感染。陈院士虽已逾耄耋之年，却依然在科研工作的前线奋斗，这种为我国科研事业忘我奉献的执着与热情，让我们每个人都备受鼓舞，受益匪浅。

感谢在研究生阶段一直无私指导我学习与工作的章洋副教授。章老师对待工作与科研都尽职尽责，一丝不苟，这种专注的精神也深深地影响着我，让我在工作与学习中都保持着严谨与自律的态度。章老师在科研工作中也非常注重研究方向与实现细节的平衡，每周的例会，章老师都会认真听取同学们的发言，指出大家当前工作中存在的问题和不足，同时为大家点明未来研究工作的方向与重点，使我们在科研的道路上不至迷茫。同时章老师也有很高的学术造诣，在本文的成文过程中，章老师给予了我巨大的帮助，令我收获颇多。研究生阶段，章老师为我们付出了很多，在此，对章老师表示衷心的感谢。同时，也要感谢刘传昌老师、程渤老师、乔秀全老师以及商彦磊老师为实验室所做的巨大付出，他们在平日的工作学习中，为我们树立了良好的榜样。

感谢臧亚强、王兴、陈阳等学长学姐对我的照顾与帮助。在与他们的交流中，我对项目的认识才慢慢深入，直到最终完成方案的设计与实现。感谢牛琳琳、高翔、杨欣、周永江等同届学生，研究生阶段与他们一起共同科研学习、交流想法的日子，既丰富了我的思想，也充实了我的生活。感谢韩波、贺路路、李启波等下一届师弟，通过他们的新鲜想法，我也进一步完善了设计方案。感谢博士王亚丽学姐，她在 SDN 方向上的研究深度与广度令我叹服，正因为有了她的帮助，使得我在科研的道路上又迈出了新的一步。

感谢我的家人、朋友对我的支持和鼓励，他们是我在求职、毕设遇到困难之时的动力与寄托。

最后，真诚感谢各位参加评审和答辩工作的老师们在百忙之中抽出宝贵时间来评阅本文。您们的肯定是我努力的目标，更是我前行的动力。谢谢！

攻读学位期间发表的学术论文目录

[1] 刘昌威, 章洋. 基于 SDN 的发布/订阅中间件路由计算模块的优化与实现[EB/OL]. 北京：中国科技论文在线 [2016-12-06]. http://www.paper.edu.cn/releasepaper/content/201612-116.