Mininet – “懒惰”网络研究者的福音

Saturday, January 8 2011, 1:07 PM

注:本文的一个精简版本2011/01/08发表于弯曲评论，最新版本可以从http://security.riit.tsinghua.edu.cn/~bhyang/publications.html#tech_review找到。

【摘要】在本系列前两篇文章里，我们分别介绍了软件定义网络的两大利器——OpenFlow和Open vSwitch。不少研究者可能很有兴趣尝试一下，却拿不出太多的时间。本文将介绍一套强大的轻量级网络研究平台——mininet，通过它相信大家可以 很好地感受到软件定义网络的魅力。

概述篇

作为研究者，你是否想过，在自己的个人笔记本上就可以搭建一套媲美真实硬件环境的复杂网络，并轻松进行各项实验？无论是用专业级的硬件实验平台，还 是用传统的虚拟机，都显得太过昂贵，且十分不方便进行操作。如果你有类似的需求，不妨试试mininet，绝对是“懒惰”却又追求效率的研究人员的福音。

Stanford大学Nick McKeown的研究小组基于Linux Container架构，开发出了这套进程虚拟化的平台。在mininet的帮助下，你可以轻易的在自己的笔记本上测试一个软件定义网络（software-defined Networks），对基于Openflow、Open vSwitch的各种协议等进行开发验证，或者验证自己的想法。最令人振奋的是，所有的代码几乎可以无缝迁移到真实的硬件环境中，学术界跟产业界再也不是 那么难以沟通了。想想吧，在实验室里，一行命令就可以创建一个支持SDN的任意拓扑的网络结构，并可以灵活的进行相关测试，验证了设计的正确后，又可以轻 松部署到真实的硬件环境中。

mininet作为一个轻量级软定义网络研发和测试平台，其主要特性包括

- 支持Openflow、OpenvSwitch等软定义网络部件
- 方便多人协同开发
- 支持系统级的还原测试
- 支持复杂拓扑、自定义拓扑
- 提供python API
- 很好的硬件移植性（Linux兼容），结果有更好的说服力
- 高扩展性，支持超过4096台主机的网络结构

实战篇

获取镜像

官方网站已经提供了配置好相关环境的基于Debian Lenny的虚拟机镜像，下载地址为http://openflowswitch.org/downloads/OpenFlowTutorial-081910.vmware.zip， 压缩包大小为700M左右，解压后大小为2.1G左右。虚拟机镜像格式为vmware的vmdk，可以直

接使用vmware workstation或者virtualbox等软件打开。如果使用QEMU和KVM则需要先进行格式转换。后面我们就以这个虚拟os环境为例，介绍 mininet的相关功能。

如果使用virtualbox进行加载，需要注意

- 尽量使用最新版本，
- host操作系统需要支持pae，并在virtualbox中打开pae支持。

登录镜像

默认用户名密码均为openflow，建议通过本地利用ssh登录到虚拟机上使用（可以设置自动登录并将X重定向到本地），比较方便操作。
注意事项：

- 建议将guest主机采用bridge方式联网，以获取host可见的独立IP；也可采用为guest配置两块网卡方式，一块采用NAT，一块采用host-only，但host-only的网卡可能无法自动dhcp到地址，需要手动配置（ifconfig eth1 ip/mask）
- 将host机.ssh目录下id_rsa.pub复制到guest机的.ssh目录下，并写入authorized_keys，实现自动认证

简单测试

创建网络

mininet的操作十分简单，启动一个小型测试网络只需要下面几个步骤。

1. 登录到虚拟机命令行界面，打开wireshark，使其后台运行, 命令为sudo wireshark &
2. 启动mininet，命令为sudo mn，则默认创建如下图所示的网络拓扑
3. 经过短暂的等待即可进入以mininet>引导的命令行界面

好了，从现在开始，我们就拥有了一个1台控制节点(controller)、一台交换(switch)、两台主机(host)的网络，并且用wireshark进行观测。下面进行几项简单的测试。

注意：使用wireshark检测lo网卡，并通过过滤of协议可以看到OpenFlow的网包。

查看信息

查看全部节点：

mininet> nodes

available nodes are:

c0 h2 h3 s1

查看链路信息：

mininet> net

s1 <-> h2-eth0 h3-eth0

输出各节点的信息：

mininet> dump

c0: IP=127.0.0.1 intfs= pid=1679

s1: IP=None intfs=s1-eth1,s1-eth2 pid=1682

h2: IP=10.0.0.2 intfs=h2-eth0 pid=1680

h3: IP=10.0.0.3 intfs=h3-eth0 pid=1681

对节点进行单独操作

如果想要对某个节点的虚拟机单独进行命令操作，也十分简单，格式为 node cmd。例如查看交换机s1上的网络信息，我们只需要在执行的ifconfig命令前加上s1主机标志即可，即 s1 ifconfig，同样，如果我们想用ping 3个包的方法来测试h2跟h3之间连通情况，只需要执行 h2 ping -c 3 h3 即可。得到的结果为

mininet> h2 ping -c 3 h3

PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.

64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=7.19 ms

64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.239 ms

64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.136 ms

— 10.0.0.3 ping statistics —

3 packets transmitted, 3 received, 0% packet loss, time 2006ms

rtt min/avg/max/mdev = 0.136/2.523/7.194/3.303 ms

在本操作执行后，可以通过wireshark记录查看到创建新的流表项的过程，这也是造成第一个ping得到的结果偏大的原因。更简单的全网络互ping测试命令是pingall，会自动所有主机节点逐对进行ping连通测试。

常用功能

快捷测试

除了cli的交互方式之外，mininet还提供了更方便的自动执行的快捷测试方式，其格式为sudo mn --test cmd，即可自动启动并执行cmd操作，完成后自动退出。
例如 sudo mn --test pingpair，可以直接对主机连通性进行测试，sudo mn --test iperf启动后直接进行性能测试。用这种方式很方便直接得到实验结果。

自定义拓扑

mininet提供了python api，可以用来方便的自定义拓扑结构，在mininet/custom目录下给出了几个例子。例如在topo-2sw-2host.py文件中定义了一个mytopo，则可以通过--topo选项来指定使用这一拓扑，命令为
sudo mn --custom ~/mininet/custom/topo-2sw-2host.py --topo mytopo --test pingall
同样的，我们可以通过下面的python脚本来完成对一个2层tree拓扑网络的测试

```
from mininet.net import Mininet
from mininet.topolib import TreeTopo
tree4 = TreeTopo(depth=2,fanout=2)
net = Mininet(topo=tree4)
net.start()
h1, h4  = net.hosts[0], net.hosts[3]
print h1.cmd('ping -c1 %s' % h4.IP())
net.stop()
```

使用友好的mac编号

默认情况下，主机跟交换机启动后分配的MAC地址是随机的，这在某些情况下不方便查找问题。可以使用--mac选项，这样主机跟交换机分配到的MAC地址跟他们的ID是一致的，容易通过MAC地址较快找到对应的节点。

使用XTerm

通过使用-x参数，mn在启动后会在每个节点上自动打开一个XTerm，方便某些情况下的对多个节点分别进行操作。命令为

sudo mn -x

在进入mn cli之后，也可以使用 xterm node 命令指定启动某些节点上的xterm，例如分别启用s1跟h2上的xterm，可以用

xterm s1 h2

链路操作

在mn cli中，使用link命令，禁用或启用某条链路，格式为 link node1 node2 up/down，例如临时禁用s1跟h2之间的链路，可以用

link s1 h2 down

指定交换机跟控制器类型

通过--switch 选项跟--controller选项可以分别指定采用哪种类型的交换机跟控制器，例如使用用户态的交换

sudo mn --switch user

使用OpenvSwitch

sudo mn --switch ovsk

使用NOX pyswitch

1）首先确保nox运行

cd $NOX_CORE_DIR

./nox_core -v -i ptcp:

然后ctrl-c 杀死nox进程

2）然后指定nox交换机

sudo -E mn --controller nox_pysw

注意：通过-E选项来保持预定义的环境变量（此处为NOX_CORE_DIR）。

名字空间

默认情况下，主机节点有用独立的名字空间（namespace），而控制节点跟交换节点都在根名字空间（root namespace）中。如果想要让所有节点拥有各自的名字空间，需要添加 --innamespace 参数，即启动方式为 sudo mn --innamespace
注意：为了方便测试，在默认情况下，所有节点使用同一进程空间，因此，在h2跟h3或者s1上使用ps查看进程得到的结果是一致的，都是根名字空间中的进程信息。

启动参数总结

```
-h, --help            show this help message and exit
--switch=SWITCH       [kernel user ovsk]
--host=HOST           [process]
--controller=CONTROLLER [nox_dump none ref remote nox_pysw]
--topo=TOPO           [tree reversed single linear minimal],arg1,arg2,...argN
-c, --clean           clean and exit
--custom=CUSTOM       read custom topo and node params from .py file
--test=TEST           [cli build pingall pingpair iperf all iperfudp none]
-x, --xterms          spawn xterms for each node
--mac                 set MACs equal to DPIDs
--arp                 set all-pairs ARP entries
-v VERBOSITY, --verbosity=VERBOSITY [info warning critical error debug output]
--ip=IP               [ip address as a dotted decimal string for aremote controller]
--port=PORT           [port integer for a listening remote controller]
--innamespace         sw and ctrl in namespace?
--listenport=LISTENPORT [base port for passive switch listening controller]
--nolistenport        don't use passive listening port
--pre=PRE             [CLI script to run before tests]
```

--post=POST [CLI script to run after tests]

常用命令总结

help 默认列出所有命令文档，后面加命令名将介绍该命令用法
dump 打印节点信息
gterm 给定节点上开启gnome-terminal。注：可能导致mn崩溃
xterm 给定节点上开启xterm
intfs 列出所有的网络接口
iperf 两个节点之间进行简单的iperf TCP测试
iperfudp 两个节点之间用制定带宽udp进行测试
net 显示网络链接情况
noecho 运行交互式窗口，关闭回应（echoing）
pingpair 在前两个主机之间互ping测试
source 从外部文件中读入命令
dpctl 在所有交换机上用dptcl执行相关命令，本地为tcp 127.0.0.1:6634
link 禁用或启用两个节点之间的链路
nodes 列出所有的节点信息
pingall 所有host节点之间互ping
py 执行python表达式
sh 运行外部shell命令
quit/exit 退出

其他操作

执行sudo mn -c会进行清理配置操作，适合故障后恢复。
执行exit会退出mininet的cli，同时给出运行时间统计。
py cmd 使用python来执行cmd。
测试mininet启动后立刻关闭的时间可以用 sudo mn --test none

高级篇

下面我们通过一个具体管理of switch的例子来介绍一些比较高级的命令。
首先，启动vm，然后执行

```
sudo mn --topo single,3 --mac --switch ovsk --controller remote
```

生成一个小的网络，三台主机连到一台交换机上，交换机为ovs交换机，指定remote控制器（默认为本地）。
dpctl
执行

```
dpctl show tcp:127.0.0.1:6634
```

可以查看到交换机的端口等基本情况，其中tcp端口6634是默认的交换机监听端口。

执行

```
dpctl dump-flows tcp:127.0.0.1:6634
```

可以看到更详细的流表信息。
此时，流表为空，执行h2 ping h3无法得到响应。因此我们需要通过dpctl手动添加流表项，实现转发。
命令为

```
dpctl add-flow tcp:127.0.0.1:6634 in_port=1,actions=output:2
dpctl add-flow tcp:127.0.0.1:6634 in_port=2,actions=output:1
```

此时查看流表可以看到新的转发信息，同时可以在h2和h3之间ping通。

控制器
通过执行

```
controller ptcp:
```

我们可以启动一个简单的控制器，默认没有任何流表项，仅仅作为一台带学习功能的交换机。控制器默认监听端口是6633。以下控制器与交换机之间的消息交互过程，可以通过wireshark，配置of过滤器观察到。

Message	Type	Description
Hello	Controller->Switch	following the TCP handshake, the controller sends its version number to the switch.
Hello	Switch->Controller	the switch replies with its supported version number.
Features Request	Controller->Switch	the controller asks to see which ports are available.
Set Config	Controller->Switch	in this case, the controller asks the switch to send flow expirations.
Features Reply	Switch->Controller	the switch replies with a list of ports, port speeds, and supported tables and actions.
Port Status	Switch->Controller	enables the switch to inform that controller of changes to port speeds or connectivity. Ignore this one, it appears to be a bug.

同样，我们可以用wireshark观察到当第一次有ping包从h2发到h3时，控制器如何自动添加相

应的表项到交换机。wireshark相应的过滤器为

```
of && (of.type != 3) && (of.type != 2)
```

相关的消息过程为

Message	Type	Description
Packet-In	Switch->Controller	a packet was received and it didn't match any entry in the switch's flow table, causing the packet to be sent to the controller.
Packet-Out	Controller->Switch	controller send a packet out one or more switch ports.
Flow-Mod	Controller->Switch	instructs a switch to add a particular flow to its flow table.
Flow-Expired	Switch->Controller	a flow timed out after a period of inactivity.

使用NOX
首先确定没有其他控制器在运行（占据6633端口）

```
sudo killall controller
```

同样的，启动mininet

```
sudo mn --topo single,3 --mac --switch ovsk --controller remote
```

然后启动nox，默认路径为~/noxcore/build/src，重新打开一个ssh终端执行

```
./nox_core -v -i ptcp: pytutorial
```

会自动打开运行tutorial应用的nox，打印出详细的调试信息，并监听6633端口。
直到打印出类似如下信息，说明交换机已经成功连接到nox。

```
00039|nox|DBG:Registering switch with DPID = 1
```

通过互ping测试，各个主机连通，此时switch等同于一个hub。

然后通过修改**~/noxcore/src/nox/coreapps/tutorial/pytutorial.py**中代码，让nox工作成一个带学习功能的交换机。相关命令参考ofinclude代码，以及nox对各个包的解析代码目录：

```
~/noxcore/src/nox/lib/packet/
```

通过编写nox程序，我们可以让交换机的行为更加智能化、复杂化。为了测试我们编写的nox程序，我们可以使用cbench来进行测试。

多条配置命令
可以写到一个文件中，用mn直接调用。例如脚本文件名为my_cli_script
则可以

```
mininet> source my_cli_script
```

或者

```
# mn --pre my_cli_script
```

总结篇

除了使用mn命令进行交互式操作以外，mininet最为强大之处是提供api可以直接通过python编程进行灵活的网络实验。在 mininet/example目录下给出了几个python程序的例子，包括使用gui方式创建拓扑、运行多个测试，在节点上运行sshd，创建多个节 点的tree结构网络等等。运行这些程序就可以得到令人信服的结果，而且这些程序大都十分短小，体现了mininet平台的强大易用性。
对mininet感兴趣可以在主页上找到更多的内容。
参考
<1> *A Network in a Laptop : Rapid Prototyping for Software-Defined Networks*, Bob Lantz, Brandon Heller, Nick Mckeown, ACM Hotnets 2010;
<2> http://www.openflowswitch.org/foswiki/bin/view/OpenFlow/Mininet