

OpenHW2014开源硬件与嵌入式计算大赛

项目论文

项目名称：基于SDN的网络流量管理与故障恢复系统

学校名称：西安交通大学

指导老师：胡成臣

团队成员：杨骥、龚志敏、许琛、乔思祎

电子邮箱：yangjifaust@gmail.com

联系电话：15691763667

网址：www.openhw.org

视频展示链接：

http://v.youku.com/v_show/id_XNzM3MTM4NjMy.html

目录

1. 背景.....	2
1.1 软件定义网络与OpenFlow交换机设计.....	2
1.2 数据中心网络的带宽管理与故障恢复.....	3
2. 总体方案设计.....	4
3. 硬件与逻辑设计.....	5
3.1 硬件板卡设计.....	6
3.2 FPGA逻辑设计	8

3.2.1 基于FPGA的基本网络系统	8
3.2.2 OpenFlow交换逻辑设计	13
4. 软件设计	15
4.1 SDN控制器选择	15
4.2 流量带宽管理功能设计	16
5. 系统测试	22
5.1 拓扑搭建	22
5.2 实验流量设计	24
5.3 实验过程	25
6. 总结	28
7. 参考文献	29

1. 背景

1.1 软件定义网络与OpenFlow交换机设计

软件定义网络（Software Defined Networking, SDN）是一种全新的网络体系结构，对网络的数据平面和控制平面进行了分离，进而对数据平面的处理细节实现抽象，基于控制平面的统一管理视图和编程接口实现基于全局视图的网络管控。SDN的提出改变了人们设计和控制网络的方法，它将网络中的控制平面与数据平面分离开来，使得控制平面能够专注于控制流量而转发平面专注于数据包的转发。在这样的网络中，控制平面的软件可以直接控制整个数据平面。OpenFlow是SDN的一种实现，目前已经成为事实上的SDN标准。它作为最初的SDN实现，将网络设备分为OpenFlow交换机以及控制器两部分。

OpenFlow交换机主要用于报文转发，内部被分为两个部分：流表与通道。流表是报文转发的核心部分，根据流表中的包匹配规则与包操作规则，对数据包进行匹配、操作和转发。目前OpenFlow协议中定义的流表支持多级流表结构。通道是OpenFlow交换机与控制器通信的接口，用

于接收控制器发来的控制信息，并根据当前情况向控制器发送特定消息，并能够支持特定的安全协议。除了上述两个部分之外，限速模块也是非常重要的一部分，它的出现为QoS在OpenFlow中实现打下了重要的基础。

目前对于OpenFlow交换机的实现，不论是工业界还是学术界都在探索当中。利用纯软件来实现OpenFlow数据平面是最主流的方法，它的主要优势在于软件的灵活性，它能够迅速实现最新的OpenFlow协议，同时可以达到最大的兼容性。但是它需要在性能和兼容性之间做平衡，很难达到限速线速转发，并且不适用于基础设施部署的情况。这一类的典型代表是OpenvSwitch。纯软件实现的交换机吞吐率非常低，无法满足实际的网络应用需求，也无法实现精确的流量带宽管理。在本项目中，我们基于Zynq平台实现OpenFlow交换机，通过FPGA和ARM软硬件协同工作的方式，对交换机进行硬件加速。

1.2 数据中心网络的带宽管理与故障恢复

数据中心是计算、存储、网络的资源汇聚中心，承载着大量的网络业务，对网络带宽的要求非常高。云计算模式的提出更是促进了数据中心的发展，目前大量的网络服务都部署在云数据中心的中心，例如搜索引擎（Google/Bing）、社交网络（Facebook/twitter），视频点播（YouTube/Netflix），网络存储（Dropbox/SkyDrive），大规模计算（数据挖掘/生物信息）。有些业务对带宽有着较为苛刻的需求，例如对于视频点播而言，如果带宽得不到保障，那么视频播放可能会降低播放质量或者卡顿，造成较差的用户体验，导致视频服务提供商用户数量减少，经济效益降低。虽然网络设备的带宽不断提高，但是网络用户和网络应用也在不断增加，导致网络流量井喷，并且流量增长的速度超过了网络带宽的增长速度，由于需求大于网络的链路的承受能力，所以在网络中就会出现拥塞，拥塞造成了数据包的丢弃，从而导致用户通信质量下降。然而，在TCP/IP“尽力而为”的服务模式下，网络不会给端到端的通信提供服务性能（包括吞吐率、带宽、时延、抖动）的保障，当发生网络拥塞时，网络通信质量过低，网络服务可能变得不可用。为了保证服务质量，应用提供商会与数据中心签订SLA（Service Level Agreement），要求数据中心为指定的业务提供带宽保障，所以带宽管理是数据中心网络的一项重要任务。

如今，一个数据中心有几十万甚至上百万的服务器已经很常见了。规模如此庞大的网络管理起来也不容易，因为即使一台机器的故障概率很小，几十万台机器出错概率就比较大了。数据中心中经常出现各种硬件、软件或人为配置错误会影响其网络性能。所以现在数据中心都有各种冗余设备，在机器故障时迅速替代出错机器尽力使网络快速恢复正常。最近几年，一些具有灵活性、有效性、易管理性和稳定性的数据中心网络结构被提出，如：BCube、DCell、FatTree和VL2。尽管它们都在路由中加入了容错方法，但是它们不能在错误严重影响网络性能的情况下采取进一步措施，这可能会造成巨大损失（如：Google Gmail 在2012年7月崩溃影响了480多万人，2013年8月，Google的搜索引擎、Gmail、YouTube和GoogleDrive等一系列服务突然中断了一到五分钟，期间全球网络流量锐减约四成）。所以，对网络进行实时监控，并且在发生故障的时候进行快速恢复对数据中心意义重大。

SDN网络是一种集中管控的架构，SDN控制器可以实时地查询交换机的业务流、端口的统计信息，然后计算出获得业务流速率和交换机端口的收发速率，以及丢包情况，从而获知当前的链路利用率以及判断是否发生拥塞。OpenFlow交换机的流表提供了细粒度的流量识别功能，业务流在流表中拥有一条流表项，我们可以在该流表项的Instruction/Action字段中添加meter指令或者Set queue动作，从而将业务流和对应的meter和queues关联起来，就可以对业务流执行某种配置的带宽控制。可以在OpenFlow交换机实现meter table和端口队列queues等特性，这些特性可以对业务流进行带宽预留、带宽限制，从而达到限制一般业务流的最高速率、给优先业务流提供带宽预留和保

障的目的。

通过SDN架构可以很方便地实现网络拓扑发现，根据当前的网络拓扑，利用最短路径算法生成全局路由。SDN控制器可以动态监控交换机端口状态，从而发现链路的连接和断开情况。当网络发生故障之后，动态更新当前拓扑，快速计算新路由，更新网络路由策略，从而绕开发生故障的线路，恢复网络的正常通信。

2. 总体方案设计

本文所设计的SDN流量管理与故障恢复系统见图2-1所示。整体架构包括硬件和软件两部分，硬件部分主要是OpenFlow交换机的硬件设计，包括平台设计、FPGA逻辑设计。软件部分主要是基于开源SDN控制器之上进行网络应用程序设计，这部分具体包括路由故障恢复系统和网络带宽管理系统。下面根据系统的软硬件结构分为三个层面对整个系统进行介绍。

1) 交换机层面。这是本项目的硬件设计部分。数据中心网络由OpenFlow交换机互联而成，交换机内部结构放大图显示了带宽管理所用到的OpenFlow交换机硬件特性，包括flow table, Meter table, Queues, 这三个特性相配合，达到带宽保障的目的，在本项目中，我们使用基于zynq的硬件平台实现交换机硬件，在FPGA中实现交换机的数据平面，在ARM中运行交换机软件。

2) SDN控制器层面。控制器负责维护所有交换机的连接，向应用程序提供网络编程的API，向应用程序通知网络事件。在本项目中，我们使用了开源的SDN控制器Ryu。

3) 应用层面。这是本项目的软件设计部分。我们主要关注两个网络应用，即路由故障恢复系统（Routing manager）和流量带宽管理系统（Bandwidth manager）。Routing manager负责收集网络拓扑，并利用最短路径算法计算路由表生成flow entry，然后将flow entry下发给网络中的交换机，同时把flow entry备份存入Flow entry database中。Bandwidth manager实现对网络数据流的带宽管理，在配置带宽的时候，需要查询Flow entry database获取交换机上的流表信息，然后对交换机下发带宽分配策略。status monitor模块对所有交换机的状态进行监控，监控内容包括：交换机端口状态、链路状态、端口带宽等。当链路发生故障的时候，status monitor会检测到链路状态变化，Routing manager对当前的网络拓扑进行更新，给因网络故障而不能通信的主机重新计算路由，然后把新的路由规则下发给交换机，使网络恢复正常通信。status monitor将收集到的端口带宽信息发送给Bandwidth manager，Bandwidth manager根据链路带宽利用率判断当前是否发生链路拥塞，如果发生链路拥塞，则计算拥塞链路的带宽分配策略，给关键业务流分配较大的带宽保障，并限制一般性业务流的速率，然后将带宽分配策略下发给交换机，从而实现流量带宽的动态调整，保障关键性业务的服务质量。

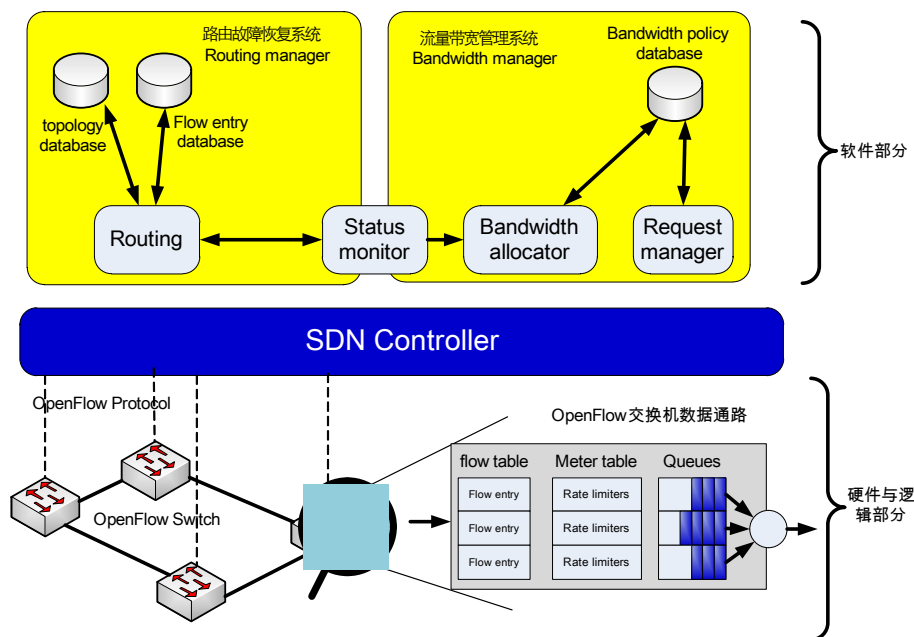


图2-1 SDN流带管理与故障恢复系统总体设计

3. 硬件与逻辑设计

本节介绍实现OpenFlow交换机所用的硬件板卡和逻辑设计，总体结构如图31所示。

硬件板卡部分包括Zynq板卡和网络扩展卡，网络扩展卡包括4个网络接口，利用bcm5464实现以太网物理通信，bcm5464通过gmii接口连接到FPGA上。扩展卡与zynq板卡之间通过FMC接口相连。

FPGA内部主要实现和网络处理有关的逻辑，vivado提供了Ethernet（内部包含PCS/PMA、MAC）、DMA等IP核，同时我们将OpenFlow的交换逻辑封装为一个IP核，然后把所有的IP核在IP integrator中集成在一起，所有的IP核都使用AXI总线连接，数据包传输采用AXI-stream接口，实现流水线式的高速吞吐；寄存器访问采用AXI-lite接口，实现基于地址的寄存器读写。AXI-stream通过interconnect模块连接到PS的HP接口上，AXI-lite通过interconnect模块连接到PS的GP接口上。

除了硬件之外，交换机还需要运行软件程序，这部分程序运行在ARM上，实现对硬件功能的扩展，首先，软件部分扩展了OpenFlow交换机流表，硬件无法处理的数据包将被DMA传到软件继续处理；其次，软件部分需要和OpenFlow控制器建立连接，执行控制器的命令。

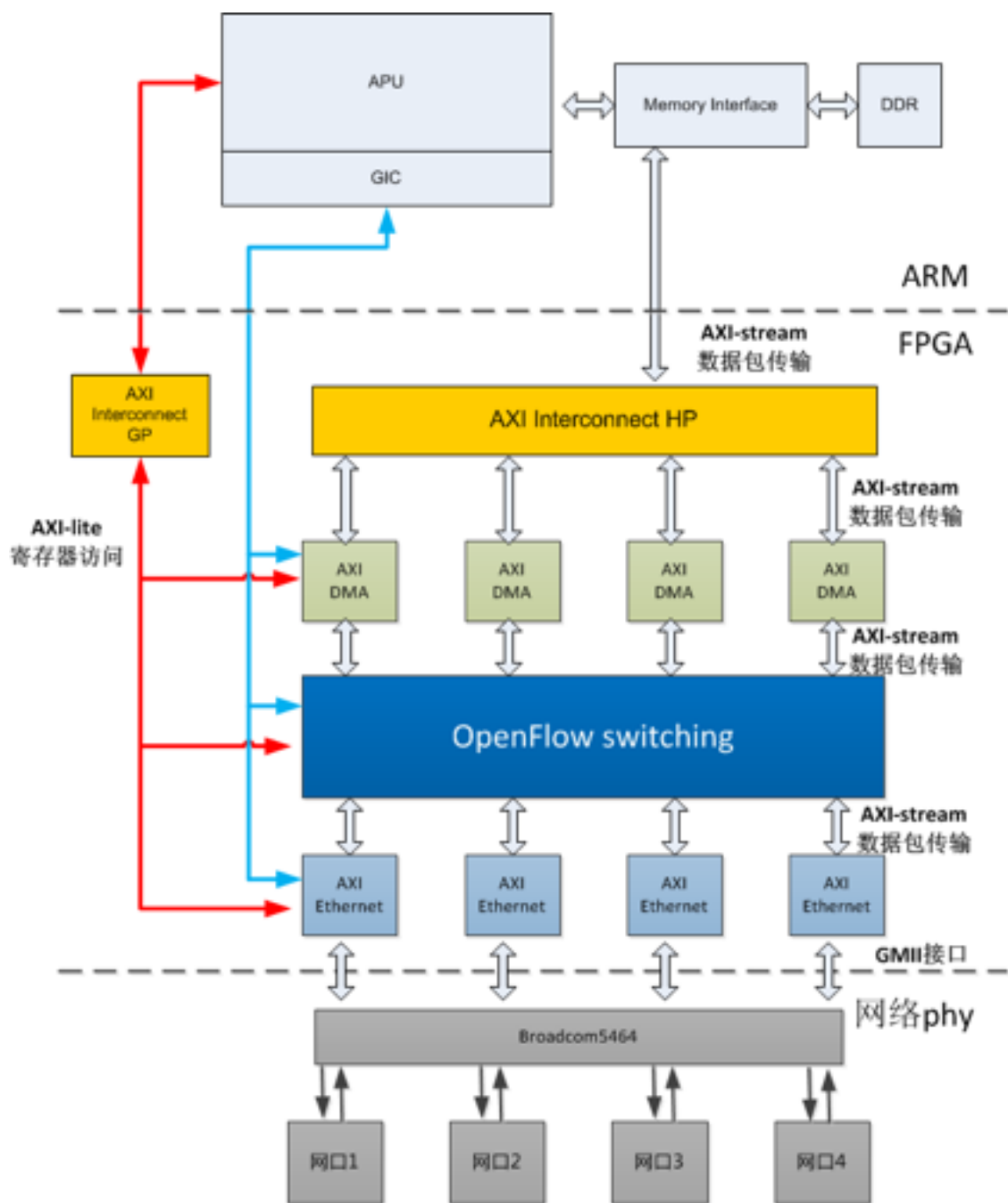


图31 硬件系统设计

3.1 硬件板卡设计

ONetSwitch是基于Xilinx Zynq SOC的网络创新平台。它提供三个层面的灵活性：包括“软件可编程，逻辑可重构，硬件可扩展”，让用户更加快速灵活地构建网络教学研究和创新平台，全面支持教学和科研工作。

ONetSwitch开放平台旨在提供小型化的开放网络创新平台，其主芯片采用Xilinx新一代的全可编程Zynq-7000系列芯片，以ARM Cortex A9通用处理器与强悍的可配置逻辑器件结合，提供高速灵活的网络接口，与强大的网络流转发处理能力，利用基于业界标准接口的模块化设计，增强型驱动架构设计，提供更加丰富的API，实现更加灵活的网络处理与协议支持，尽可能多地支持各

种网络系统应用，包括交换机、路由器、OpenFlow交换机等。

ONetSwitch提供ONetSwitch45高性能版本与ONetSwitch20通用版两个平台。本次实验中选用ONetSwitch20平台的构架如图32所示。整个设计核心为Zynq7020，速度等级-1，引脚数量为484。主芯片的PS（ARM Cortex A9）部分的连接 SPI Flash作为启动存储提供ARM以及FPGA的初始化配置，同时可以存放启动镜像以及应用程序；系统提供512MB DDR3 SDRAM，作为程序运行空间。SD卡接口支持16GB SDHC卡，可以存放启动镜像，Linux Kernel镜像，文件系统等；USB-UART提供基本的输入输出，USB主设备可以接驳闪存盘等一些USB设备，以及独立的千兆RJ45直接与ARM相连，可以用于程序调试，OpenFlow安全通道等。

4个千兆以太网接口使用Broadcom的BCM5464作为物理层（PHY）芯片，BCM5464是一款多端口10/100/1000自适应的PHY，支持全/半双工，内部集成了四路千兆位电信号收发器，能够分别独立工作在1000M模式下。4个千兆以太网口与PL部分连接。PL部分主要由Artix FPGA实现，用户可以通过合理划分PS与PL的工作部分，编写逻辑，使用FPGA作为系统网络数据处理，使得系统吞吐率成倍提升，达到线速吞吐率的目标。

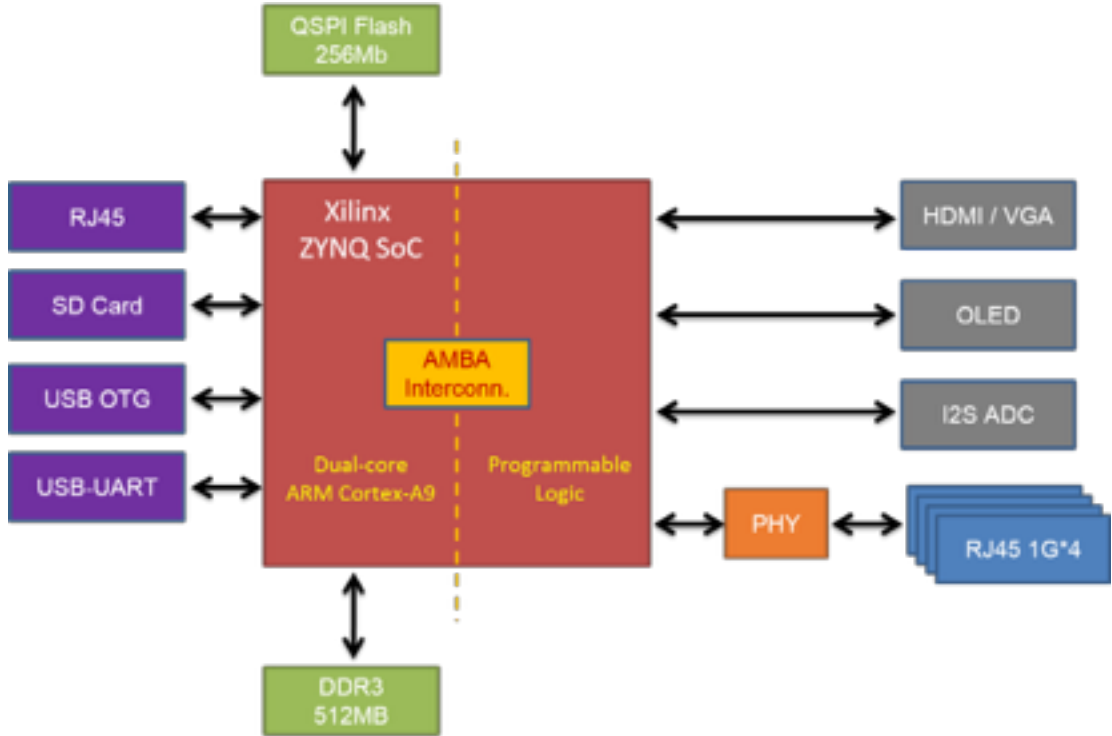


图32 ONetSwitch 20系统框图

ONetSwitch实物如下图所示，其参数见表3-1所示。

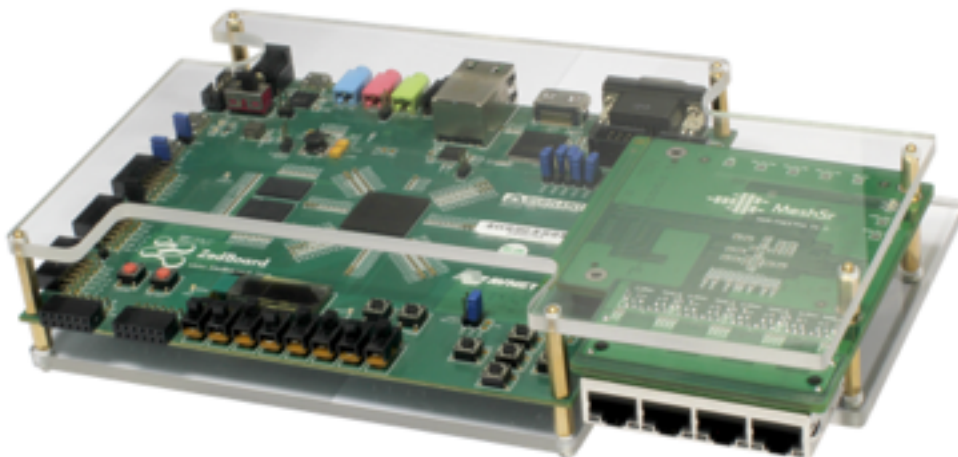


图33 ONetSwitch实物图

表31 ONetSwitch参数总结

资源	参数
处理器	Dual ARM Cortex-A9 @800MHz
DRAM	DDR3 512MBytes
QSPI Flash	32 MB
FPGA资源	Artix-7 FPGA，约85K逻辑单元
网络接口	1个千兆电口与ARM相连 4个千兆电口与FPGA相连

3.2 FPGA逻辑设计

3.2.1 基于FPGA的基本网络系统

1) 使用IDE创建一个vivado工程

选择Start > All Program > Xilinx Design Tools > Vivado 2013.3 > Vivado 2013.3打开Vivado。

1. 点击Create New Project来开始创建工程向导。看见Create A New Vivado Project 对话框。点击Next。
2. 在弹出的对话框中， 设置Project name为lab1， 工程目录设置为Workshop/Labs目录下， 点击Next。
3. 选择RTL Project并点击Next。

4. 在Add Source对话框中点击Next,在工程建立好后手动工程源文件。
5. 一直Next,直到Default Part对话框出现。
6. 在Default Part对话框中, 使用Boards选项, 在Filter中选择Name为zed, Version为d, 选择zedboard作为目标开发板, 点击Next.
7. 点击Finish完成Vivado工程的创建。

2) 使用IP Integrator创建IP子系统

2-1 在Flow Navigator窗口中点击 IP integrator Create Block Design 按钮 

打开 Create Block Design对话框, 输入Design name为lab1_1, 点击OK完成创建,

2. 此时一个空白的Block Design窗口已经展现在我们面前
3. 接下来在这个空白的画布中设计工程, 首先添加一个ZYNQ IP 核。右键空白区域, 在弹出来的菜单中选择Add IP选项, 在搜索框中输入ZYNQ并双击 ZYNQ7 Processing System选项添加ZYNQ IP。
4. IP Integrator将ZYNQ IP添加到设计中, 并在画布中以图形化的方式显示出来。
5. 双击画布中的Processing_system7_0, 对ZYNQ进行重新定制,在弹出的Re-customize IP对话框中, 单击左上角的Presets按钮, 选择ZedBoard选项, Vivado将按照zedboard的默认配置对ZYNQ进行定制, 虽然这样, 我们仍然还是要进行一些手动的更改来对我们自己系统进行适配。
6. 在Page Navigator中点击Clock Configuraton配置时钟, 展开PL Fabric Clocks, 选择前3个时钟, 设置CLK0为125MHz, CLK1为75MHz,CLK2为200MHz
7. 点击Interrupts设置中断。
8. 启用HP Port.点击PS-PL Configuration.展开 HP Slave AXI Interface.选中S AXI HP0 interface。
9. 点击OK结束ZYNQ配置。
10. 按如上方法, 添加下表中的IP, 并根据Configuration进行配置。最终在Diagram画布中的视图应该如图所示:

Name of the IP(instance name)	Configuration
AXI Ethernet(axi_etherne_0)	Physical Interface: RGMII Shared Logic: Include Share Logic in Core

AXI Ethernet(axi_etherne_1)	Physical Interface: RGMII Shared Logic: No Share Logic in Core
AXI Ethernet(axi_etherne_2)	Physical Interface: RGMII Shared Logic: No Share Logic in Core
AXI Ethernet(axi_etherne_3)	Physical Interface: RGMII Shared Logic: No Share Logic in Core
Concat(xlconcat_0)	Number of Ports : 12
Constant(xlconstant)	default

11. 点击Run Block Automation,选择/processing_system7_0选项，在弹出的对话框中点击点击OK完成ZYNQ模块的自动化连接。
12. 如上，继续使用Run Block Automation，自动连接4个axi Ethernet 到相应的DMA上，这里是VIVADO给我们提供的自动化默认连接，当然我们也可以自己添加DMA IP，然后再进行手动连接。
13. Vivado不仅给我们提供了Block 自动连接，它还能使用户可以轻松的对匹配的接口进行自动化连接。点击run Connection Automation, 接口菜单列表中选择/axi_ethernet_0/s_axi, 在弹出的Run Connection Automation对话框中确保Master选项为/Processing_system7_0/M_AXI_GP0/ 点击OK完成连接。
14. 在进行接口自动化连接之后，由于软件默认连接时钟为fclk0, 但是我们这里使用的S_AXI时钟为fclk1, 所以我们需要将ZYNQ7 Processing System的FCLK_CLK0断开，然后将FCLK_CLK1连接到 processing_system7_0_axi_periph的ACLK上面。



15. 选中FCLK_CLK0,右键点击Disconnect pin选项，断开FCLK_CLK0的连接;选中FCLK_CLK1，左键拖移到ACLK处，当时钟信号线变粗的时候放开，连接FCLK_CLK1;选中proc_sys_reset的slowest_sync_clk，先将其断开连接，然后相同的方法连接到FCLK_CLK1上。
16. 使用run Connection Automation工具依次将接口连接到M_AXI_GP0上。
17. 接下来我们利用Run Connection Automation工具来连接DMA到ZYNQ HP接口上。点击Run Connection Automation，选择/Processing_system7_0/S_AXI_HP0选项，在弹出的对话框中点击OK。
18. 仍然使用Run Connection Automation，依次选择选项进行连接，在弹出的对话框中不做任何改变，点击OK即可。
19. 在diagram窗口左边的工具栏中点击自动调整布局工具按钮
20. 下面我们进行手动连接，选中processing_system7_0（以下简称PS）的FCLK_CLK0接口，右键选择Make Connection，在弹出来的对话框中选中4个ethernet的axis_clk信以及axi_ethernet_0的gtx_clk时钟信号进行连接。
21. 连接中断。首先连接xlconcat_0的dout[11:0]与PS端的IRQ_F2P[0:0]进行连接，然后按下表将各模块的中断信号进行连接。

Source IP	xlconcat_0
axi_ethernet_0/interrupt	In0
axi_ethernet_1/interrupt	In1
axi_ethernet_2/interrupt	In2
axi_ethernet_3/interrupt	In3
axi_ethernet_0_dma:mm2s_introut :s2mm_introut	In4 In5
axi_ethernet_1_dma:mm2s_introut :s2mm_introut	In6 In7
axi_ethernet_2_dma:mm2s_introut :s2mm_introut	In8 In9
axi_ethernet_3_dma:mm2s_introut :s2mm_introut	In10 In11


22. 共享时钟连接。将axi_ethernet_0的gtx_clk90_out与gtx_clk_out连接到其它的axi_ethernet，gtx_clk与gtx_clk90。


23. 拉高。Xlconstant_0的作用就是固定输1，设计中我们需要将proc_sys_reset的dcm_locked拉高，将其按如下图连接即可。
24. 连接ref_clock,与系统复位信号。Axi_ethernet_0的ref_clk为200MHz，将其与PS端的FCLK_CLK2连接；PS端的FCLK_RESET0_N应该接到proc_sys_reset的aux_reset_in作为系统输入reset。
25. Make External。使用Run Connection Automation 工具，依次选中下图中红圈中的选项，将其扩展到外部。然后依次右键每个ethernet的phy_rst_n接口，Make External。
26. 修改外部端口名字，以使实例化的时候方便一点。在TCL Console中依次输入下面的命令对相应的名字进行修改。

```
set_property name mdio_0 [ get_bd_intf_ports mdio_io]
set_property name mdio_1 [ get_bd_intf_ports mdio_io_0]
set_property name mdio_2 [ get_bd_intf_ports mdio_io_1]
set_property name mdio_3 [ get_bd_intf_ports mdio_io_2]
set_property name rgmii_0 [ get_bd_intf_ports rgmii_rtl]
set_property name rgmii_1 [ get_bd_intf_ports rgmii_rtl_0]
set_property name rgmii_2 [ get_bd_intf_ports rgmii_rtl_1]
set_property name rgmii_3 [ get_bd_intf_ports rgmii_rtl_2]
set_property name phy_rst_n [ get_bd_ports phy_rst_n_0]
```

27. 点击  按钮，对布局进行系统调整。
28. 点击Diagram左边工具栏  按钮进行设计有效性检测。当出现如下图时说明设计连接之间没有错误， 否则，请根据错误信息进行相应的修改，点击OK完成系统设计。

3) 创建顶层文件以及约束文件

1. 在Project Manager中选择Source > Design Source，右键lab1_1，选择Generate Output Products选项，在弹出的对话框中点击Generate。
2. 在Design Source下右键lab1_1， 选择Create HDL wrapper。创建IP子系统顶层文件。在弹出的对话框中选择默认选项， 让系统自动管理子系统的顶层文件。
3. 创建系统顶层文件。展开Project Manager, 单击Add Source按钮 ，在弹出的对话框中选择Add or Create Design Source选项，点击Next。
4. 选择Add Files按钮， 定位到~/Workshop/Source/lab1/lab1_top.v文件，点击OK返回主界面，点击finish完成添加。

添加约束文件。点击  按钮打开添加Source对话框。选择Add or Create Constraints选项。点击Next, 在Add source的对话框中点击Add Files 按钮，定位到/Workshop/Source/lab1/lab1_1.xdc文件，点击Ok，点击finish完成添加。

基础硬件搭建完成之后，vivado原理图如图34所示。

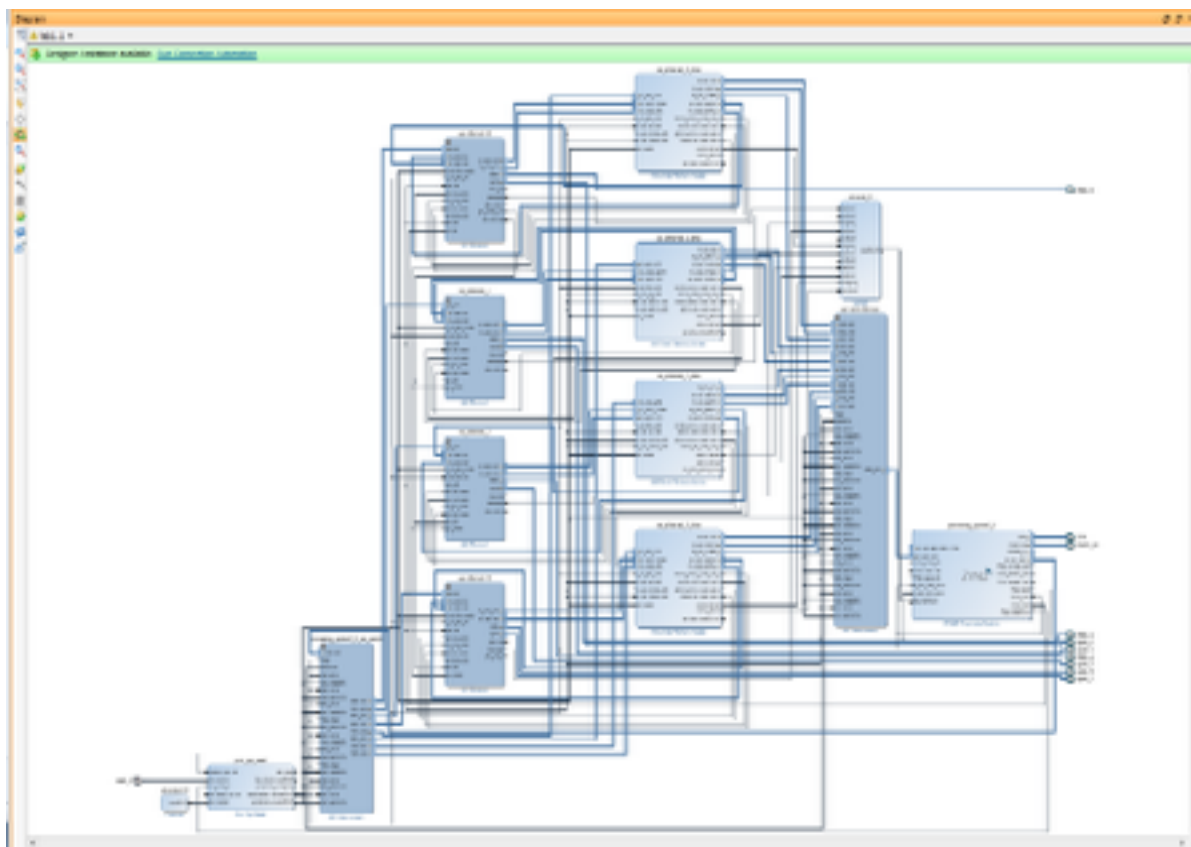


图34基于zynq的基本网络系统

3.2.2 OpenFlow交换逻辑设计

上一节介绍了基础网络功能的搭建，本节介绍OpenFlow交换逻辑，并将交换逻辑添加到基础网络工程中。在基础网络工程里，Ethernet直接和DMA相连。加入OpenFlow交换逻辑之后，OpenFlow交换逻辑实现数据包交换的功能，所以我们需要断开原有的Ethernet和DMA之间的连接，将所有的Ethernet和DMA都连到OpenFlow交换逻辑上，接收和发送的数据包都会经过OpenFlow交换逻辑，OpenFlow交换逻辑将会对数据包进行处理和转发。来自以太网口的数据包首先被OpenFlow交换逻辑处理，对于交换逻辑无法处理的数据包，可以通过DMA上传到软件中，由软件交换功能程序进行处理，用户可以灵活地配置硬件交换逻辑和软件交换程序以实现特定的功能。

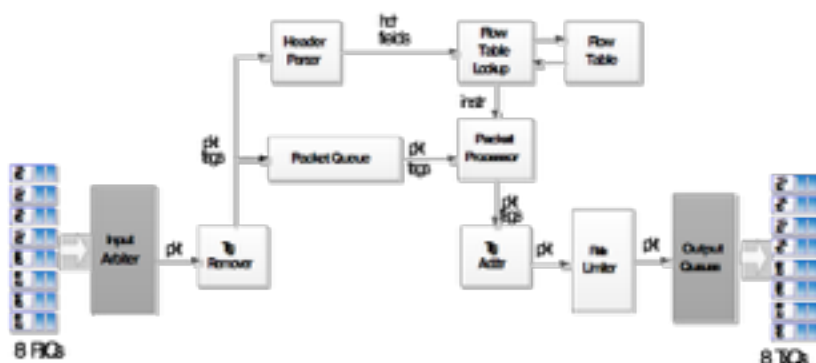


图35 OpenFlow交换逻辑的数据包处理流水线

功能模块

- Input Arbiter: 从多路输入中选择一路进行处理。
- Output Queues: 根据数据包的处理结果将数据包转到相应的输出队列。
- Tag Remover: 将数据包中的VLAN和MPLS标签和原始数据包分离。
- Tag Adder: 将处理完成后的标签和数据包合并, 组成完整的数据包。
- Header Parser: 解析数据包的各个域, 将各域的值提取出来, 用来进行流表查找。
- Flow Table: 存储流表内容, 利用寄存器访问通道进行配置, 采用了三态匹配。
- Flow Table Lookup: 实现对flow table的管理。
- Packet Queue: 缓存数据包。
- Packet Processor: 根据数据包的查找结果对数据包进行处理。
- Rate Limiter: 采用漏桶算法对数据流进行限速。

流水线处理流程

来自8个输入通道(4路Ethernet和4路DMA)的数据包由Input Arbiter进行仲裁, 合成一路输入。然后Tag Remover将数据包所带的标签去除, 以简化处理逻辑。然后数据包被暂存到Packet Queue中, 等待查找完成, 同时Header Parser对数据包进行解析, 得到包头域, Flow table lookup利用包头域查找流表, 得到查找结果, 即instructions, Packet Processor根据instructions对数据包进行处理, 例如修改包头域, 添加或者修改tags, 设置限速参数, 设置转发端口等。然后Tag Adder将前面分离的tag合并到数据包中, 组合成完整的数据包。Rate Limiter实现对数据流的限速, 不需要限速的数据包将以全速率通过。然后由Output Queues将数据包进行分发。

将OpenFlow交换逻辑添加到基础网络工程中之后, 得到完整的FPGA模块设计如图36所示。

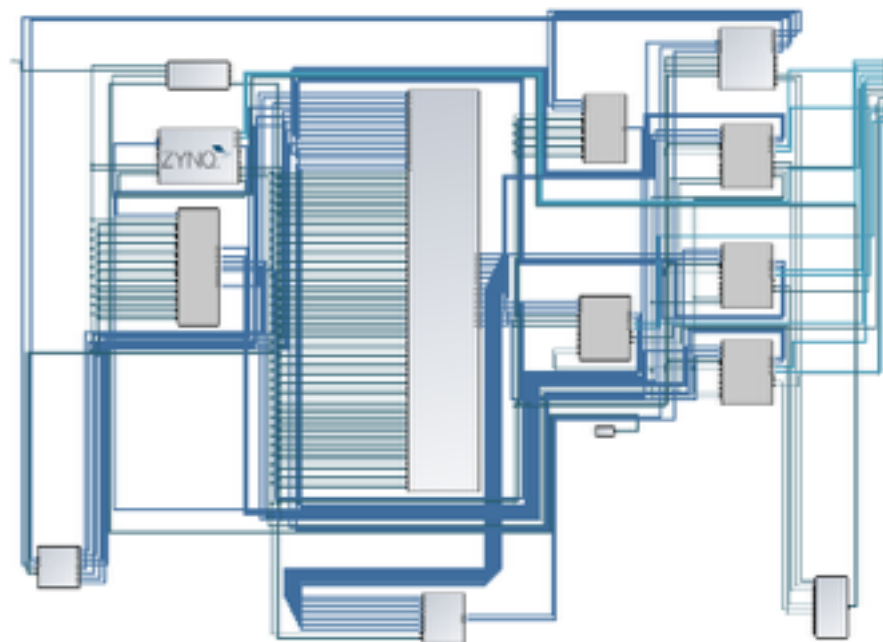


图36 加入OpenFlow交换逻辑之后的硬件工程

4. 软件设计

4.1 SDN控制器选择

随着学术界研究的深入以及业界与开源力量的推动，SDN控制器的种类也变得越来越多样，例如最初的NOX，POX，Beacon，设备制造商NEC、HP、IBM各自的闭源专用控制器，以及非开源商用控制器Onix。影响较大的开源控制器包含Floodlight，POX，Open Daylight，NOX，Ryu，OVS-Controller，Trema等等。Floodlight使用Java实现，性能相对其他使用Python的控制器有一定优势，但是目前支持的OpenFlow协议版本仅为1.1。Open Daylight是大厂商联盟的开源控制器，它包含丰富的南向接口。NOX是由C++实现的性能较高的控制器，其官方版本支持OpenFlow协议1.1，而CPqD公司对其进行了升级，能够部分支持1.3的特性，但是在容错方面不能尽善尽美。我们在使用过程中发现，其稳定性不足，在特定情况下可能发生主程序崩溃。

综合上面容易获得的开源控制器的优缺点，我们选择RYU 3.8作为控制器。Ryu是一个以Python实现的OpenFlow控制器，它的名字在日语中是“流”的意思。Ryu对外提供RESTful控制API，方便与OpenStack对接的Quantum REST API，以及用户可以自定义的REST API和用户RPC调用。Ryu内部包括若干个不同的层次，从高层次向低层次分别为：内建应用程序层，其中有L2交换，租户隔离等；内部库，包含OF REST库，拓扑发现，防火墙等；OF协议解析与序列化部分，包含OF1.0，1.2，1.3，1.4以及OF-Config 1.1协议解析；非OF协议的解析与序列化，包含netconf，vrrp，netflow，xFlow，snmp，packet lab，ovsdb等部分。使用者可以通过编写内建程序或者使用RESTful API两种方法获取当前Ryu控制的SDN网络状态，并进行SDN网络的控制。图41为Ryu控制器接口与对外接口图。

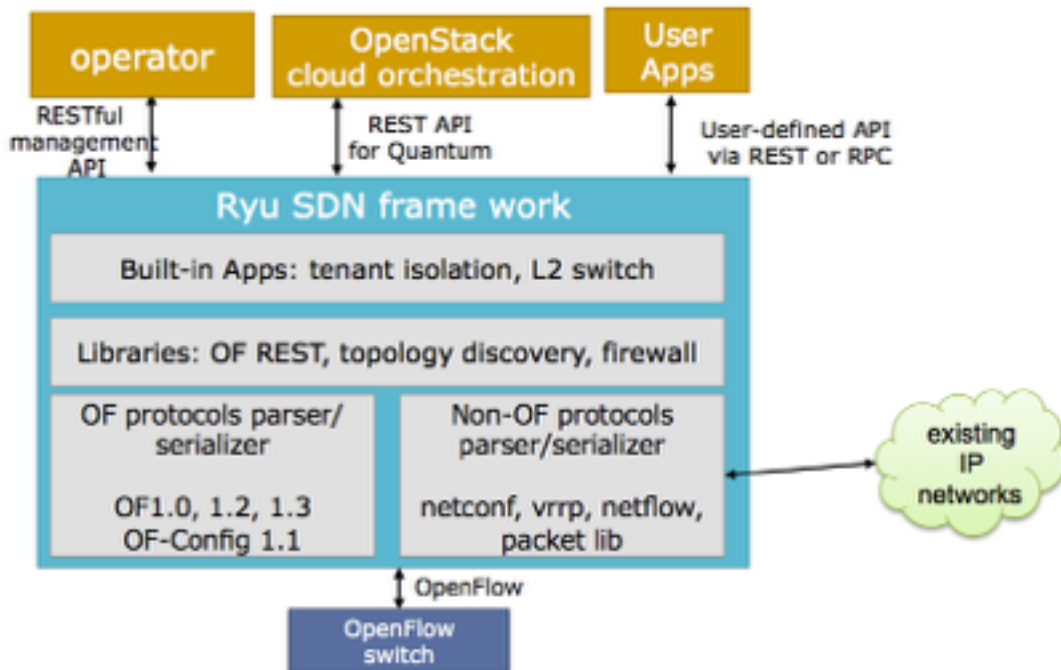


图41 Ryu 控制器结构与对外接口

4.2流量带宽管理功能设计

Bandwidth manager系统流程图见图42，它包含四个子模块。

1) Request manager是处理带宽保障请求的模块，它向外提供网络服务端口，和经过注册并认证的业务流服务器（服务商）建立连接，受理业务流服务器的带宽需求。同时，它也提供基于Web的图形化人机接口，允许网络管理员用来进行手工的带宽策略配置。Request manager将带宽保障策略写入Bandwidth policy database中。

2) Bandwidth policy database是带宽管理策略库，里面存放了需要进行带宽保障的业务，以及该业务对带宽预留的需求量。

3) Status monitor是网络状态监控模块，负责监控当前网络中各交换机的端口链路状态，判断是否发生拥塞，如果发生拥塞，则通知Bandwidth allocator，对业务流进行带宽调整。

4) Bandwidth allocator是带宽分配模块，它对交换机的带宽资源进行管理，根据Flow entry database中存储的业务流以及Bandwidth policy database中存储的业务流带宽分配策略，对交换机的meter表或者端口queue进行配置，并将业务流的flow entry和meter或者queue进行关联，从而达到给业务流限速或者给业务流预留带宽的目的。

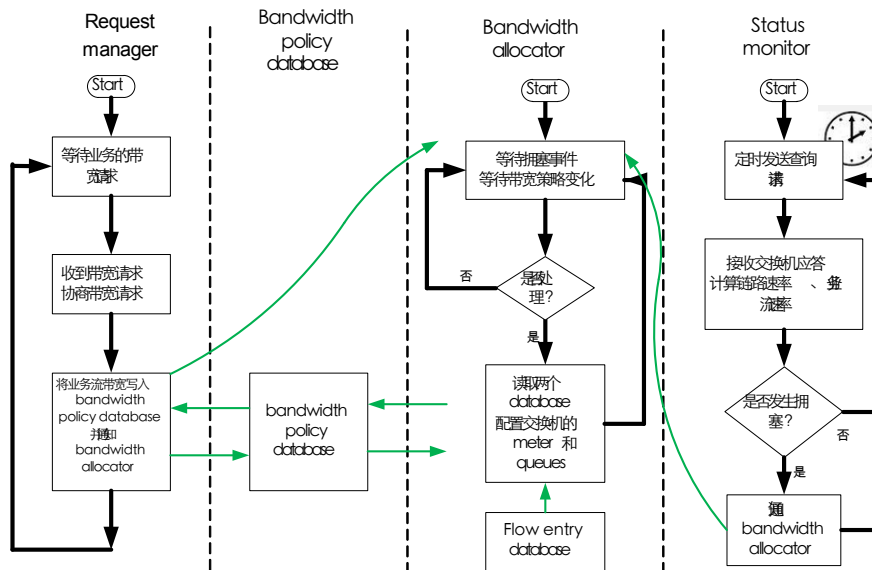


图42 Bandwidth manager系统流程图

下面对带宽管理系统中各部分的功能进行更为详细的分析，并给出具体的实现思路。

1) Request manager

Bandwidth manager的一个重要功能就是网络业务服务器可以主动发起带宽申请，例如，当某个紧急的计算任务被提交，并且该计算任务的通信量较多时，可以临时申请带宽资源预留；或者某个业务的用户数量在某个时间段可能会激增，需要临时增加带宽占用，从而保证通信高峰时的服务质量。

Request manager是运行在控制器上的一个服务进程，在某个端口进行监听，由业务服务器上的带宽管理程序或者业务管理员操作客户端来发起主动连接。业务服务器和Request manager之间的连接需要有一定的认证机制，保证只有那些在云计算提供商注册过的业务才能通过认证进行带宽申请。注册和认证都有比较成熟的技术。

通过认证建立连接之后，二者需要一定的协商机制来进行带宽的申请和受理。业务端（客户端）可以向Request manager发送带宽保障需求，Request manager根据已分配的带宽数据，计算当前状态能否满足满足客户端的带宽需求，如果不能满足，则返回False，同时返回建议的带宽量，客户端可以根据Request manager返回的建议带宽重新发起新的带宽申请，或者放弃。

对于那些协商成功的带宽请求，Request manager将业务特征和带宽预留写入Bandwidth policy database中。根据客户端请求的类型，决定是否立即通知Bandwidth allocator配置交换机。如果客户端请求的时候要求立刻进行带宽预留，Request manager将立即通知Bandwidth allocator，并将和该业务流有关的交换机进行配置。如果客户端请求的时候不要求立即进行带宽分配，可以只修改Bandwidth policy database，网络发生拥塞时再检查Bandwidth policy database，根据具体业务的带宽需求进行带宽分配。

2) Bandwidth policy database

带宽策略库中记录的信息至少要支持以下几种基本功能：（1）根据新业务的带宽请求，查询出当前的Qos配置记录，计算带宽请求能否满足。（2）针对一个业务，查询出该业务的所有流表项、流表项所在的交换机、每个交换机给该业务流分配的Qos策略，以便进行具体的带宽配置。

(3) 根据发生拥塞的交换机和端口号，查询出受影响的所有业务流，以及每个业务流分配的带宽保障。

带宽策略库存储了所有需要带宽保障的业务以及给它分配的带宽保障策略，带宽保障策略是交换机上的Qos配置，meter table、Queue的配置都需要在这里记录，除此之外还需要记录该业务对应的每一条流表项flow-entry-id、流表项所在的交换机dpid， Qos策略id之间的关联关系，以便到其他数据库中查询所需要的信息。

3) Status monitor

SDN控制器需要维持很多状态信息，组成一个状态库，Status monitor主要记录和网络链路带宽相关的记录。

交换机上设置了很多计数器，Status monitor最基本的功能是监控交换机每个端口是否发生了拥塞，以及监控每一条流的速率，实现的时候可以利用OpenFlow的OFPT_MULTIPART_REQUEST类型的消息不断查询交换机的统计信息，例如OFPMP_FLOW/OFPMP_AGGREGATE，OFPMP_PORT_STATS。

下面介绍两种计算速率的方法。

(1) 利用流表统计信息计算业务流速率和端口速率

OFPMP_FLOW消息用于查询满足特定条件的流表项的统计信息。请求消息为：

```
struct ofp_flow_stats_request {
    uint8_t table_id; /* ID of table to read (from ofp_table_stats), OFPTT_ALL for all tables. */
    uint8_t pad[3]; /* Align to 32 bits. */
    uint32_t out_port; /* Require matching entries to include this as an output port. A value of OFPP_ANY indicates no RESTriction. */
    uint32_t out_group; /* Require matching entries to include this as an output group. A value of OFPG_ANY indicates no RESTriction. */
    uint8_t pad2[4]; /* Align to 64 bits. */
    uint64_t cookie; /* Require matching entries to contain this cookie value */
    uint64_t cookie_mask; /* Mask used to RESTrict the cookie bits that must match. A value of 0 indicates no RESTriction. */
    struct ofp_match match; /* Fields to match. Variable size. */
};
```

在请求消息中可以指定flow entry的match域、以及该流从哪个网口出去。例如，如果我们需要监控端口1的拥塞程度，就可以将out_port设为1，交换机将返回所有去往端口1的流的统计信息，统计信息的应答消息格式为：

```
struct ofp_flow_stats {
    uint16_t length; /* Length of this entry. */
    uint8_t table_id; /* ID of table flow came from. */
    uint8_t pad;
    uint32_t duration_sec; /* Time flow has been alive in seconds. */
    uint32_t duration_nsec; /* Time flow has been alive in nanoseconds beyond duration_sec. */
    uint16_t priority; /* Priority of the entry. */
    uint16_t idle_timeout; /* Number of seconds idle before expiration. */
    uint16_t hard_timeout; /* Number of seconds before expiration. */
    uint16_t flags; /* One of OFPFF_* */
    uint8_t pad2[4]; /* Align to 64-bits. */
    uint64_t cookie; /* Opaque controller-issued identifier. */
    uint64_t packet_count; /* Number of packets in flow. */
    uint64_t byte_count; /* Number of bytes in flow. */
    struct ofp_match match; /* Description of fields. Variable size. */
    //struct ofp_instruction instructions[0]; /* Instruction set. */
};
```

根据应答消息中的duration_sec、duration_nsec、byte_count，我们可以计算出这段时间该业务流的平均速率。

设第一次（t1）读取流表统计信息的时候，读到一条流的duration_sec和duration_nsec分别为duration_sec_t1和duration_nsec_t1，读到的byte_count为byte_count_t1。设第二次（t2）读取端口统计信息的时候，读到的duration_sec和duration_nsec分别为duration_sec_t2和duration_nsec_t2，读到的byte_count为byte_count_t2。那么从t1到t2这段时间的平均速率flow_rate（单位bps）为：

$$flow_rate = \frac{byte_count_t2 - byte_count_t1}{(duration_sec_t2 + duration_nsec_t2 \times 10^{-9}) - (duration_sec_t1 + duration_nsec_t1 \times 10^{-9})} \times 8$$

设发往一个端口的各个flow的速率为flow_rate，那么把这些flow_rate相加可以得到该端口的发送流量速率：

$$port_tx_rate = \sum_{flow} flow_rate$$

从这些数据中，我们可以得到两点信息：

1. 获得特定业务流的实际带宽，记录业务流的流量历史特征，不仅可以查看业务流的带宽是否得到保障，而且也可为科研和流量优化提供支持。
2. 获得端口是否拥塞，将出端口相同的所有流加起来，可以得到该端口队列enqueue的总速率，如果enqueue的总速率大于dequeue的速率（即链路的最大带宽），那么该端口一定会生拥塞。该功能也可以利用OFPMP_AGGREGATE消息查询。

（2）利用端口统计信息计算端口速率

OFPMP_PORT_STATS可以查询交换机端口的统计信息，格式为：

```
struct ofp_port_stats_request {
    uint32_t port_no; /*single port or all ports*/
    uint8_t pad[4];
};
```

应答信息包括：

```
struct ofp_port_stats {
    uint32_t port_no;
    uint8_t pad[4]; /* Align to 64-bits. */
    uint64_t rx_packets; /* Number of received packets. */
    uint64_t tx_packets; /* Number of transmitted packets. */
    uint64_t rx_bytes; /* Number of received bytes. */
    uint64_t tx_bytes; /* Number of transmitted bytes. */
    uint64_t rx_dropped; /* Number of packets dropped by RX. */
    uint64_t tx_dropped; /* Number of packets dropped by TX. */
    uint64_t rx_errors; /* Number of receive errors. This is a super-set of more specific receive errors and should be greater than or equal to the sum of all rx_*_err values. */
    uint64_t tx_errors; /* Number of transmit errors. This is a super-set of more specific transmit errors and should be greater than or equal to the sum of all tx_*_err values (none currently defined.) */
    uint64_t rx_frame_err; /* Number of frame alignment errors. */
    uint64_t rx_over_err; /* Number of packets with RX overrun. */
    uint64_t rx_crc_err; /* Number of CRC errors. */
    uint64_t collisions; /* Number of collisions. */
    uint32_t duration_sec; /* Time port has been alive in seconds. */
    uint32_t duration_nsec; /* Time port has been alive in nanoseconds beyond duration_sec. */
};
```

根据应答消息中的tx_bytes可以算出这段时间该端口平均的bps速率。如果平均的bps接近链路最大速率，那么可能就发生拥塞了。链路的最大速率可以通过OFPMP_PORT_DESCRIPTION消息

查询curr_speed和lmax_speed。

设第一次（t1）读取端口统计信息的时候，读到端口的duration_sec和duration_nsec分别为duration_sec_t1和duration_nsec_t1，读到的tx_byte为tx_byte_t1。设第二次（t2）读取端口统计信息的时候，读到的duration_sec和duration_nsec分别为duration_sec_t2和duration_nsec_t2，读到的tx_byte为tx_byte_t2。那么从t1到t2这段时间的平均速率flow_rate（单位bps）为：

$$port_tx_rate = \frac{tx_bytes_t2 - tx_bytes_t1}{(duration_sec_t2 + duration_nsec_t2 \times 10^{-9}) - (duration_sec_t1 + duration_nsec_t1 \times 10^{-9})} \times 8$$

利用目前OpenFlow协议所提供的支持虽然能够实现一定程度上的网络状态监控，但是在实践中仍然存在缺点，例如：

（1）统计信息从硬件通过总线读到交换机CPU中，然后软件将统计信息打包放入OpenFlow reply消息中发出去，经过控制通道时中间的跳数也有一定的传输时延，最后控制器I/O接受以及解包，所以控制器发现拥塞具有一定的滞后性。

（2）监控功能使用的轮询的方式有较大的缺点，因为我们关心的触发条件在大部分情况下都不会发生，所以只有少量的查询触发了Status monitor所设置的阈值，绝大部分查询都浪费了时间和控制通道的带宽资源，也浪费能源。如果能允许控制器在交换机上设置触发条件，例如链路利用率达到90%，就可以在达到阈值的时候通知控制器，把“轮询”的模式变成“中断”模式，减少OpenFlow协议通信量。

这些固有的局限性以后可以通过改进OpenFlow协议和增加OpenFlow交换机功能来得到改善。

4) Bandwidth allocator

Bandwidth allocator是执行带宽预留策略的模块，它是对交换机里面Qos相关的特性进行配置，主要包括meter table和queue。

（1）meter table

meter table包含多条meter entry，meter entry由多个meter band，每个band可以配置不同的速率等级，将业务流限制到规定的速率，限速在硬件上是由rate limiter利用令牌桶/漏桶算法实现的。Meter的限速功能允许我们设置业务流的所占用的带宽上限，防止某些流吞噬带宽造成不公平或者资源滥用，从而给其他业务流预留带宽。meter band包头字段有：

```
struct ofp_meter_band_header {
    uint16_t type; /* One of OFPMBT_* */
    uint16_t len; /* Length in bytes of this band. */
    uint32_t rate; /* Rate for this band. */
    uint32_t burst_size; /* Size of bursts. */
};
```

rate字段规定了该meter bound对数据流所限制的最大传输速率。burst_size指定了令牌桶可以burst的最大数据量。

使用meter进行带宽预留的思路见图43，假设端口链路速率为1Gbps，需要给业务流1预留200Mbps带宽，可以将其转发到meter0，或者不设置meter，将出端口相同的其他流转发给meter1，设置meter1的限速速率为800Mbps，那么就可以给业务流1预留200Mbps带宽。

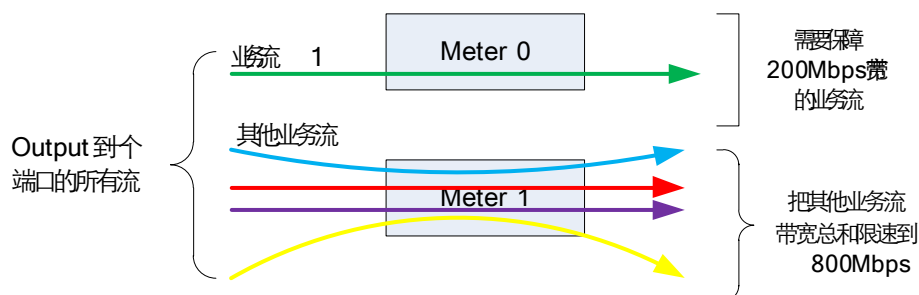


图43使用meter预留带宽

(2) 端口队列queues

根据OpenFlow标准，交换机的每个端口可以设置多个队列，每个队列可以设置最大速率max-rate和最小速率min-rate。最大速率可以用于流量限速，最小速率可以用于带宽预留。

使用queues进行限速的基本思路见图44，假设端口速率为1Gbps，需要给业务流1，业务流2，业务流3分别预留200Mbps，300Mbps，500Mbps的带宽，那么可以让这三个业务流分别转发到queue0、queue1、queue2，并且设置queue0、queue1、queue2的min-rate为200Mbps、300Mbps、500Mbps。

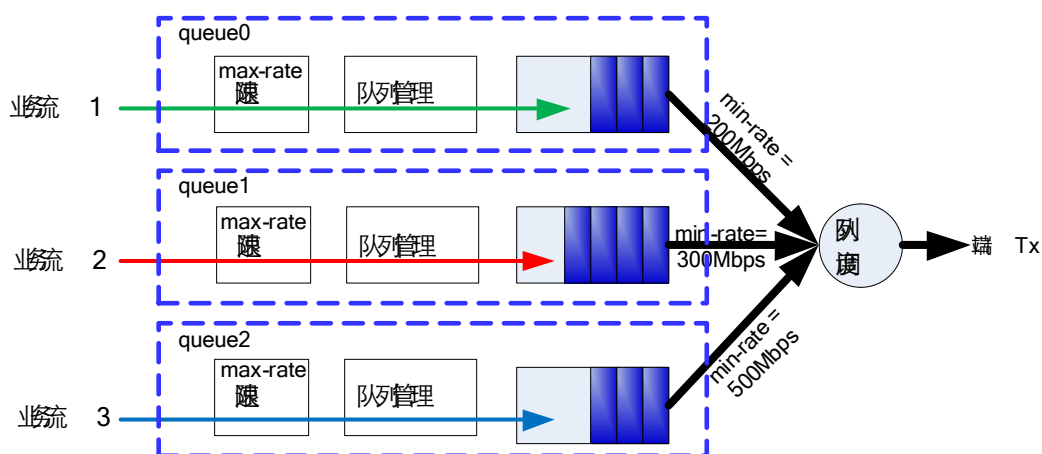


图44使用queues进行资源预留

不同的控制器对应的API不同，但是只要支持OpenFlow标准中定义的meter、queue特性，都可以用上面所介绍的方法去对meter和queue进行配置。

我们对程序功能进行了简化，基于Ryu控制器实现了一个带宽管理应用，利用OpenFlow交换机实现网络流量的细粒度带宽管理，流程图见图45。

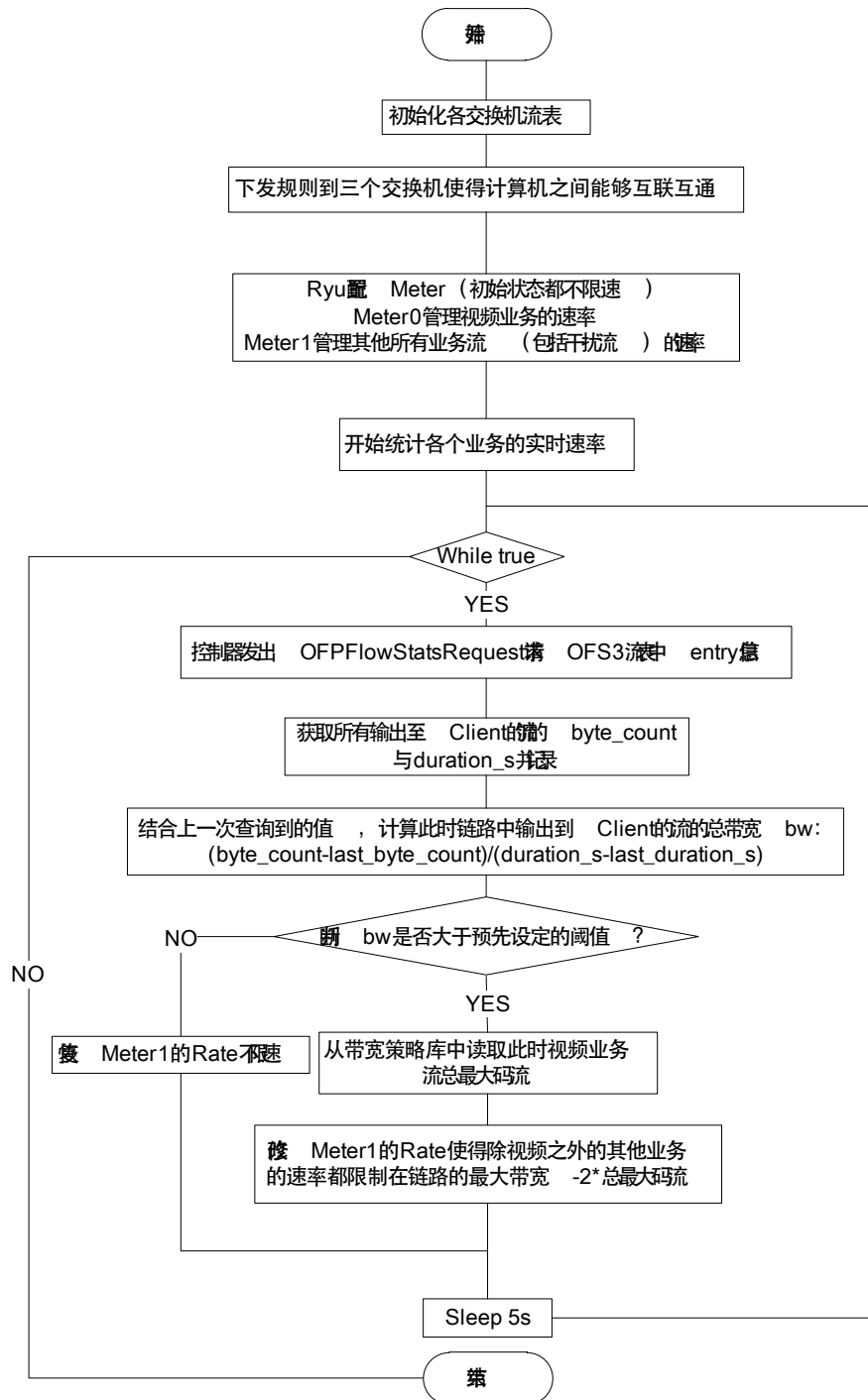


图45 流量带宽管理程序流程图

5. 系统测试

5.1 拓扑搭建

图51是我们的实验拓扑图，在这个实验环境中，OpenFlow网络由三台OpenFlow交换机OFS1、OFS2、OFS3和一台Ryu控制器组成。有3台主机连入OpenFlow网络，分别是Video server、Packet generator和Client，Video server和Packet generator分别接到OFS1和OFS2上，Client连接到

OFS3上。视频流量是我们需要保障带宽的业务，Packet generator产生网络背景干扰流量。

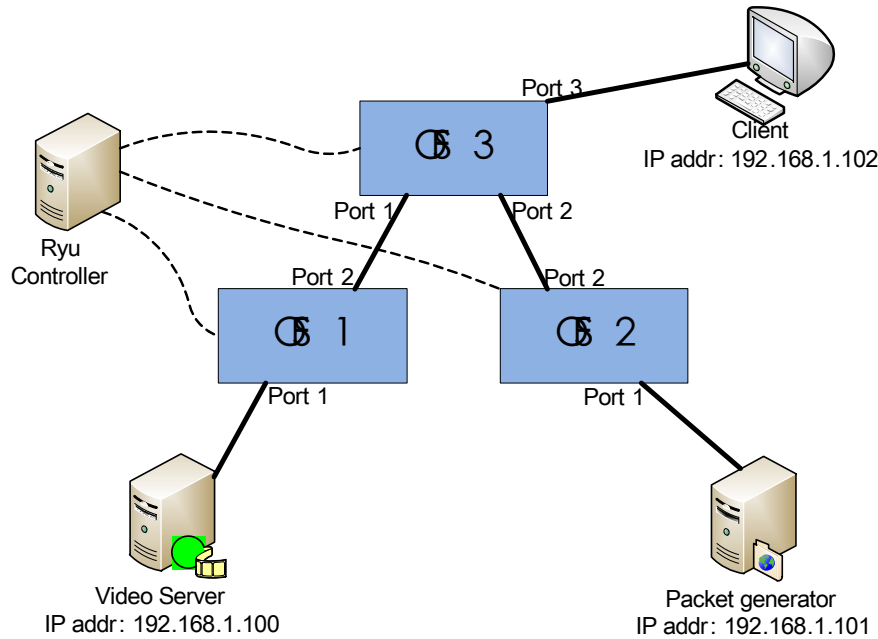


图51实验拓扑

三台OpenFlow交换机利用单独的网口作为控制口连接到一个Hub上，同时Ryu控制器也连接到该Hub上，由此建立了交换机的控制通道。交换机和控制器的配置见下表。

表51OpenFlow交换机和控制器配置

设备名	核心软件/硬件	OpenFlow版本	控制通道IP地址
Ryu Controller	Ryu软件	1.3	10.0.0.1
OFS1	ONetSwitch板卡	1.3	10.0.0.11
OFS2	ONetSwitch板卡	1.3	10.0.0.12
OFS3	ONetSwitch板卡	1.3	10.0.0.13

Video server采用VLC（VideoLAN）软件，将VLC配置成VOD模式，即可运行视频服务器。在实验中，指定视频服务器启动RTSP协议，使用RTSP进行视频传输。

视频Client也采用VLC播放器，在Client机器上使用VLC播放器打开指向Video server视频的URL，即可播放视频。

Packet generator使用了tcpreplay工具，我们首先使用tcpdump抓取500MB左右的网络流量，然后用tcpdump将网络流量的目标IP全部设为Client的IP地址，在实验过程中，重复发送该pcap文件，制造网络干扰流量。

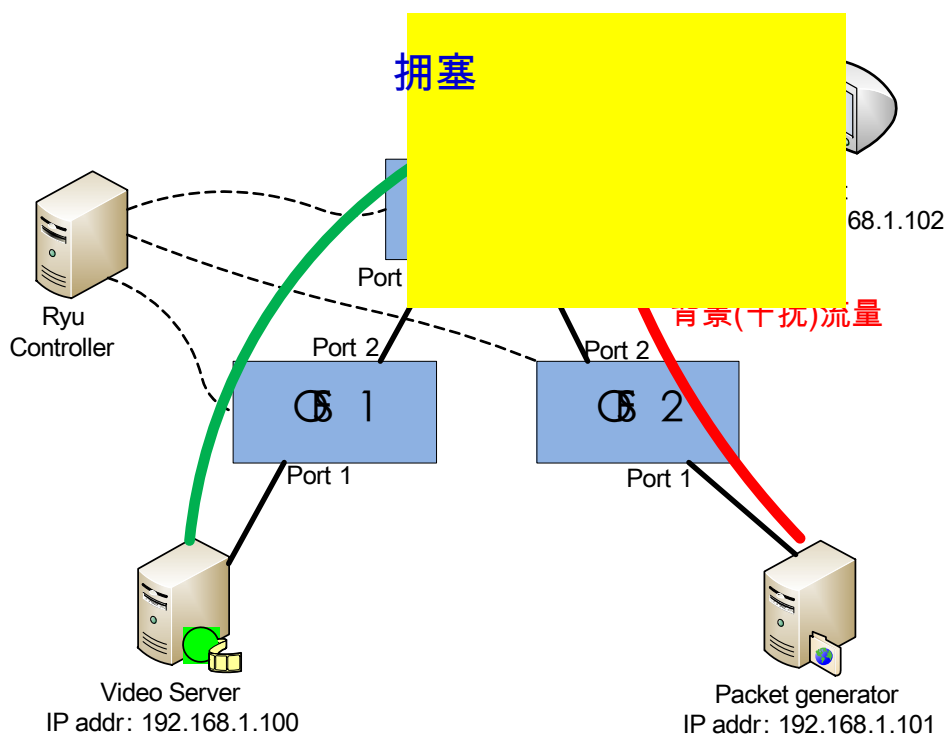


图52网络流量设计

5.2 实验流量设计

在本实验中，为了降低复杂度，我们设计了两条业务流（见图52），第一条业务流是从Video server到Client的视频传输流，这条流经过分别经过OFS1和OFS3。第二条业务流是从packet generator到Client的背景干扰传输流，这条流经过OFS2和OFS3。视频流对实时性和带宽的要求较高，视频流和干扰流在OFS3的port3上产生带宽竞争，产生拥塞，使得视频传输质量下降，通过限制干扰流的速率，或者增加视频流的带宽预留，从而给视频流提供了带宽保障。

经过wireshark抓包分析，视频流是使用UDP，而且每次请求视频流，源端口号和目标端口号都改变，所以无法像WEB或者FTP的应用那样，通过端口号进行流的识别，但是我们可以根据IP地址进行识别，因为通常来说提供某种业务的服务器IP地址数量是有限的。

干扰流是我们构造的数据包，以pcap文件的形式存在，文件大小为500MB，IP源地址是packet generator的IP地址，目标地址是Client的IP地址，全是1024字节大小的TCP数据包，利用tcpreplay发送。

由于拓扑比较简单，所以视频流和干扰流的区分非常容易，在本次实验中，我们只需要使用源IP+目的IP即可把不同的流区分开来。

实验中面临的一个问题是如何制造拥塞，由于OpenFlow交换机的端口是10M/100M/1000M自适应，为了达到链路阈值，我们将实验中使用的计算机网卡都调整到百兆模式（Linux下使用命令 `Ethtool -s Eth0 speed 100 duplex full`，windows下可以在属性中修改），所以各端口的链路容量将为百兆，播放码率为17Mbps的视频，加上tcpreplay的高速发包，总速率很容易达到链路阈值100Mbps。

5.3 实验过程

把拓扑搭建完成之后，我们需要启动交换机、控制器和带宽管理APP，然后启动server和client并进行通信，下面将对每一个步骤进行详细介绍。

1) 启动交换机

交换机运行Linux系统，开机运行启动脚本，在启动脚本中启动OpenFlow 软件并连接控制器：

```
#!/bin/bash
cd /root/ofsoftswitch13
#Start up the dataplane
./udatapath/ofdatapath --datapath-id=000000000001 --interfaces=Eth1,Eth2,Eth3,Eth4 ptcp:6632 --no-slicing &
sleep 5
#Start up the Security channel
./secchan/ofprotocol tcp:127.0.0.1:6632 tcp:10.0.0.15:6633 &
```

每个交换机的启动脚本中datapath-id的设置都不一样。OFS1、OFS2、OFS3的datapath-id分别为000000000001、000000000002、000000000023。

2) 启动控制器和Bandwidth manager

控制器启动脚本为：

```
#!/bin/bash
ryu-manager --verbose --observe-links \
ryu.topology.switches \
ryu/app/rest_topology.py \
ryu/app/ofctl_rest.py \
ryu/topology/bandwidthManager.py
```

使用REST API下发流表项，使服务器和客户端之间能够正常通信。根据拓扑结构，我们下发了表52所示的流表项。为了简化问题，我们根据拓扑结构，在保证互联互通的前提下尽量减少流表项的条数和流表的复杂度。

表52各交换机流表项配置

交换机名	序号	流表项	含义
OFS1	1	匹配in_port=1，执行ouput=2	使video server和OFS3互通
	2	匹配in_port=2，执行ouput=1	
OFS2	1	匹配in_port=1，执行ouput=2	使packet generator与OFS3互通
	2	匹配in_port=2，执行ouput=1	
OFS3	1	匹配Eth_type=0x0806,执行output = flood.	对ARP包执行洪泛
	2	匹配Eth_type=0x0800, ip_src=192.168.1.100,ip_dst=192.168.1.102，执行ouput=3 使用meter0，meter0不限速	根据源IP和目的IP 匹配，使video server和client互通

3	匹配Eth_type=0x0800, ip_src=192.168.1.102,ip_dst=192.168.1.100, 执行 ouput=1	根据源IP和目的IP 匹配, 使packet generator和client互 通
4	匹配Eth_type=0x0800, ip_src=192.168.1.101,ip_dst=192.168.1.102, 执行 ouput=3 使用meter1, meter1限制干扰数据流	
5	匹配Eth_type=0x0800, ip_src=192.168.1.102,ip_dst=192.168.1.101, 执行 ouput=2	

下发流表的REST API调用脚本如下:

```
#!/bin/bash
#initial sw1 and sw2
curl -X DELETE http://127.0.0.1:8080/stats/flowentry/clear/2
curl -X DELETE http://127.0.0.1:8080/stats/flowentry/clear/1
#Rules for sw1: (port1<-->port2)
curl -d '{"dpid":"1", "priority":"1",\
  "actions":[{"type":"OUTPUT","port":2}],\
  "match":{"in_port":1}}' http://127.0.0.1:8080/stats/flowentry/add
curl -d '{"dpid":"1", "priority":"2",\
  "actions":[{"type":"OUTPUT","port":1}],\
  "match":{"in_port":2}}' http://127.0.0.1:8080/stats/flowentry/add
#Rules for sw2: (port1<-->port2)
curl -d '{"dpid":"2", "priority":"1",\
  "actions":[{"type":"OUTPUT","port":2}],\
  "match":{"in_port":1}}' http://127.0.0.1:8080/stats/flowentry/add
curl -d '{"dpid":"2", "priority":"2",\
  "actions":[{"type":"OUTPUT","port":1}],\
  "match":{"in_port":2}}' http://127.0.0.1:8080/stats/flowentry/add
#Rules for sw3:
curl -d '{"dpid":"23", "flags":"KBPS", "meter_id":0,\
  "bands":[{"type":"DROP","rate":0}]]'\
  http://127.0.0.1:8080/stats/meterentry/add
curl -d '{"dpid":"23", "flags":"KBPS", "meter_id":1,\
  "bands":[{"type":"DROP","rate":0}]]' http://127.0.0.1:8080/stats/meterentry/add
curl -d '{"dpid":"23", "priority":"1",\
  "actions":[{"type":"OUTPUT","port":3},{type":"METER","meter_id":0}],\
  "match":{"dl_type":0x0800,"in_port":1}}' http://127.0.0.1:8080/stats/flowentry/add
curl -d '{"dpid":"23", "priority":"2",\
  "actions":[{"type":"OUTPUT","port":3},{type":"METER","meter_id":1}],\
  "match":{"dl_type":0x0800,"in_port":2}}' http://127.0.0.1:8080/stats/flowentry/add
curl -d '{"dpid":"23", "priority":"3",\
  "actions":[{"type":"OUTPUT","port":0xfffffff}],\
  "match":{"dl_type":0x0806}}' http://127.0.0.1:8080/stats/flowentry/add
curl -d '{"dpid":"23", "priority":"4",\
  "actions":[{"type":"OUTPUT","port":1}],\
  "match":{"dl_type":0x0800,"in_port":3,"nw_dst":"192.168.1.100/24"}}' http://127.0.0.1:8080/stats/flowentry/add
curl -d '{"dpid":"23", "priority":"5",\
  "actions":[{"type":"OUTPUT","port":2}],\
  "match":{"dl_type":0x0800,"in_port":3,"nw_dst":"192.168.1.101/24"}}' http://127.0.0.1:8080/stats/flowentry/add
```

基于Ryu的带宽控制应用的设计在5.4.1节介绍过了, 代码文件见附件资源。

3) 启动Video server和Client

启动video server, 如图53所示。该命令指定了使用554端口进行rtsp通信, vodconf.vlm文件配

置了vod的媒体库。

```
labsvr@labsvr: ~  
$ ./bin/bash  
vlc-wrapper -vvv --rtsp-host 0.0.0.0 --rtsp-port 554 --rtsp-caching 50000  
--vlm-conf /home/labsvr/vodconf.vlm
```

图53启动Video server

启动Client上的VLC播放器，请求，打开rtsp://10.0.0.15:554/tea，由于带宽非常充裕，所以视频播放流畅。

4) 动态带宽管理效果

带宽管理的动态效果详见我们的视频，在这里简要介绍一下。

首先，在Client机上启动VLC播放器，此时由于网络中只有视频流量，所以视频流独占带宽资源，播放流畅。见图54。



图54视频流畅播放

然后，在Packet generator上启动tcpreplay发送干扰流量数据包。此时在交换机OFS3的port3上发生拥塞。

```
tcpreplay -l 1000 -i Eth0 tcp_101_102.pcap
```

稍等几秒，等待视频缓冲耗尽，视频发生失真、画面卡住等现象，见图55。

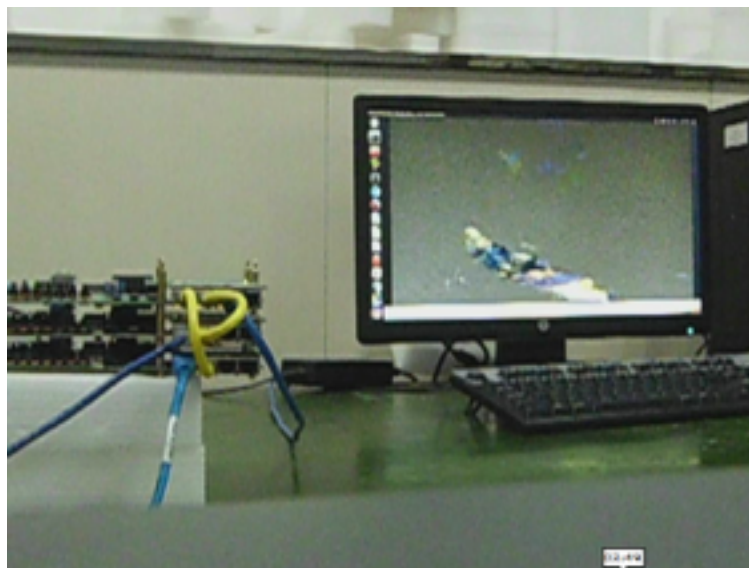


图55视频卡住

此时，带宽管理模块监测到端口流量超过阈值，我们首先延迟了10s，让用户能够观察到拥塞现象，10s之后，带宽管理模块重新配置了干扰流量的meter，将干扰流量进行限速，稍等几秒之后，视频恢复缓冲，开始流畅播放。

6. 总结

软件定义网络（SDN）是当前互联网的革命性技术之一，传统互联网所面临的低效率、无Qos、脆弱、安全风险高等难题在SDN中都有一些较好的解决思路，是当前的热门研究方向。SDN架构可以利用全局优化的特性有效地提升网络的效率，在网络管理方面具有显著的优势，云数据中心资源高度集中、网络性能要求苛刻等特点也使得它成为SDN实践的一片沃土。本项目聚焦SDN在数据中心网络中的应用，兼顾软硬件结合，主要的工作内容总结如下：

（1）基于可编程Zynq SOC设计SDN交换机硬件

OpenFlow交换机是SDN网络的基础设施，是搭建SDN环境的基础。目前商业OpenFlow交换机主要面向运营商，交换容量巨大，价格非常昂贵。软件交换机虽然价格低廉，简单易用，但是受限于软件运行速度，软件交换机仅能实现数十兆字节每秒的交换速率，性能太低。

我们的方案基于Zynq平台，采用软硬件协同设计的方法，在FPGA上实现OpenFlow交换逻辑，在ARM上运行OpenFlow交换机软件，实现了一个完整的四网口OpenFlow交换机，具有线速交换、功耗小、成本低、体积小、便携易用等特点。

（2）网络流量带宽管理

我们设计了一套基于SDN架构的业务带宽动态管理系统，实现了如下的功能：A）实时动态监测链路利用率、拥塞状态、业务流速率等状态信息，根据链路情况动态调整。B）进行细粒度的业务流带宽管理；C）利用OpenFlow提供的meter、table、端口queues等特性实现速率限制、带宽资源预留等多种功能；D）支持业务的实时带宽保障请求，并对带宽策略进行快速的自动化部署。最后基于Ryu编写了网络应用程序，实现了一个简化版系统，并利用ONetSwitch平台完成了业务流带宽保障实验。

(3) 网络路由故障恢复

在控制器上编写网络应用程序，对网络中的交换机端口、链路状态进行实时监控，从而动态地维护全网拓扑结构。当链路发生故障，导致链路通信中断的时候，根据当前的拓扑结构快速重新计算最短路径，通过更新交换机的流表项来对流量的路由路径进行重新调整，将发生故障的链路上的流量引导到冗余备份路径中，实现快速的故障恢复。

7. 参考文献

- [1] Fei Long,Zhigang Sun,Ziwen Zhang et al.Research on TCAM-based OpenFlow Switch Platform[C].//2012 International Conference on Systems and Informatics. [v.2].2012:1218-1221.
- [2] Nick Feamster, Jennifer Rexford, Ellen Zegura. The road to SDN[J]. ACM SIGCOMM Computer Communication Review, 2014, 44(2).
- [3] Lei Huang, Yuan Zhang. Research on Application of ONF OpenFlow-Based SDN Typical Architecture[J]. Applied Mechanics and Materials, 2014, 3082(543).
- [4] Si Quan Hu, Peng Yuan Zhou, Jun Feng Wang. The Inter-Datacenter Connection in SDN and Traditional Hybrid Network[J]. Advanced Materials Research, 2014, 3140(915).
- [5] Zhang Yang, Yan Gan Zhang, Xue Guang Yuan et al.. Design of SDN-Based Multi-Layer Control Plane for Elastic Wavelength Switch in Optical Network[J]. Applied Mechanics and Materials, 2013, 2700(411).
- [6] Masayoshi Kobayashi, Srini Seetharaman, Guru Parulkar et al.. Maturing of OpenFlow and Software-defined Networking through deployments[J]. Computer Networks, 2013.
- [7]N. Mckeown, T. Anderson, H. Balakrishnan, et al. OpenFlow: Enabling innovation in campus networks[J]. COMPUTER COMMUNICATION REVIEW,2008,38(2):69-74.
- [8] 杨安,毛席龙,吕高峰等.基于TLV结构的Openflow转发的研究与实现[C].//2012全国计算机体系结构学术年会论文集.2012:144-148,202.
- [9] 孙成龙,王换招,胡成臣等.一种容错的软件定义网络多控制器体系结构[C].//第二届中国互联网学术年会论文集.2013:319-325.
- [10] Bakshi, K..Considerations for Software Defined Networking (SDN): Approaches and use cases[C].//2013 IEEE Aerospace Conference.2013:1-9.