基于 OpenFlow 的未来互联网试验网①

韦世红, 卢 威

(重庆邮电大学 通信与信息工程学院, 重庆 400065)

摘 要: OpenFlow 是为支持互联网创新研究而提出的基于流分类的新型网络试验技术, 以 OpenFlow 为研究对象, 在 mininet 测试平台下, 利用 Open vSwitch 交换机搭建虚拟交换试验网络模型, 并进行了简单测试, 从而为未来互联网试验创新技术的分析和研究提供便利.

关键词: 未来互联网; OpenFlow; 网络创新; 虚拟交换机

Test Network of Future Internet Based on OpenFlow

WEI Shi-Hong, LU Wei

(School of Communication and Information Engineering, Chongqing University of Posts and Telecommunication, Chongqing 400065, China)

Abstract: OpenFlow, put forwarded to meet the requirement of Internet innovation research, is a new network test technology based on traffic classification. In this paper, we take OpenFlow as the research object to construct a virtual network test model in the mininet test platform by utilizing the Open vSwitch virtual machine. What's more, we also do some simple tests to facilitate the analysis and research on the futher Internet innovation technology.

Key words: future Internet; OpenFlow; network innovation; virtual switch

OpenFlow[1]是斯坦福大学 Clean Slate 计划资助的一个开放式协议标准,同时也是 GENI 计划的一个子项目. OpenFlow 是为支持网络创新研究而提出的基于流分类的新型网络试验技术, 满足可编程网络对试验新型网络协议的需求,主要用于在现有网络上部署新的协议和新的业务应用,协助致力于未来互联网研究的研究人员利用技术在现有网络上试验新型的网络协议, 其最终目标是重新设计互联网技术[2]. OpenFlow 作为互联网技术创新性研究的代表, 以其开源性、接口开放、支持控制的特点, 从提出便得到了广泛的关注和发展[3,4].

自开展未来互联网研究计划以来, 如何快速而方便的搭建大规模网络测试平台成为摆在研究者面前的一大难题, 在当前的网络上难以找到足够多的用户群和足够大的网络拓扑群来测试新协议、新运用的性能和功能, 因此, 虚拟化的网络试验网平台构建显得至关重要[5]. 本文主要以 OpenFlow 为研究对象, 利用 Open vSwitch 交换机搭建的 OpenFlow 虚拟交换试验网络模型进行简单的测试, 从而对未来互联网试验创新技术进行分析和研究.

1 OpenFlow技术

OpenFlow 作为一种新型网络试验技术, 将原来完全由交换机/路由器控制的报文转发过程转化为由 OpenFlow 交换机(OpenFlow Switch)和控制服务器(Controller)来共同完成, 实现了数据转发和路由控制的分离. 控制器可以通过事先规定好的接口操作来控制 OpenFlow 交换机中的流表, 从而达到控制数据转发的目的.

OpenFlow 的流表(Flow Table)是 OpenFlow 交换机进行数据转发的核心结构. 如图 1 所示 Flow Table 的每一个 Entry 包括 3 个部分: 规则, 操作, 状态:

① 规则(Rule): 用来定义数据包的特征, 其中包括 10 个包头域, OpenFlow 里 flow 定义十分宽泛, 除了

① 基金项目:重庆市重点自然科学基金(cstc2012jjB0168)

收稿时间:2012-10-04;收到修改稿时间:2012-11-11

传统的 7 元组之外增加了交换端口、 以太网类型、Vlan ID, 头域是个十元组, 是流表项的标识;

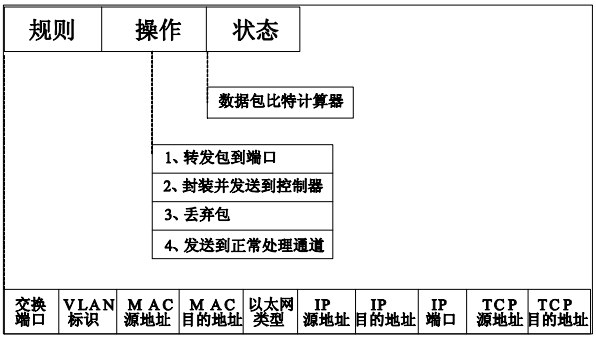

图 1　Flow Table Entry

② 操作(Action): 定义了对数据包进行的处理操作类型, 包括转发数据包到所有出口(不包括入口)、封装并转发数据包到控制器、丢弃数据包、将数据包转发到绑定到某个端口的队列中等;

③ 状态(Status): 主要是用来统计 OpenFlow 中相关表数据流和端口等相关数据的参数.

2　OpenFlow网络组成

OpenFlow 网络由 OpenFlow 交换机、FlowVisor 和 Controller(控制器)三部分组成. OpenFlow 交换机进行数据层的转发; FlowVisor 对网络进行虚拟化; Controller 对网络进行集中控制, 实现控制层的功能. OpenFlow 网络的结构示意图如下:

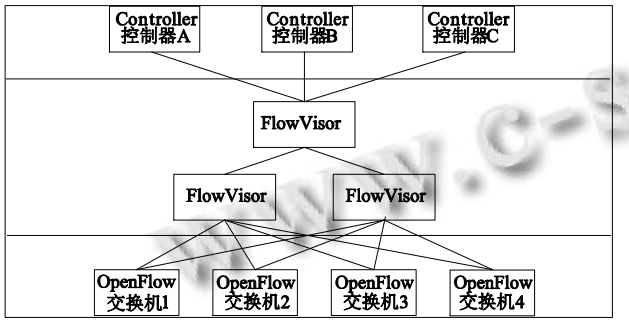

图 2　OpenFlow 网络的结构

2.1 OpenFlow 交换机

OpenFlow 交换机是整个 OpenFlow 网络的核心, 它主要管理数据层的接收、转发. 每个 OpenFlow 交换机都有一张流表, 进行数据包转发和查找. 可以采用 OpenFlow 协议通过一个安全通道连接到外部 Controller, 对流表进行管理和查询. 其简要工作流程是在接收到数据包后, 首先在本地的流表上查找转发目标的端口, 如果没有找到, 则把数据包转发给外部 Controller, 由控制层决定合适的转发端口.

OpenFlow 交换机由流表、安全通道和 OpenFlow 协议三部分组成. 如图 3:

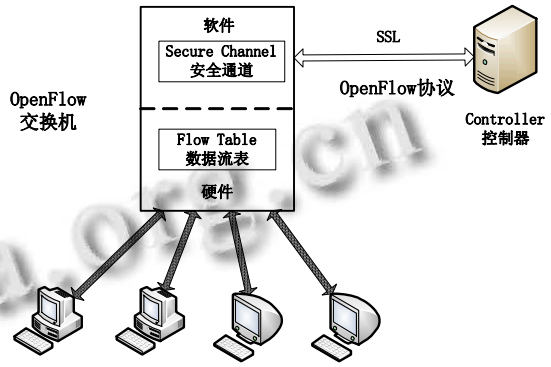

图 3　OpenFlow 交换机组成

① 流表(Flow Table)

流表是 OpenFlow 交换机进行转发策略控制的核心, 由很多个流表项组成, 每个流表项就是一个转发规则. 用来指示 OpenFlow 交换机如何处理数据流.

每个流表项包括三个部分, 包头域(Header field), 计数器(Counters), 行动(Actions).

② 安全通道(Secure Channel)

安全通道是连接 OpenFlow 交换机和 Controller 的接口. 用来实现 OpenFlow 交换机与 Controller 之间的通信, 指令和数据包的安全传递等.

③ OpenFlow 协议(OpenFlow Protocol)

用来描述 Controller 和 OpenFlow 交换机之间通信所用的标准, 以及 Controller 和 OpenFlow 交换机的接口标准. OpenFlow 协议支持的三种信息类型: Controller-to-Switch, Symmetric(对称)和 Asynchronous (异步).

2.2 支持虚拟化网络的 FlowVisor

FlowVisor 和计算机虚拟化原理相似, 它是位于软件和硬件结构元件之间的一个网络虚拟层, 允许多个控制器同时控制一台 OpenFlow 交换机, 但是每个控制器仅仅可以控制经过这个 OpenFlow 交换机的某一个独立的虚拟网络. 因此使用 FlowVisor 建立的实验网络平台可以在不影响商业流转发速度的情况下, 允许多个网络实验在不同的虚拟网络上同时进行. 更重要的是, FlowVisor 与一般的商用交换机是兼容的, 不

需要使用网络处理器和 FPGA 等其他可编程硬件.

2.3 控制器 Controller

OpenFlow 最大的创新就是实现了数据层和控制层的分离, 其中 OpenFlow 交换机负责数据层的转发, 而 Controller 实现控制层的功能. Controller 通过 OpenFlow 协议这个标准接口对 OpenFlow 交换机中的流表进行控制, 从而实现数据包的转发和对整个网络的集中控制. 同时控制层的分离也可以让我们在远端直接对 Controller 进行可编程操作, 避免了直接对交换机进行操作.

3 虚拟交换网络试验网构建及测试

3.1 试验网测试环境构建

① Linux 操作系统

由于 Ubuntu10.04 LTS 这个版本拥有良好的稳定性和可靠性, 因此选择此版本作为试验网测试环境构建的操作平台.

② 多层虚拟交换机 Open vSwitch

Open vSwitch 简称 OVS, 是一个产品级的多层虚拟交换机. 通过可编程扩展, 可实现大规模网络自动化, 包括虚拟网络的配置、管理及维护, 同时也能检测多物理主机在动态虚拟环境中的流量情况.

③ 轻量级网络测试平台 mininet

mininet 是一套进程虚拟化平台, 由 stanford 大学 Nick McKeown 的研究小组基于 Linux Container 架构研究开发完成, 支持系统级的还原测试、复杂网络拓扑、自定义拓扑, 提供 python API 接口,高扩展性, 用于在 Linux 进程下定制一个任意数量的软件定义的网络(software-defined Networks, SDN).

④ 网络封包分析软件 Wireshark

Wireshark(前称 Ethereal)是一个网络封包分析软件. 网络封包分析软件的功能是抓取网络封包, 并尽可能显示出最为详细的网络封包资料.

3.2 试验网测试及结果分析

这里通过 mininet 的操作命令, 对自定义的小型测试网络进行验证:

① 登陆到虚拟机操作界面并启动终端打开 wireshark, 使其在后台运行, 执行:

root@ubuntu10.04:~# wireshark &

② 利用 mininet 提供的 python api, 我们可以方便的创建自己的自定义网络拓扑. 下面介绍的是利用 python api 创建如图 4 所示的网络拓扑.

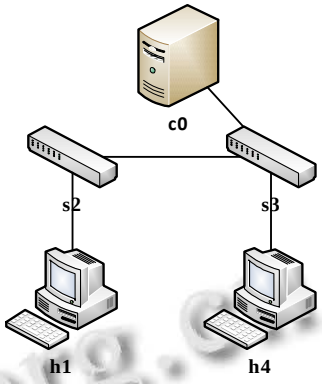

图 4 试验网络拓扑

3.2.1 编写 python api 文件

主要有两种方式: 第一种是直接新建一个空文件编写代码, 完成后保存为.py 文件即可; 第二种是用 Ubuntu10.04 提供的 GNU emacs 23 编辑器写 python 代码, 这里选择第二种方式.

将文件源代码之后保存到 mininet/custom 目录下方便调用. 这里命令了名为 newnet.py 的文件, newnet.py 文件程序代码如下:

```
"""the code introduction
Two directly connected switches plus a host for each switch:
    host --- switch --- switch --- host
Adding the 'topos' dict with a key/value pair to generate our newly defined
topology enables one to pass in '--topo=mytopo' from the command line.
"""
from mininet.topo import Topo, Node
class MyTopo( Topo ):
"Simple topology example."
def __init__( self, enable_all = True ):
    "Create custom topo."
# Add default members to class.
super( MyTopo, self ).__init__()
        # Set Node IDs for hosts and switches
        leftHost = 1
        leftSwitch = 2
        rightSwitch = 3
```

```
        rightHost = 4
    # Add nodes
    self.add_node(leftSwitch,Node(is_switch=True) )
    self.add_node(rightSwitch,Node( is_switch=True) )
    self.add_node( leftHost, Node( is_switch=False) )
    self.add_node(rightHost,Node( is_switch=False) )
        # Add edges
        self.add_edge( leftHost, leftSwitch )
        self.add_edge( leftSwitch, rightSwitch )
        self.add_edge( rightSwitch, rightHost )
        # Consider all switches and hosts 'on'
        self.enable_all()
topos = { 'mytopo': ( lambda: MyTopo() ) }
```

3.2.2 启动终端，执行以下代码:

root@ubuntu10.04:~# mn – custom /home/openflow /mininet/custom/newnet.py --topo mytopo 得到结果如下:

```
root@ubuntu10.04:~# mn --custom /home/openflow/mininet/custom/newnet.py --topo my
topo
custom in sys.argv
*** Adding controller
*** Creating network
*** Adding hosts:
h1 h4
*** Adding switches:
s2 s3
*** Adding links:
(h1, s2) (s2, s3) (s3, h4)
*** Configuring hosts
h1 h4
*** Starting controller
*** Starting 2 switches
s2 s3
*** Starting CLI:
mininet>
```

图 5　mininet 启动信息

分析可知, mininet 通过调用 newnet.py 文件创建了一个拥有两个主机(h1, h4)、两个 OpenFlow 交换机(s2, s3)和一个控制节点(c0)的网络.

3.2.3 对 newnet 的网络状态检测

注意 wireshark 监控的接口为 s2-eth1. 分别利用 nodes、net、dump 命令对其测试, 结果如下:

```
mininet> nodes
available nodes are:
c0 h1 h4 s2 s3
mininet> net
s2 <-> h1-eth0 s3-eth1
s3 <-> s2-eth2 h4-eth0
mininet> dump
c0: IP=127.0.0.1 intfs= pid=2242
s2: IP=None intfs=s2-eth1,s2-eth2 pid=2245
s3: IP=None intfs=s3-eth1,s3-eth2 pid=2246
h1: IP=10.0.0.1 intfs=h1-eth0 pid=2243
h4: IP=10.0.0.4 intfs=h4-eth0 pid=2244
```

图 6　newnet 网络基本信息

利用 nodes、net、dump 命令可以分别得到网络的节点、链路状态以及各个节点的信息.

3.2.4 对 newnet 的网络连通测试

在终端下输入: h1 ping -c 4 h4, 从 h1 发 4 个数据包到 h4, 并用 wireshark 抓包分析, 结果如下图:

```
mininet> h1 ping -c 4 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=10.2 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=5.55 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.083 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.082 ms

--- 10.0.0.4 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3013ms
rtt min/avg/max/mdev = 0.082/3.996/10.266/4.253 ms
```

图 7　newnet ping 测试信息

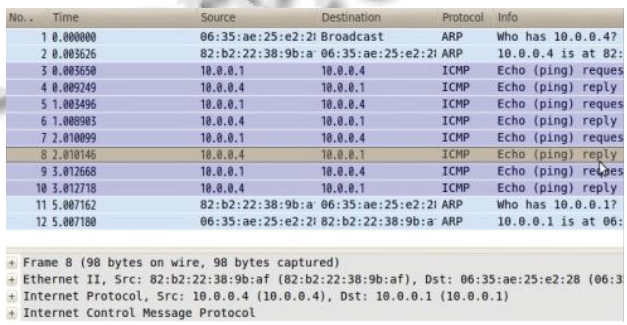

图 8　wireshark 抓包信息

从上面两张图分析可知: h1 和 h4 之间可以直接通信，但是 ping 前两个数据包的时间偏大. 利用 wireshark 监控 lo 口，重新 ping 一次可知，由于 OpenFlow 协议创建新的流表项, 造成前两个时间偏大(wireshark 监控 lo 口可以看到 OpenFlow 的网包).

3.2.5 对 newnet 网络的传输测试

使用 iperf、iperfudp 命令对 h1、h4 之间的 TCP、UDP 传输速度测试如下:

```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['2.03 Gbits/sec', '2.04 Gbits/sec']
mininet> iperfudp
*** Iperf: testing UDP bandwidth between h1 and h4
*** Results: ['10M', '10.0 Mbits/sec', '10.0 Mbits/sec']
```

图 9　newnet TCP/UDP 传输速度测试

使用 mininet 创建的小型网络拓扑，不仅方便快捷, 而且在 OpenFlow 协议的转发机制下，虚拟客户机之间的传输速度也很快.

以上是搭建了小型的网络进行验证测试，下一步在 mininet 基础上，采用虚拟化技术在 PC 机上虚拟更多交换机，并增加 NOX 网络操作系统，可以搭建大型网络级的虚拟网络，OpenFlow 的控制器功能优势将能得到更大的体现.

(下转第 97 页)

在整个运动过程中都不会发生碰撞. 这样的避碰算法是有效的, 可以很好的避免两只机械臂在对人体背部掌推时的碰撞, 算法达到了预期的结果.

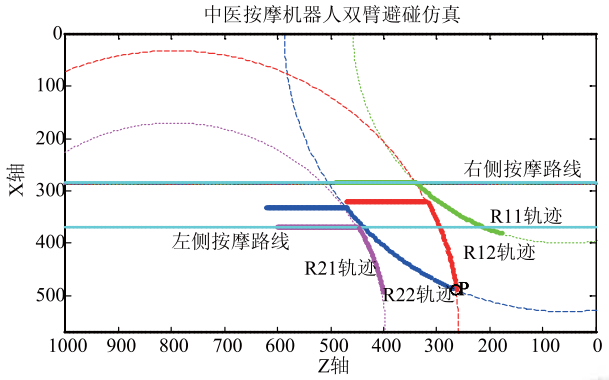

图 5 按摩机器人双臂碰撞情况的运动学仿真

3 结论

本文中运用平面几何的方法设计的协调运动控制策略, 可以有效的实现双臂在反向掌推时的避碰, 简单, 可靠, 为其他按摩手法的避碰提供了思路, 也可为研究其他系统的双臂避碰问题参考.

当然, 中医按摩机器人双臂间的协调控制问题还有许多待研究之处, 例如, 可采用新的控制算法, 通过动用更多的关节, 更好的实现避碰.

参考文献

1 罗才贵,刘明军,陈立.实用中医推拿学[M].四川科学技术出版社,2004,06:3–5.

2 曹仁发.推拿功法与治病[M].上海:上海科学技术文献出版社,1992,07:2–4.

3 周长伟,宋胜捷,于豪光,王伟生.一种新型串联按摩机器人的设计与实现.《机器人技术与应用》.2010,(6):36.

4 高焕兵,鲁守银,王涛,刘存根,康炳元,季远,毕鸿雁.中医按摩机器人研制与开发.机器人.2011,33(5):553–554.

5 肖南峰.智能机器人[M]华南理工大学出版社,2008,01:140–144.

6 黄献龙,梁斌,吴宏鑫.机器人避碰规划综述.航天控制,2002,(1):34–46.

7 谭林.中医按摩机器人的运动控制系统研究[D].山东:山东建筑大学,2011,06:41–42.

8 李恩,梁自泽,谭民.约束条件下的巡线机器人逆运动学求解.控制理论与应用.2006,23(1):43–44.

9 刘全,禹华刚,刘冰.基于几何分析的机械臂运动路径规划问题.2008,38(14):122–124.

10 陈峰,丁富强,赵锡芳.双臂机器人无碰撞运动规划.上海交通大学机器人研究所,机器人,2002,24(2):112–114.

11 Yong-Jin Liu,Zhan-Qing Chen,Kai Tang.Construction of Iso-contours,Bisectors and Voronoi Diagrams on Triangulated Surfaces.IEEE Transactions on Pattern Analysis and Machine Intelligence.2011,33(8):1502–1517.

12 Nikolaus Vahrenkamp,Dmitry Berenson,Tamim Asfour,James Kuffner,R ¨ udiger Dillmann.Humanoid Motion Planning for Dual-Arm Manipulation and Re-Grasping Tasks. The 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems.2009:2464–2470.

(上接第 24 页)

4 结语

本文在介绍未来互联网发展的前提下, 重点介绍了软件自定义网络的 OpenFlow 技术, 并且利用其开源性, 在 mininet 网络测试环境下, 对其进行了验证和小型网络平台的搭建. OpenFlow 基于它的开源性以及在可编程性和虚拟化等方面具备较大的技术优势, 在实现虚拟网络平台的搭建, 研究未来网络及网络创新提供了方便. 可以预见, 在未来互联网研究的研究过程中, 以 OpenFlow 为代表的交换创新技术将开启未来互联网的新时代.

参考文献

1 McKeown N, Anderson T, et al. OpenFlow: Enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 2008,38(2):69–74.

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

3 韦新军.OpenFlow 交换机模型及关键技术研究与实现[硕士学位论文].长沙:国防科技大学,2008.

4 东南大学计算机科学与工程学院.互联网改革新军 OpenFlow 论坛.中国教育网络,2008,12:26–27.

5 贺鹏,洪涛,谢高岗,等.支持未来网络创新的可编程虚拟化路由器技术.中兴通讯技术,2011,17(2):5–6.