第一届全国高校软件定义网络（SDN）创新应用开发大赛

基于 SDN 的园区级多媒体流量优化

参赛队： F. B.

（张健男、丁健、胡文博、汪硕、李将旭）

北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

目录

1 应用的简介和摘要

互联网出现于上个世纪七十年代，至今已经发展成为生活中不可缺少的基础设施。然而随着网络规模的爆炸性增长，互联网的发展面临越来越多的挑战，其中多媒体流量冗余是最为突出的问题之一，人们浏览静态网页、点播视频或者收看网络直播的时候，相同的内容在网络中重复传输，冗余流量消耗大量带宽资源，极大地影响了用户体验。利用 SDN 集中管控的优势，可以灵活调度多媒体流量的传输，从而降低网络中的冗余流量。本项目基于 SDN/OpenFlow 研究多媒体流量优化机制，设计了一套基于 Floodlight 控制器的园区级网络资源管理系统，试图解决多媒体流量传输时的冗余流量问题。

本项目研究内容如下：

提出一套基于 OpenFlow 实现的园区级多媒体流量优化方案，该方案分两个服务模块分别对多媒体流量中的实时流量和非实时流量进行处理：

①实时流量服务模块处理直播视频等实时业务流量，利用组播技术减少相同内容的重复传输，并根据业务需求以及网络负载状态，动态规划组播路径，消减冗余流量，节约网络带宽资源，从而提高用户体验。

②非实时流量服务模块处理浏览网页、点播视频等非实时业务流量，通过在控制器中定义不同的缓存使用规则和策略，实现服务质量可定义，为用户提供定制化服务。

本项目利用 Floodlight 控制器和 Open vSwitch 交换机模拟校园网络环境，通过实验验证多媒体流量优化方案。实验结果表明实时流量的冗余传输明显减少，网络效率大大增加，非实时流量的出口带宽消耗明显降低，从而证明本方案在传输过程中有效降低了多媒体的冗余流量。

2 应用场景介绍

本方案的动机来源于校园网用户访问多媒体时产生的诸多问题，因此本方案的主要应用场景是校园网等园区网络。对多媒体访问时，可能产生两种流量，一种是实时流量，例如，IPTV 视频直播。一种是非实时流量，例如，网页视频点播。我们的方案针对这两种多媒体流量进行优化、减少流量冗余，下面将针对两种多媒体流量分别介绍典型的应用场景。

2.1 场景 1：实时直播视频

现在越来越多的用户选择通过互联网的方式观看各种重大赛事直播，如世界杯期间，很多用户通过 IPTV 或者 CNTV 等收看世界杯实况直播，如果采用传统的点对点传输，会造成源服务器出口出流量过大。针对这种应用场景，我们的方案试图构建基于 SDN 的直播视频流方案来有效缓解视频源出口处的压力，改善用户体验。

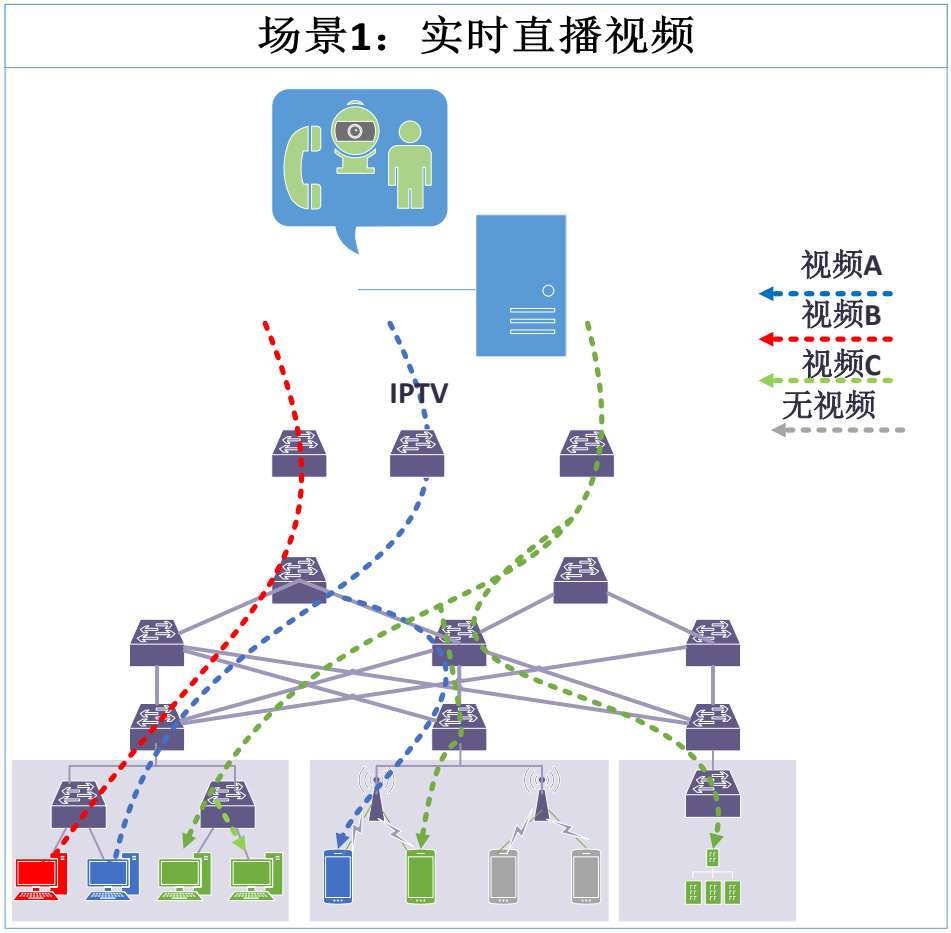

图 1 直播视频流

如上图所示，基于 SDN 的视频直播模块根据不同的直播视频内容来动态构建不同的组播树，如图中的蓝色、红色、绿色部分分别对应不同的用户加入了各自的直播组来收看视频。SDN 控制器能够掌控每个直播组都有哪些用户，从而动态调整图中的组播树。当用户加入直播组的时候，能够迅速收集到用户的加入请求，然后根据可灵活定义的组播树算法，及时调整底层交换机的转发策略，从而保证用户能够正常收看节目。当用户离开直播组，停止收看时，能够及时发现用户的离开请求，从而关闭对相应端口的转发，节约链路的带宽。

2.2 场景 2：网页点播视频

当用户在诸如优酷等视频网站上看热点新闻节目或者热播电视剧的时候，由于观看用户数量很多，而每个用户都需要从远端视频网站的服务器上下载视频，所以网络中会产生大量的重复传输的视频流量。另外，不同用户可能会有不同的优先级。如果能在局域网内部署灵活可定义的缓存服务则可以在很大程度上缓解出口网关处的压力，减少出口流量，同时给不同优先级用户提供不同的点播体验。我们的方案针对这种应用场景试图在园区网内建立一种基于 SDN 的缓存策略来灵活定义并管理缓存资源，从而减小冗余传输的多媒体流量。

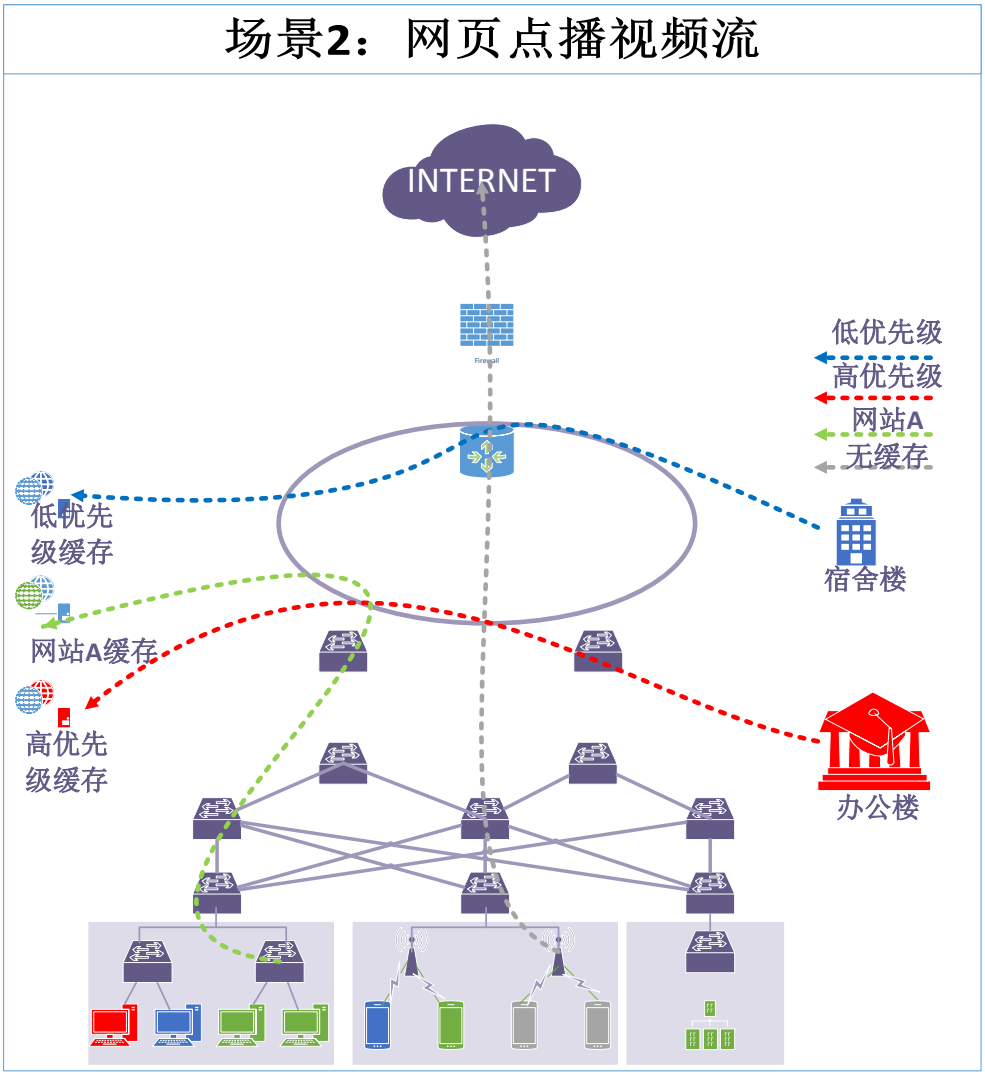

图 2 网页点播视频流

以校园网举例，如图 2 所示，作为校园网的管理者，首先要保障的是办公、科研区域的访问速度，其次是学生宿舍楼。基于 SDN 的视频点播策略可以让管理员根据预定义规则指定办公楼使用高优先级的链路与缓存（如图中红色部分），宿舍楼使用低优先级的链路与缓存（如图中蓝色部分）。此外，还可以根据用户

访问的不同资源来定义使用的缓存资源，例如可以为访问观看学习视频的用户提供单独的缓存资源（如图中绿色部分），对其他一些娱乐资源不提供缓存服务（如图中灰色部分），从而使用户获得更好的体验。基于 SDN 的点播视频流量策略不仅能为用户提供点播视频的加速，而且管理灵活，可以做到每个楼、每个交换机、每个端口的控制。如同一接入交换机上的不同用户可以通过指定使用不同的缓存而带来不同的点播视频体验。

3 方案特色和创新

本方案主要使用 SDN 技术对多媒体流量进行了冗余优化，考虑到多媒体流量根据特征可以分为实时流量和非实时流量，因此我们在这两个方面分别做出了创新优化，具体如下：

1. **提出了灵活敏捷、易配置的组播树方案来优化实时流量：**相比传统互联网中部署复杂、不灵活的组播树方案，本文提出的基于 SDN 的组播方案可快速感知用户/链路负载状况，并基于感知信息敏捷地对组播树及其策略进行调整。另外，传统组播技术难以配置维护多个组播树，集中式的组播树策略调整大大简化了对网络节点的配置，具有更好的可扩展性。

2. **设计了缓存能力可定义的方案来优化非实时流量：**针对用户对网络服务质量的不同需求，本文提出了基于 SDN 的可以灵活定义网络缓存策略的方案，此方案可以方便地运行管理员指定的缓存规则，从而为用户提供差异化的缓存能力。

4 应用具体设计论述

4.1 背景

校园网中的网络流量具有非常明显的“潮汐效应”。白天课业期间，网络较为空闲，出口带宽充裕；晚间时段，网络用户增多，出口流量暴涨，无法继续提供高质量的网络服务，如出现打开网页速度慢、点播视频节目出现卡顿等网络阻塞现象。另外，在一些重大体育赛事期间，如世界杯、奥运会、NBA 总决赛，通过校内 IPTV 服务收看直播的用户数目急剧增加，造成内部带宽消耗严重，甚至造成部分链路堵塞，不仅直播质量无法保障也影响其他网络业务的正常使用。

由此可以看出，如何在校园网中对上网高峰时期的多媒体流量进行优化是一个比较突出的问题。这里的多媒体流量包括①收看 IPTV 直播等实时流量，②浏览网页、点播视频时产生的非实时流量。如果能够有效地解决多媒体流量冗余传输的问题，就能给用户提供更好的上网体验。

为了解决这个问题，从实时流量角度来看，组播传输中组播数据包在组播源和组播组成员之间实现一对多的网络传输方式。当多个接收者请求接收发送者发送的组播数据包时，需要在路由器或者交换机等转发设备进行数据包的复制，然后发送给不同的组播用户。组播传输模式能在一定程度上能够节省带宽，降低组播中心网络负载压力、减少出现网络阻塞和发送延时的可能性。

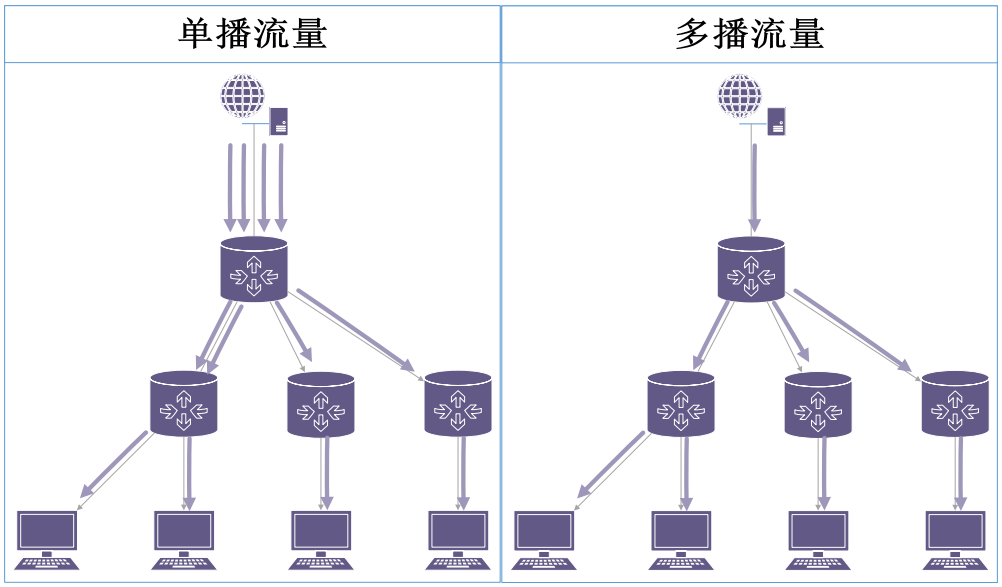

图 3 单播组播对比

如图 3 所示，通过组播技术，源站只需要传输一份流量就可以发送给所有的用户终端了，从而可以有效的减少网络中的冗余流量。

从非实时流量角度来看，本方案从用户获取内容的角度分析，用户的大部分访问都是当时的热点内容，如：浏览热点新闻、收看热播的电视剧、或者收看在线直播节目。传统网络提供端到端服务，即使同一地区的不同用户请求同一内容，也会把相同的内容反复传输到同一地区的用户，这就造成了流量冗余现象。流量冗余表现最突出的就是视频业务流，一方面视频业务消耗较多的带宽资源，另一方面视频流量对传输时延要求较高。所以如果能够解决视频业务的冗余传输可以有效改善现有的网络体验。

在现有网络中解决流量冗余的主要手段是各种缓存技术，包括正向代理、反向代理、透明缓存等。举例来说，CDN 系统使用的反向代理技术的基本思路是：用户请求资源时，通过 DNS 返回代理服务器地址，代理服务器将所请求资源回传或经由平台获得内容提供商原始资源。

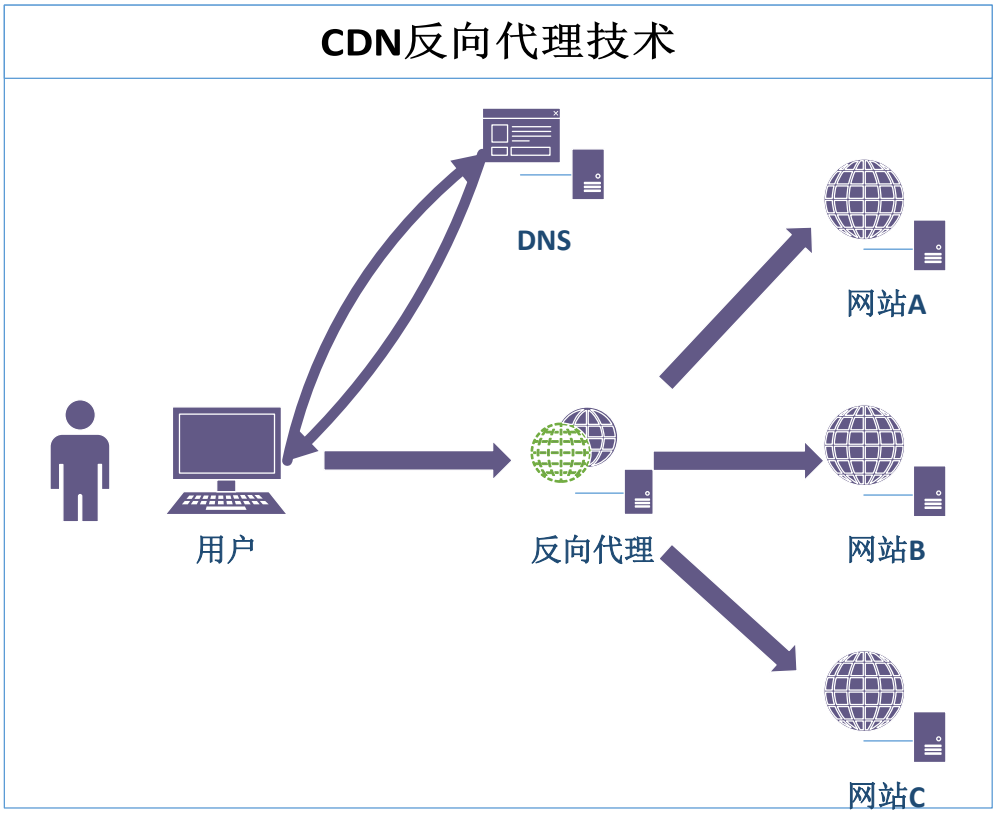

图 4 CDN 反向代理

如图 4 示，用户首先通过 DNS 查询目的网站的 IP 地址，而 DNS 服务器返回代理服务器的 IP 地址，从而用户发送请求给反向代理服务器，然后反向代理服务器再代理访问网站 A、B、C。

透明缓存技术使用的基本技术思路是：在园区网内或者网关节点部署缓存服务器，通过配置网关路由或者用户手工配置等方法，把所有的用户上网流量都发送给透明缓存服务器，然后透明缓存服务器代理用户请求资源。

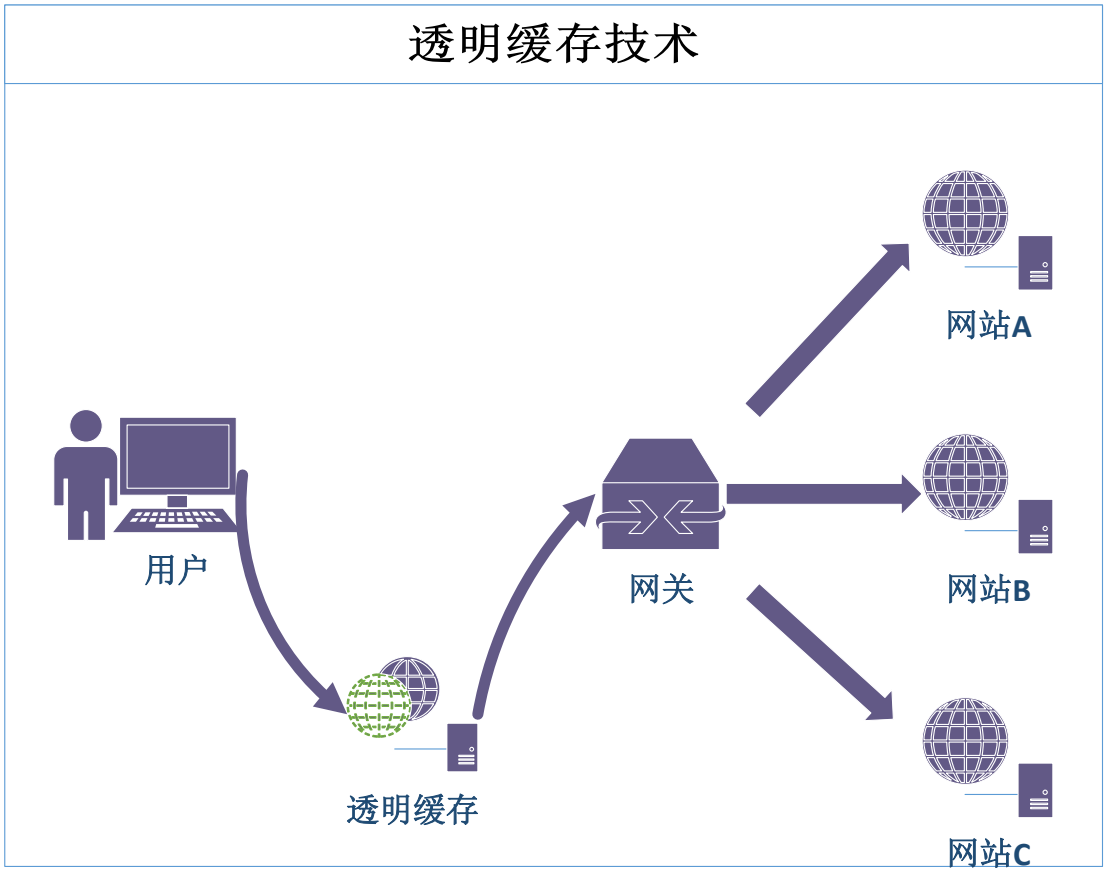

图 5 透明缓存

如图 5 所示，用户通过手工配置的方式，所有的上网流量都会先发送给透明缓存服务器，然后透明缓存服务器再通过网关代理用户去请求 A、B、C 等网站的资源。

4.2 研究问题

虽然使用传统的组播技术可以有效地解决实时业务流量冗余问题，利用 CDN 技术可以优化非实时业务流量，但是它们也存在一些缺陷：

- IP 组播体系结构可扩展性差。路由器需要维护每个组播组的状态信息，随着网络中并发的组播组数量增加，网络中大量的组播信息记录需要路由器巨大的存储和处理开销。组播组成员的动态变化要求网络必须动态维护路由状态，更增加了组播路由器的处理开销，并且转发组播数据包需要更长的处理时间。
- 组播数据流的计费方式难以确定，网络运营商之间的利益取向不同。当前的组播模式没有支持组播的付费体系，要想在当前的服务模式和协议体系结构下普遍化和商业化，IP 组播会遇到很多困难。
- 透明缓存或者反向代理等方式造成了网络流量的不透明，内容提供商无法把握运行状态，难以根据流量制定策略。

- 透明缓存或反向代理因为无法获取全局网络拓扑、感知网络状态信息，并且缺乏控制路由的手段与能力、QoS 保障，从而难以为用户提供差异化的网络服务。

4.3 解决方案

利用 SDN 集中管控的特性，控制器能够获得全网拓扑以及实现流量优化控制，从而有效地补充了组播技术和 CDN 技术的不足，提升内容分发能力、改善用户体验。因此，本方案基于 OpenFlow 协议并结合组播和 CDN 缓存技术针对多媒体服务中实时业务和非实时业务两类流量分别进行处理，从而达到优化全网多媒体流量传输的目的。

4.3.1 实时业务流量

1. 概述

由于直播等实时业务流量都是动态、时延敏感的，这部分流量是在保证实时性的前提下进行缓存，因此可以利用组播的方法解决实时业务流量的冗余问题。

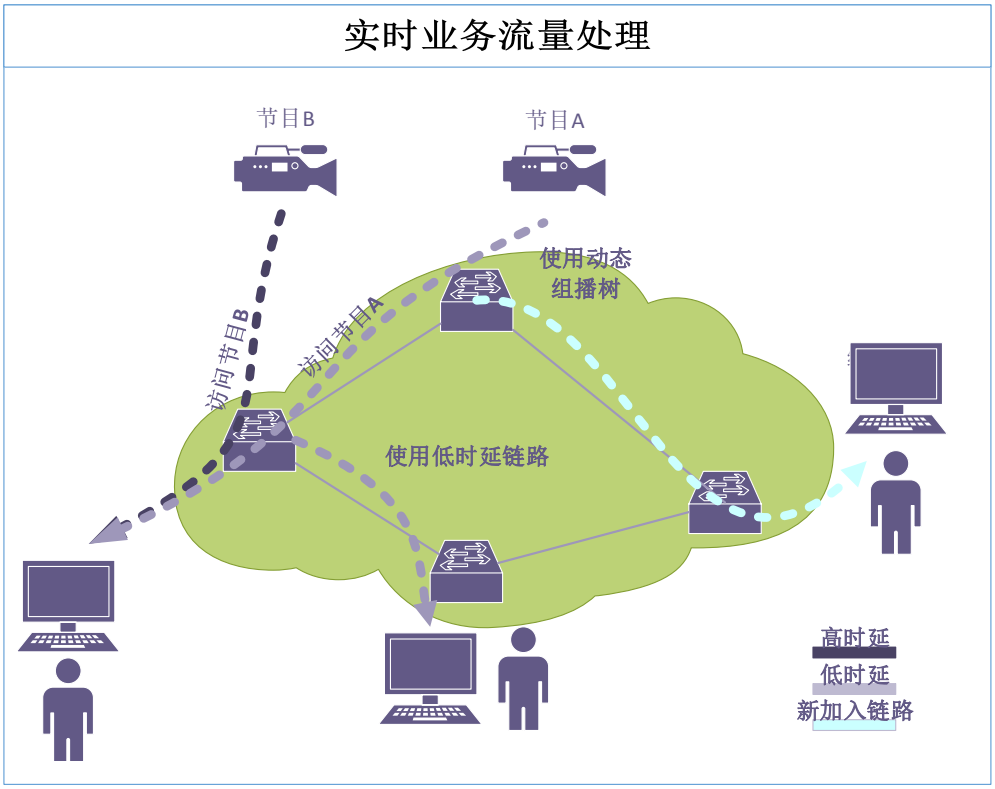

图 6 实时业务流量处理

基于 SDN 的实时业务流量组播技术可以对不同的节目选择不同质量的链路

进行访问，如图 6 中节目 B 的优先级低，节目 A 的优先级比较高，此时就可以为节目 A 选择时延较低的链路，节目 B 选择延迟相对高的链路进行转发。此外还可以对组播树进行优化，当有新的用户请求观看直播时，能够动态调整组播树的拓扑，尽量减少需要调整的网络节点数目，使整体用户的体验最好。

2．设计方案

通过 OpenFlow 信令控制实时业务流量的流向，处理过程如下图所示：

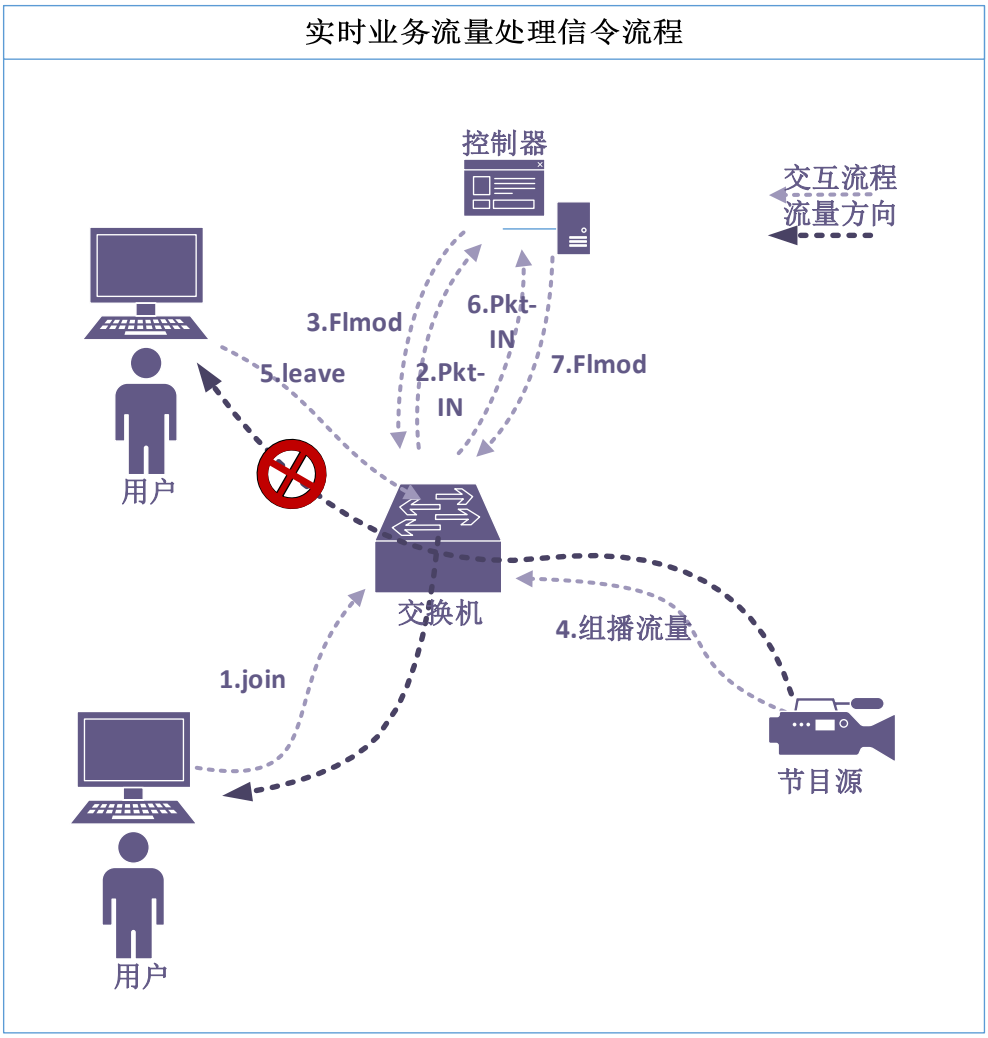

图 7 实时业务流量处理信令过程

用户请求收看直播时，用户向网络中发送 IGMP：join 命令来加入视频的组播组。

1. 由于交换机没有匹配的流表项，join 命令通过 Packet-in 上交给控制器。
2. 控制器收到 join 命令，将终端加入组播组，并且动态地计算组播树和要修改的路由信息，并通过 Flowmod 消息配置流表项。。
3. 当流表项生效后，从节目源来的实时直播流量就依据组播树进行转发。
4. 用户停止收看节目时，用户发送 leave 消息。

5. 由于交换机没有匹配的流表项，leave 命令通过 Packet-in 上交给控制器。
6. 控制器将用户从组播组中移除，并通过 Flowmod 消息配置流表项，中断组播源到该用户流量的转发。

4.3.2 非实时业务流量

1．概述

由于非实时业务流量是可以被缓存的，所以结合缓存技术，利用代理服务器对内容进行缓存，从而减少网络出口的冗余流量。

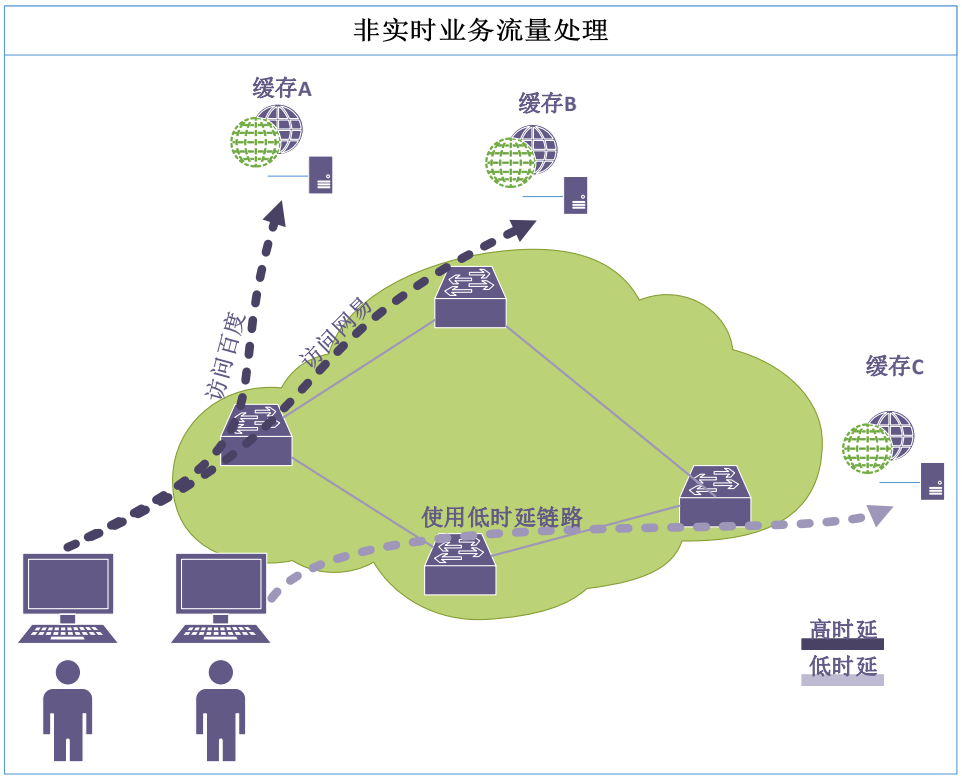

图 8 非实时流量处理方案

如图 8 所示，基于 SDN 的非实时业务流量处理可以在网内部署多个缓存来提供服务。控制器可以根据目的网站、链路状态、接入端口、接入交换机、用户级别等细粒度的参数来选择使用的缓存结点，从而为更好的对静态内容进行缓存，为用户提供更好的体验。

2．设计方案

通过 OpenFlow 信令来完成非实时业务流量的处理，处理过程如图 9 所示：

当用户通过 HTTP 请求访问网页或点播等非实时资源时，首先请求获得网关的 MAC 地址，然后向网关发送 TCP 数据包。

1. 当数据包到达交换机后，没有匹配的流表项，则请求数据包通过 Packet-in

上交给控制器。

2. 控制器收到 Packet-in 消息后，按照设定的缓存规则确定对应的缓存地址，并且下发 Flowmod 消息添加流表项，通过该流表项将请求数据包重定向至缓存节点。
3. 缓存节点收到请求后，查看所请求资源是否已缓存：若已缓存，则将内容直接返回给用户；若无缓存内容，需要先从源站获取内容，然后将其缓存并返回给用户。
4. 缓存节点发送返回数据包给用户，源 IP 和源 MAC 地址均为缓存节点的地址。
5. 交换机没有匹配的流表项，所以将返回数据包通过 Packet-in 上交给控制器。
6. 当控制器收到 Packet-in 后下发 Flowmod 消息添加流表项，修改数据包的源 IP 和 MAC 地址为原始请求的目的 IP 和 MAC 地址。
7. 交换机将修改后的返回数据包发送给用户，这样从用户端的角度看访问网络的流程没有改变，整个处理过程对用户是透明的。

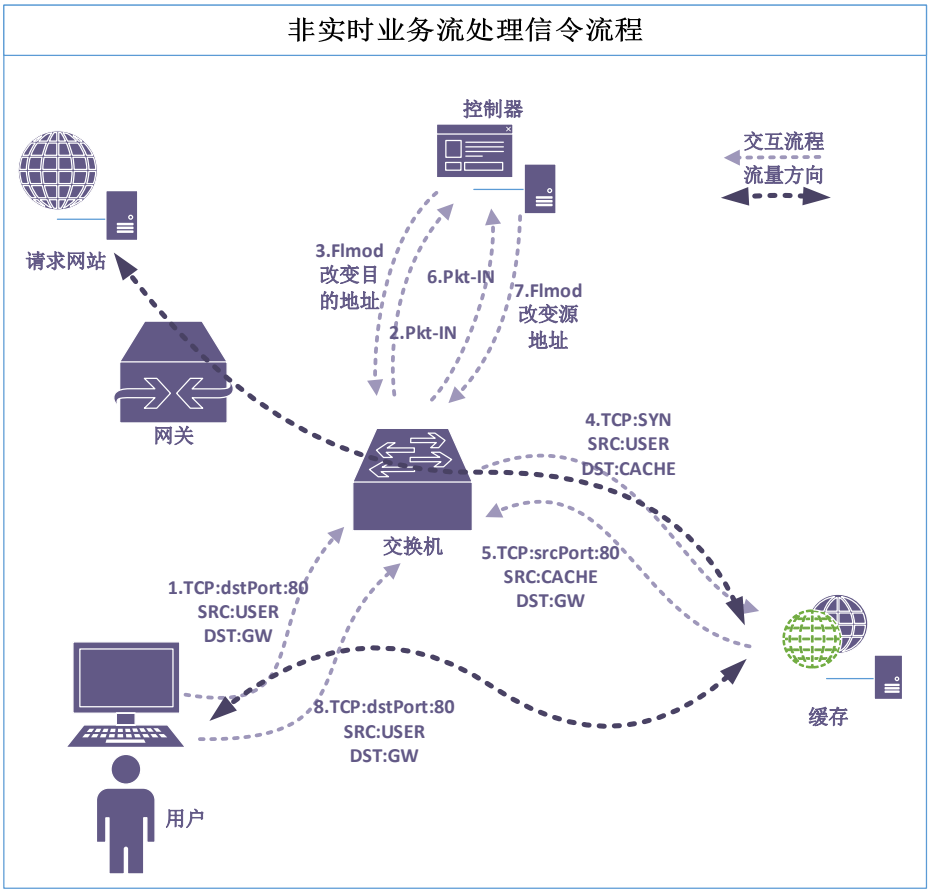

图 9 非实时流处理信令处理流程

本方案基于 OpenFlow 协议，根据管理员的定义和配置，可以按照当前网络状态，选择一个满足要求的缓存节点给用户使用。通过这种方法不仅节约了出口

带宽，消除了网络中的非实时业务冗余流量，而且减少了特定链路阻塞的可能，改善了用户的体验。

4.4 系统架构

前端 Web 管理界面和后端 SDN 控制器功能模块是本方案的主要组成部分。管理员可以通过前端 Web 页面查看全局拓扑、交换机状态、转发路径等信息，并可以配置缓存及其规则。后端控制器主要基于 Floodlight 控制器开发，控制器版本为 V_0.9 版本，南向接口基于 OpenFlow 1.0 协议。本方案对实时业务流量和非实时业务流量采用不同的处理方法，在 Floodlight 内分由两个不同模块来实现，这两个模块通过 RestAPI 提供查询和配置功能。

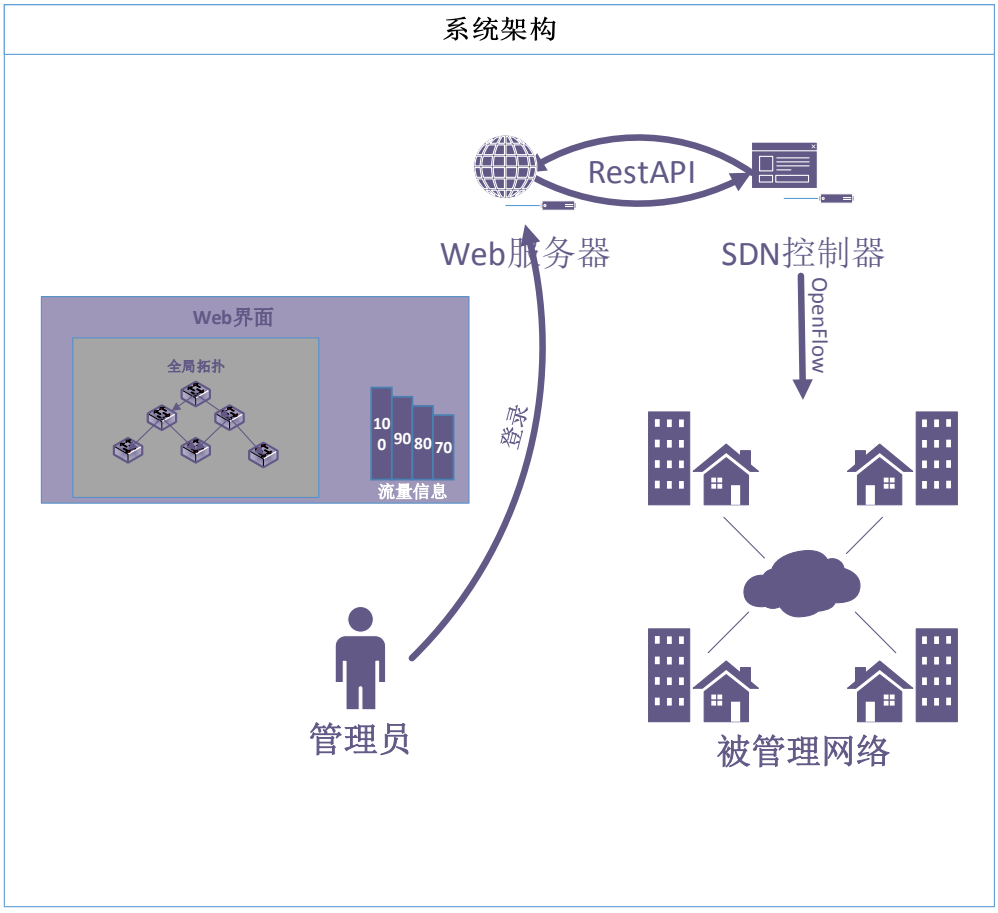

图 10 系统架构

如图 10 所示，管理员首先登陆 Web 管理界面，在界面中可以看到网络的全局拓扑和流量信息，然后管理员根据用户的需求，①在 Web 管理界面上配置需要提供优化的流量类型或者用户信息；②配置完成以后，Web 服务器根据管理员的配置信息，通过 RestAPI 命令下发控制指令到 SDN 控制器；③SDN 控制器通过实时/非实时业务功能模块的处理，将控制指令转换为 OpenFlow 信令并下发给底层的交换设备。这样管理员可以通过 Web 管理界面管理域内的网络。

5 应用实现过程

5.1 模块设计

5.1.1 依赖模块

本方案使用 Floodlight 控制器，它是由 Big Switch Network 公司主导的基于 Java 语言开发的 SDN 控制器，旨在为企业提供 SDN 应用解决方案，其代码都通过了专业化测试，具有丰富的功能集以及较高的性能和可靠性。

Floodlight 使用模块化架构开发，模块加载系统定义了模块的基础实现接口 IFloodlight。在功能上，Floodlight 由控制器模块和应用模块组成，控制器模块实现了核心的网络服务并为应用程序提供接口，应用模块根据不同的目的实现不同的方案。主要包括：

1) 控制器模块:

FloodlightProvider(Dev)

DeviceManagerImpl(Dev)

LinkDiscoveryManager(Dev)

TopologyService(Dev)

RestApiServer(Dev)

ThreadPool(Dev)

MemoryStorageSource(Dev)

Flow Cache(API only)

Packet Streamer

2) 应用模块:

VirtualNetworkFilter

Forwarding

Firewall

Port Down Reconciliation

表 1 对各个模块进行了简要的介绍：

表 1 Floodlight 各模块简介

Floodlight 模块简介表	
模块	简介
FloodlightProvider	FloodlightProvider 提供了两组重要的功能。它负责控制器与交换机之间的链接，并且把 OpenFlow 消息转化为其它模块可

	以监听的事件。第二个功能是决定 OpenFlow 消息事件监听模块的顺序，事件处理模块可以让消息转发给下一个处理模块或者停止处理。
DeviceManagerImpl	维护 OpenFlow 网络中的设备信息。
LinkDiscoveryManager	发现和维护 OpenFlow 网络中的链路状态。
TopologyService	拓扑服务为控制器维护拓扑信息，也负责计算网络转发路径。
RestApiServer	RestApiServer 通过 HTTP 协议提供 REST API。
ThreadPool	该模块包装了 java 中的 ScheduledExecutorService，它可以在指定的时间运行一个线程，也可以用来周期性的执行一个线程。
MemoryStorageSource	MemoryStorageSource 是一个在内存中的 NoSQL 存储，在数据改变时发出通知。
Packet Streamer	该模块是一个数据包流服务，使用此项服务可以让任何交换机、控制器和它的观察者之间有选择的交换数据。它由两个接口组成：①一个基于 REST 的接口，该接口定义了它所感兴趣的 Openflow 消息的特征，称之为过滤器；②一个基于 Thrift 的数据包过滤器
VirtualNetworkFilter (Quantum Plugin)	VirtualNetworkFilter 模块是一个基于二层的简单虚拟化网络，可以使用它在一个二层的域中建立多个逻辑的二层网络，该模块可以独立使用也可以结合 OpenStack 使用。
Forwarding	Forwarding 在两个设备之间转发数据包，IDeviceService 会把发送设备和接收设备分类。
Firewall	防火墙应用程序实现了一个 floodlight 模块，该模块通过检测 Packet-In 消息使用流表项在 OpenFlow 交换机上实现 ACL 策略。ACL 是一系列允许或者拒绝接入交换机上的流量的条件。每个数据流中的第一个数据包都会触发一个 packet-in 事件。防火墙模块按照规则优先级排序，并按照 Openflow1.0 标准中规定的 OFMatch 和 Packet-In 中的头字段进行匹配。优先级最高的匹配将决定处理流的动作(允许/拒绝),也可以使用 OFMatch 中定义的通配符。

如图表所示本方案主要使用 FloodlightProvider、DeviceManagerImpl、TopologyService、RestApiServer、MemoryStorageSource 等模块来帮助我们实现本方案的主要功能。

5.1.2 实时业务处理模块

1．数据结构

本模块需要处理并维护组播组的信息，需要组播信息表来记录组播组的信息。当终端加入组播组的时候更新组播信息表，添加终端信息。当终端离开组播组的时候，更新组播信息表，并删除终端信息。

表 2 组播信息表

组播信息表	
字段	含义
组播组 IP	主键，组播组对应的组播 IP 地址
终端 IP 集合	记录本组播组中的所有终端的 IP 地址

同时还需记录每个组播组的组播树的情况，所以需要转发信息表来记录每个组播树的情况，方便对组播树进行修改，动态的调整组播树。

表 3 转发信息表

转发信息表	
字段	含义
组播 IP	主键，记录了终端所加入的组播组
交换机信息集合	记录了所以转发路径上的交换机，其中记录了 DPID,流量入端口，出端口集合信息

根据上述的转发信息表，控制器可以生成具体的流表项下发到交换机当中，完成实时业务流量的转发。

2．处理流程

如图 11 所示，本模块首先提取出数据包中的信息，然后判断数据包的类型。若为终端加入组播组的消息，则执行相应的流程从新动态计算组播树，并更新流表项。若为终端离开组播组的消息，则执行相应的流程，把终端信息从组播地址映射表中删除，然后更新流表项。

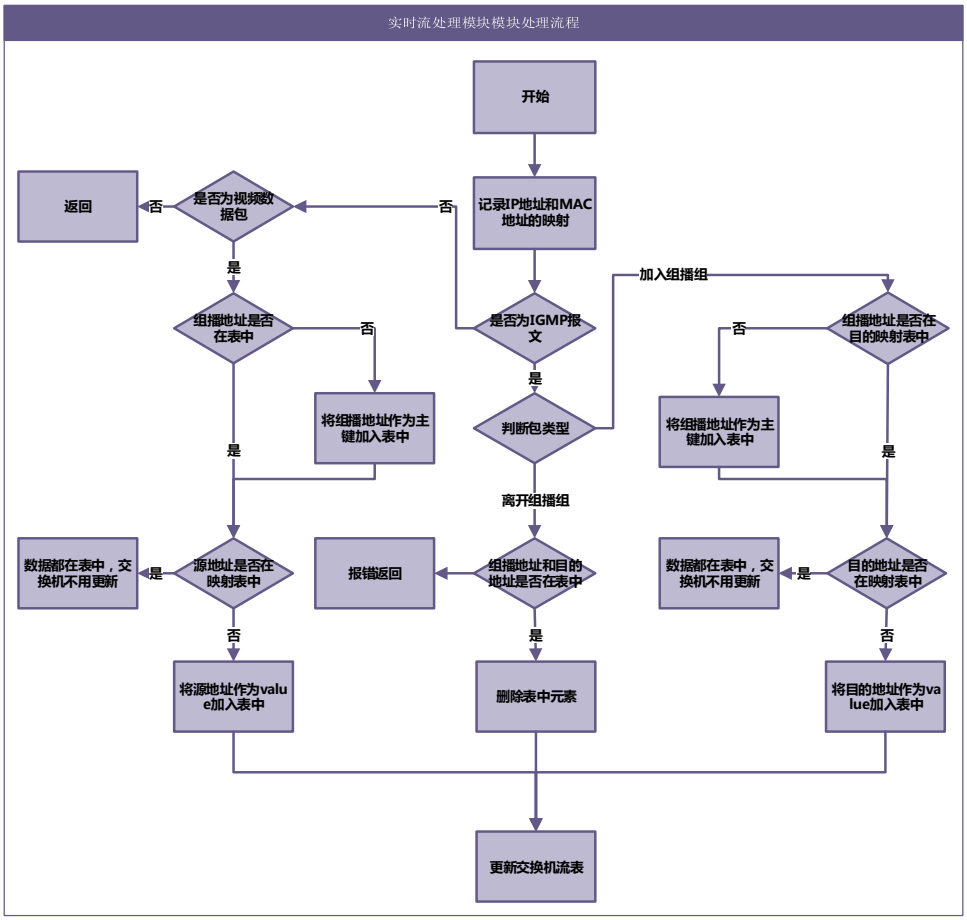

图 11 实时流处理模块流程图

3. 函数说明

IMuticastService 定义了实时业务处理模块对外提供的接口函数，如下：

```
public interface IMuticastService extends IFloodlightService {
    ConcurrentHashMap<Integer, ConcurrentHashMap<Long,
ConcurrentHashMap<Short, Set<Short>>>> getLinks();}
```

IGMPProcess 实现了 IMuticastService 接口

```
    public class IGMPProcess implements IFloodlightModule, IOFMessageListener,
IMuticastService
```

在 IGMPProcess 中 receive 负责执行对 Packet-in 消息的处理。

```
public net.floodlightcontroller.core.IListener.Command receive (
            IOFSwitch sw, OFMessage msg, FloodlightContext cntx)
```

在 receive 中 processIgmp 完成对终端消息的处理，processSource 完成对源服务器流量的处理。定义如下

```
    private void processIgmp(IOFSwitch sw, OFPacketIn pi,
            FloodlightContext cntx, IPv4 ipv4)
```

```
    private void processSource(IOFSwitch sw, OFPacketIn pi,FloodlightContext
cntx, IPv4 ipv4)
```

在 processIgmp 中调用 writeRoute 函数用来完成具体的组播树计算和下发流表的任务，定义如下：

```
    private void writeRoute(int muticastAddress,FloodlightContext cntx)
```

4. RestAPI

REST API					
URI	方法	URI 参数	数据	数据说明	说明
/wm/muticast/links/json	Get	无	无	无	返回所有组播组的组播树信息

5.1.3 非实时业务处理模块

1. 数据结构

本模块定义一张规则匹配表来记录用户定义的匹配规则，从而根据用户定义的规则来判断流量对应的缓存。规则匹配表各字段含义如下：

表 4 规则匹配表

规则匹配表	
字段	含义
Dpid	交换机 DPID
In_port	交换机接收端口
Dl_src	源 MAC 地址
Dl_dst	目的 MAC 地址
Dl_type	链路类型
Nw_src_prefix	源 IP 地址掩码长度
Nw_srcmaskbits	源 IP 地址
Nw_dstPrefix	目的 IP 地址掩码长度
Nw_dst_maskbits	目的 IP 地址
Nw_proto	网络层协议
Tp_src	传输层源端口
Tp_dst	传输层目的端口

ProxyIP	缓存 IP 地址

当发现有数据包匹配用户所定义的规则后，需要修改数据包的源地址或者目的地址，所以需要转发信息表来记录原始数据包的信息。转发信息表如下。

表 5 转发信息表

转发信息表	
字段	含义
Src-IP	数据包的源 IP 地址，为此表的主键
Src-Port	数据包的源端口
Dst-IP	原始数据包的目的 IP 地址
Dst-Port	原始数据包的目的端口

当控制器查找到数据包满足匹配规则后，会自动生成一条转发信息表的条目，把原始的数据包信息记录在转发信息表中，然后控制器修改原始数据包的目的 IP 和目的 MAC 为缓存节点的 IP 和 MAC 地址。当数据包从缓存发回终端时，控制器根据转发信息表，恢复数据包的信息为原始数据包的信息。

2．处理流程

如图 12 所示，首先控制器收到交换机的 Packet-in 消息后，提取出数据包中的目的地址和源地址。然后控制器判断该数据包是否为从缓存节点发出的返回数据包，如果不是从缓存发出来的包，说明这是由用户终端产生的原始数据包，控制器根据匹配规则对数据包进行处理。若是从缓存发出来的包，说明该数据包是发给用户终端的返回数据包，需要修改源 IP 字段。

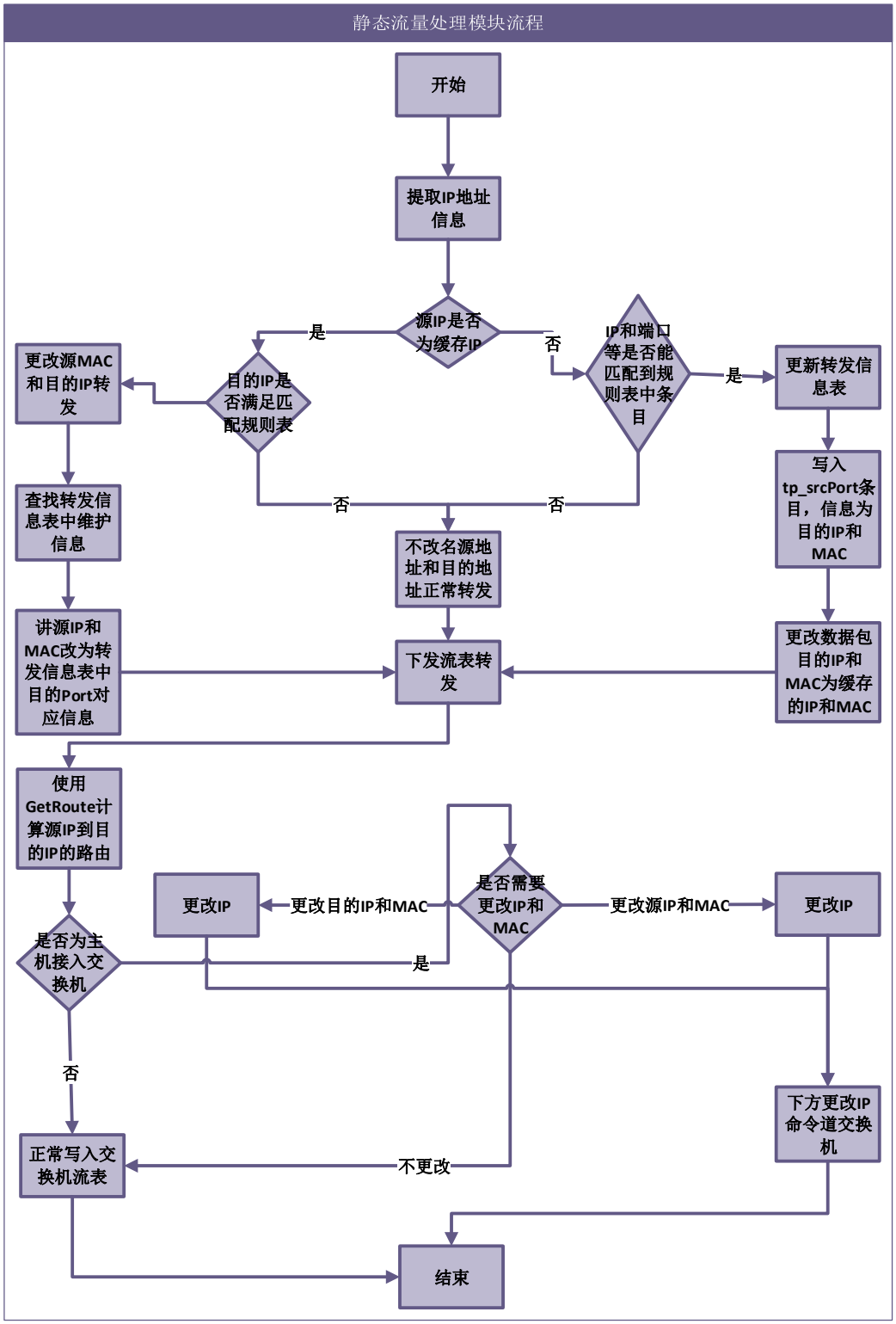

图 12 非实时流量处理模块流程图

3．函数说明

在 IProxyCacheServer 接口中定义如下函数，对外提供配置服务。其他模块可以调用对应的函数对非实时业务处理模块完成配置。

```
public interface IProxyCacheServer extends IFloodlightService {
    void addCache(int cache);
    void deleteCache(int cache);
    Set<Integer> listCache();
    void addProxy(int host,int cache);
    void deleteProxy(int host,int cache);
    ConcurrentHashMap<Integer, Integer> listProxyPair();
    Integer listProxy(int host);
    IpWithType getHostLink(int host);
    List<FirewallRule> getRules();
    void addRule(FirewallRule rule);
    void deleteRule(int ruleid);
    boolean checkCache(int ip);
    IpWithType getHostLinkWithRule(String src, short srcport, String dst,
            short dstport);
    }
```

同时 ProxyCache 类实现了 IProxyCacheServer 接口和其它主要的模块接口函数，定义如下所示。

```
public class ProxyCache implements IProxyCacheServer, IFloodlightModule,
        IOFMessageListener, IOFSwitchListener
```

在 ProxyCache 中 receive 负责执行对 Packet-in 消息的处理。

```
public net.floodlightcontroller.core.IListener.Command receive (
            IOFSwitch sw, OFMessage msg, FloodlightContext cntx)
```

receive 函数调用 doForwardFlow 函数来完成对路由和转发路径的计算，其中 changeSrc 和 changeDst 标明是否要改名源地址和目的地址。定义如下：

```
protected void doForwardFlow(IOFSwitch sw, OFPacketIn pi,
           FloodlightContext cntx,
               boolean changeSrc,boolean changeDst,int hostIP,int cacheIP)
```

doForwardFlow 函数通过 PushRoute 函数来生成 Flowmod 消息并下发给底层的交换机。定义如下：

```
    public boolean pushRoute(Route route, OFMatch match,
           Integer wildcard_hints,
           OFPacketIn pi,
           long pinSwitch,
           long cookie,
           FloodlightContext cntx,
           boolean changeSrc,
           boolean changeDst,
           short   flowModCommand,
```

```
            int hostIP,
            int cacheIP)
```

4. RestAPI

管理员需要对非实时业务处理模块进行配置，本模块提供了较多的 RestAPI，见下表所示：

表 6 静态流处理模块 REST API

REST API					
URI	方法	URI 参数	数据	数据说明	说明
/wm/proxycache/addproxy/{ip}/json	Get	ip：缓存 IP 地址	无	无	添加缓存
/wm/proxycache/delproxy/{ip}/json	Get	ip：缓存 IP 地址	无	无	删除缓存
/wm/proxycache /listproxy/json	Get	无	无	无	返回已添加的缓存
/wm/proxycache /json	POST	无	{"<field 1>":"<value 1>", "<field 2>":"<value 2>", ...}	"field":"value" pairs below in any order and combination: "switchid":"<xx:xx:xx:xx:xx:xx:xx:xx>", "src-inport":"<short>", "src-mac": "<xx:xx:xx:xx:xx:xx>", "dst-mac": "<xx:xx:xx:xx:xx:xx>", "dl-type": "<ARP or IPv4>", "src-ip": "<A.B.C.D/M>", " dst-ip": "<A.B.C.D/M>",	添加匹配规则

				"nw-proto": "<TCP or UDP or ICMP>", "tp-src": "<short>", "tp-dst": "<short>", "priority": "<int>", "proxyIP": "<A.B.C.D>"	
/wm/proxycache /json	DELETE		{"<ruleid>":"<int>"}	"**ruleid**": "<int>"	删除规则
/wm/proxycache /getlinks/{src}/{srcport}/{dst}/{dstport}	GET	src：源 IP。dST：目的 IP。Srcport 源端口。dstport 目的端口	无	无	用于显示终端所发送的一个数据包的真实转发路径。返回真实的路径信息。

5.2 实验设计

5.2.1 拓扑搭建

本应用采用以下拓扑进行验证

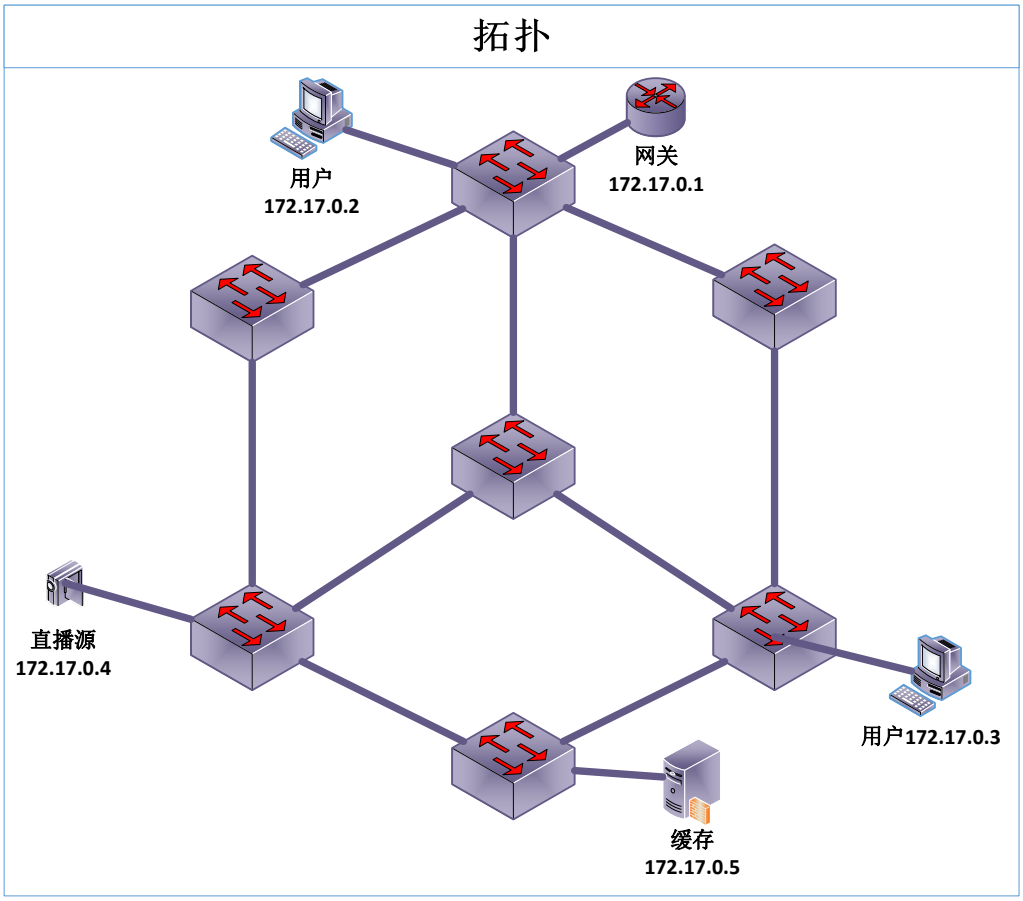

图 13 验证拓扑

如图 13 所示，缓存节点 IP 地址为 172.17.0.5，直播视频源 IP 地址为 172.17.0.4，两个用户终端 IP 地址分别为 172.17.0.2 和 172.17.0.3，网关 IP 地址为 172.17.0.1。

5.2.2 实验流程

1. 实时流量优化验证

1）搭建视频源服务器

在视频源终端使用 VLC 播放器新建一个地址为 224.0.0.3，端口为 5004 的串流，如图 14 所示：

图 14 新建串流

2）播放直播视频

在用户终端处，使用 VLC 同时打开该组播地址的串流：

图 15 打开串流

2．非实时业务流量优化验证

我们来验证对非实时业务流量的优化，证明终端可以通过管理员定义的规则通过缓存来访问 web 资源，实验步骤如下：

1）通过 web 配置界面配置缓存为 172.17.0.5

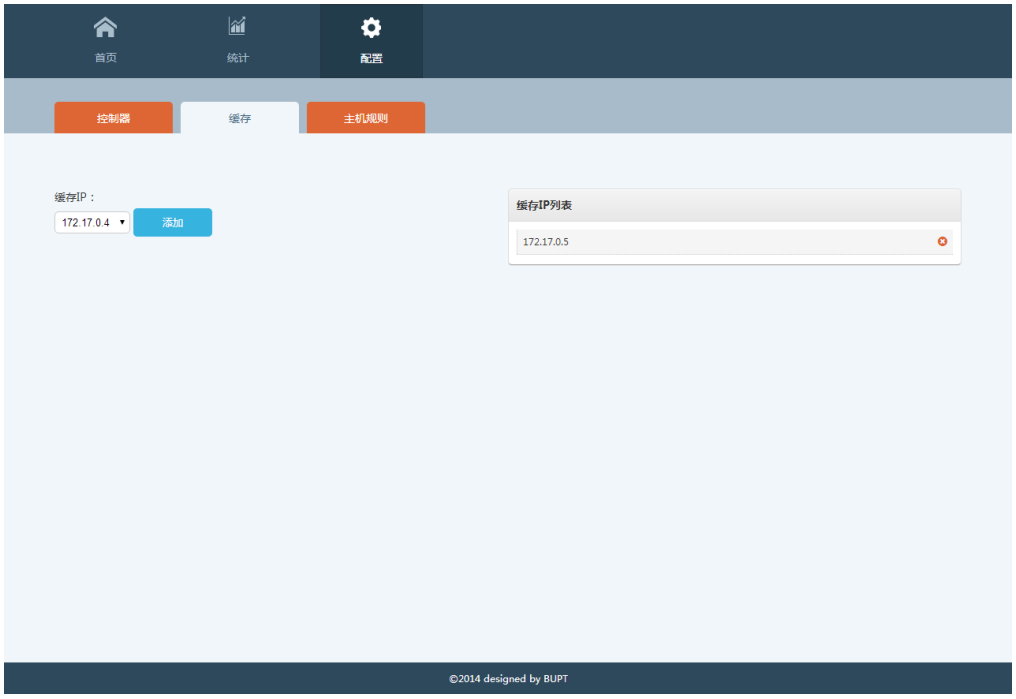

图 16 添加缓存

2）添加预先定义的策略规则：

源 IP：172.17.0.2/32，目的 IP：0.0.0.0/0（统配所有目的地址），目的端口 80，目的缓存为 172.17.0.5。这条规则制定终端 172.17.0.2 使用缓存 172.17.0.5 来获取所有的非实时资源。

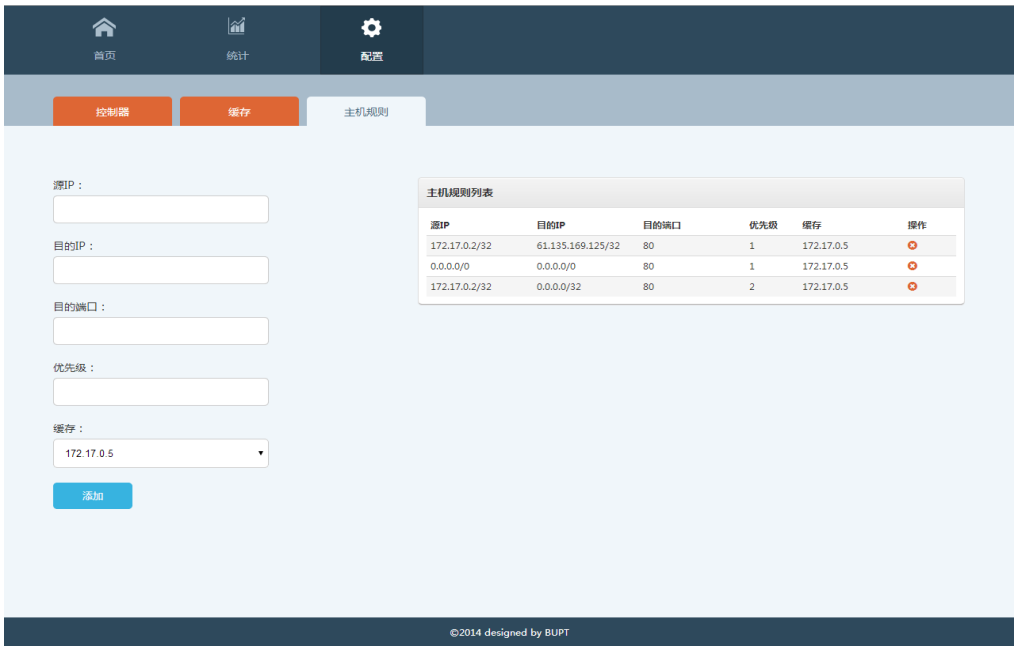

图 17 添加匹配规则

3）使用终端访问非实时资源，通过前端 Web 页面可以查看转发路径，验证流量被重定向至缓存节点。

5.2.3 实验结果

1. 实时业务流量优化结果

通过 Web 页面可以方便的查看直播流量的转发路径，如图 18 所示当单击页面下方的“组播组 1”时，显示该组播组的转发路径，此时组播组内只有一个信源和一个终端。

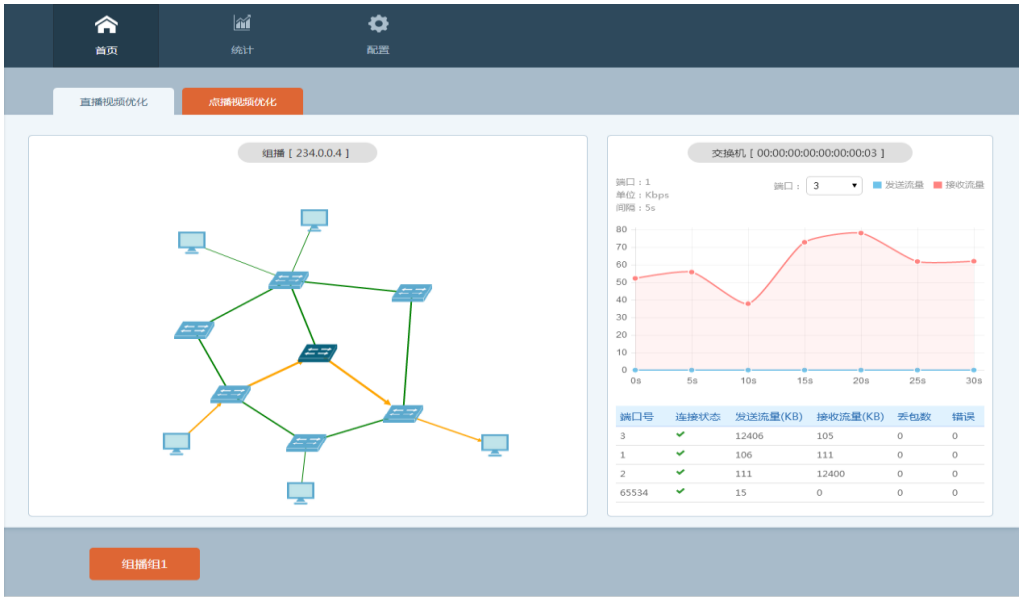

图 18 组播组转发路径

当我们选择路径上的某台交换机时，能够在右侧状态栏中查看交换机各端口的流量变化信息。

在统计页面中，可以查看到组播组的完整转发信息，如下图所示。

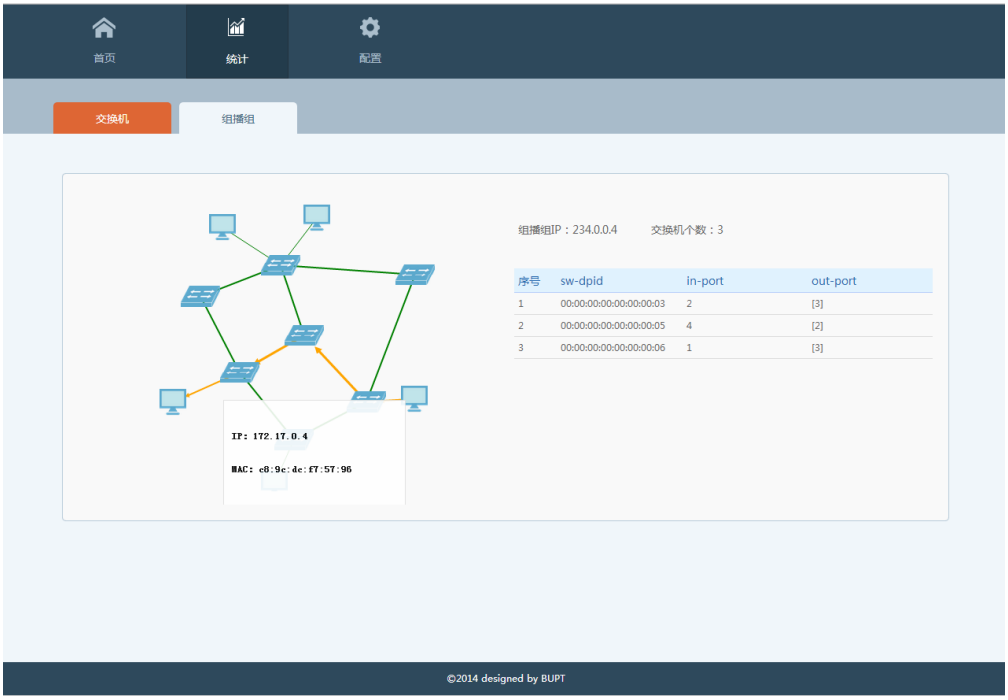

图 19 组播组转发信息

表 5 转发信息

序号	sw-dpid	in-port	out-port
1	00:00:00:00:00:00:00:05	4	[2]
2	00:00:00:00:00:00:00:03	2	[3]
3	00:00:00:00:00:00:00:06	1	[3]

接下来通过查看路径上的交换机可以看到所下发的流表项信息，如红色字体部分。

```
root@fnic:~# ovs-ofctl dump-flows s3
NXST_FLOW reply (xid=0x4):
cookie=0x30000000000000, duration=460.039s, table=0, n_packets=24556, n_bytes=33525064,
idle_age=0, priority=0,ip,in_port=2,nw_dst=234.0.0.4 actions=output:3
cookie=0xa0000000000000, duration=542.538s, table=0, n_packets=0, n_bytes=0, idle_age=542,
priority=1,tcp,tp_src=80 actions=CONTROLLER:32767
cookie=0xa0000000000000, duration=542.538s, table=0, n_packets=0, n_bytes=0, idle_age=542,
priority=1,tcp,tp_dst=80 actions=CONTROLLER:32767
```

可以看到该流表匹配发现组播 IP：234.0.0.4 的数据包，和转发信息表中的信息完全一致。

2．非实时业务流量优化结果

通过查看 Web 页面，在下方输入流量的特征，就可以在界面上显示流量的走向了，如图 20 所示可以看到访问测试网站（114.247.165.34）的流量都被发送到了缓存服务器。

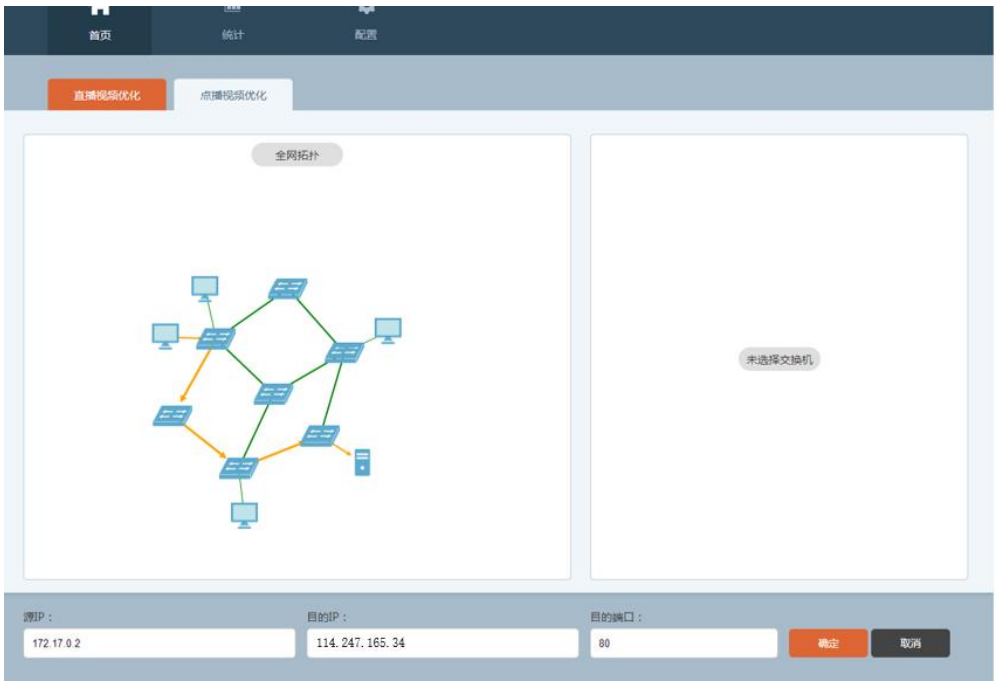

图 20 点播流量转发路径

接下来从交换机可以看到，如下红色字体的两条流表。

```
root@fnic:~# ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
cookie=0x20000000000000, duration=6.603s, table=0, n_packets=2, n_bytes=174,
idle_timeout=5, idle_age=1,
priority=0,in_port=LOCAL,vlan_tci=0x0000,dl_src=02:7f:7c:61:e7:4d,dl_dst=00:00:00:00:00:01
actions=output:4
cookie=0x20000000000000, duration=6.636s, table=0, n_packets=2, n_bytes=115,
idle_timeout=5, idle_age=1,
priority=0,in_port=4,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=02:7f:7c:61:e7:4d
actions=LOCAL
cookie=0xa0000000000000, duration=1200.857s, table=0, n_packets=3, n_bytes=222,
idle_age=56, hard_age=14, priority=1,tcp,tp_src=80 actions=CONTROLLER:32767
cookie=0x30000000000000, duration=5.553s, table=0, n_packets=4, n_bytes=639,
idle_timeout=15, hard_timeout=15, idle_age=4,
priority=2,tcp,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,nw
```

```
_src=172.17.0.5,nw_dst=172.17.0.2,tp_src=80,tp_dst=57521
actions=mod_nw_src:114.247.247.165.34,mod_dl_src:02:7f:7c:61:e7:4d,output:4
cookie=0x30000000000001, duration=6.569s, table=0, n_packets=6, n_bytes=569,
idle_timeout=15, hard_timeout=15, idle_age=4,
priority=2,tcp,in_port=4,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=02:7f:7c:61:e7:4d,nw
_src=172.17.0.2,nw_dst=114.247.165.34,tp_src=57521,tp_dst=80
actions=mod_nw_dst:172.17.0.5,mod_dl_dst:00:00:00:00:00:02,output:1
cookie=0xa0000000000000, duration=1200.857s, table=0, n_packets=4, n_bytes=296,
idle_age=6, hard_age=14, priority=1,tcp,tp_dst=80 actions=CONTROLLER:32767
```

这两条流表项对数据包进行了处理，第一条流表项匹配缓存节点发向终端的数据包，把源地址修改为测试网站的地址（114.247.165.34）。第二条流表项匹配终端发向百度的数据包，把目的地址修改为缓存节点的 IP（172.17.0.5）和 MAC（00:00:00:00:00:02）地址。

3．结论

通过上述实验验证可以看到，基于我们开发的 Web 管理界面，可以有效的管理非实时流量缓存策略，可以使管理员自定义缓存策略，同时通过 Web 管理界面可以方便地监控视频点播的转发状态、转发流量等信息。从而可以证明我们使用 SDN 的手段对多媒体流量进行了有效的优化，减少了流量冗余。