NFV实验平台的技术方案及搭建过程介绍

报告人：王睿

日期：2019年4月22日

在线版：

内容简介

- NFV背景
- 实验环境简介
- NFV实验平台简介
- 技术方案介绍
- 平台搭建过程及结果

NFV背景

- 网络功能(Network Function, NF)

在源、目的主机之间的报文传输路径上，除路由器、交换机外，任何实现报文处理功能的设备。(RFC3234)

据文献统计[1]，以专用设备形式存在的NF在数据中心中广泛使用，其数量与路由器、交换机相当。

防火墙

VPN

负载均衡

入侵防御系统

1、Making Middleboxes Someone Else’s Proablem: Network Processing as a Cloud Service[C]. SIGCOMM 2012

NFV背景

- 网络功能(Network Function, NF)

在源、目的主机之间的报文传输路径上，除路由器、交换机外，任何实现报文处理功能的设备。(RFC3234)

据文献统计[1]，以专用设备形式存在的NF在数据中心中广泛使用，其数量与路由器、交换机相当。

- NF专用设备存在的问题 [2,3]

随着用户需求日新月异，以及移动设备的爆炸性增长，用户流量呈指数级增长，同时也提出了更多新的网络服务需求。在此背景下，NF专用设备的问题日益凸显，亟待新技术的提出来解决它们。

购买与运维成本高	设备成本：至少数万/台 人员成本：培训、运维	研发与更新周期长	4年左右
灵活性差	位置固定，难以移动	可扩展性差	有研究表明， 20%负载均衡器使用率不超过5%

1、Making Middleboxes Someone Else’ s Proablem: Network Processing as a Cloud Service[C]. SIGCOMM 2012

NFV背景

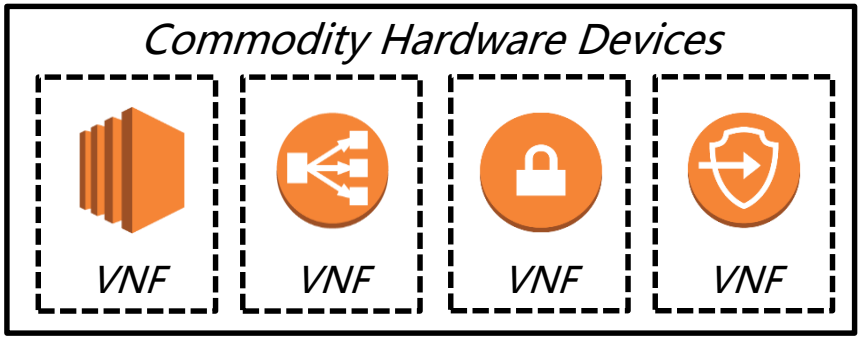

防火墙


负载均衡


VPN


入侵防御系统

虚拟化



低成本

软硬件解耦

高可扩展性

高灵活性

NFV(Network Function Virtualization)：2012年底被提出，它旨在利用虚拟化技术，将网络功能创建成运行在VM或容器之上的、可被连接、通信的虚拟化实例，而这样的虚拟化实例就被称为虚拟网络功能(Virtual Network Function, VNF)。这些VNF需要按照用户指定的顺序执行，以提供完整的端到端网络服务 。

NFV背景

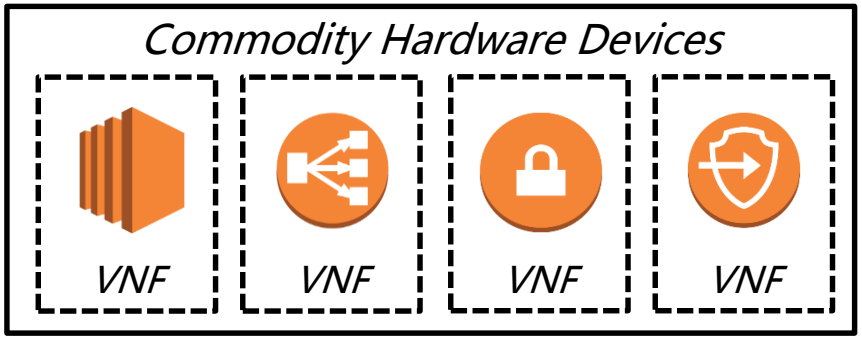

防火墙


负载均衡


VPN


入侵防御系统

虚拟化



用户需求的动态性

VNF管理的复杂性

因此，需要一种机制[1]：
①自动生成与部署VNF有序链
②根据用户请求，动态引导流通过这个链

1、A survey on service function chaining [J]. Journal of Network and Computer Applications 2016.

NFV背景


防火墙


负载均衡


VPN


入侵防御系统

虚拟化

Commodity Hardware Devices

VNF *VNF* *VNF* *VNF*

利用NFV和SFC，在解决专有设备诸多问题的同时，也提供了更为灵活的端到端网络服务解决方案。

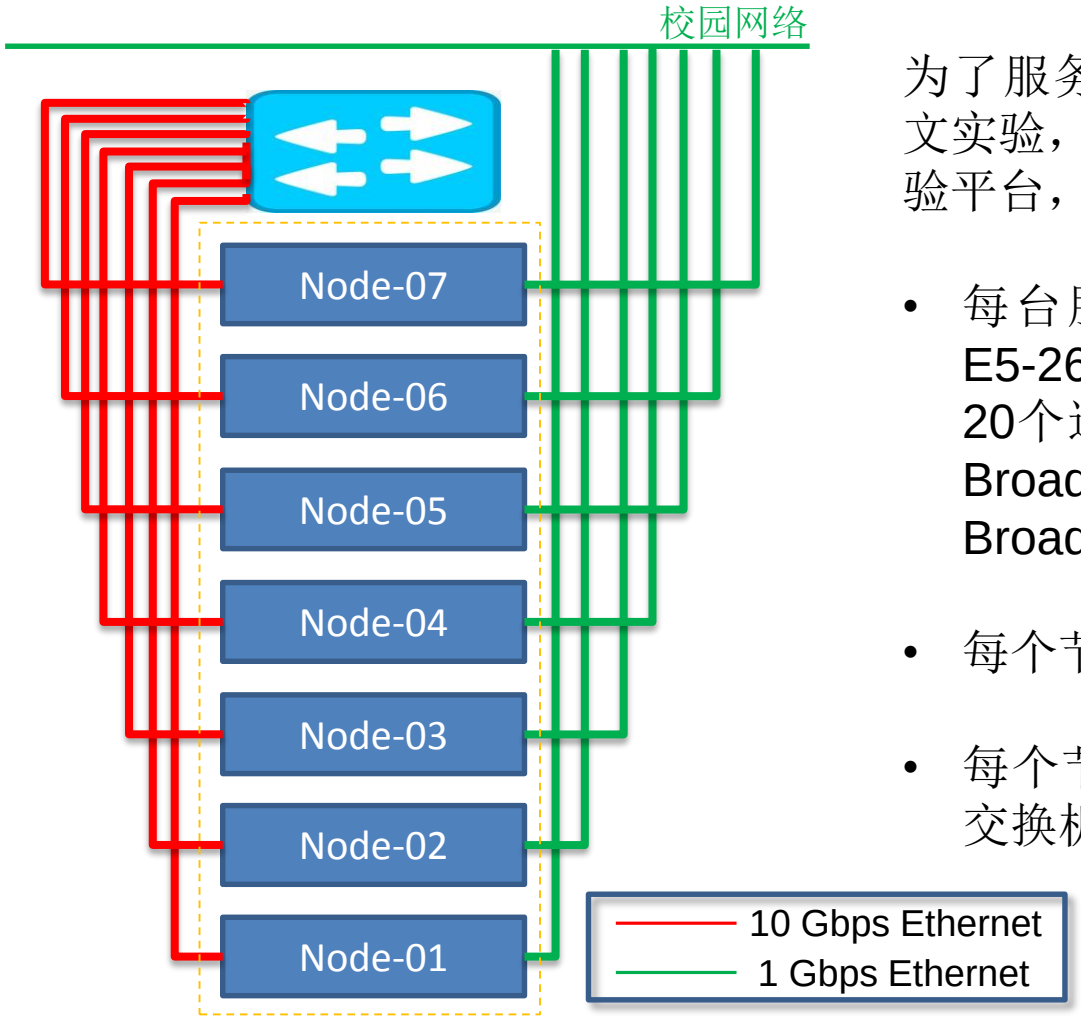
用户需求的

的复杂性

服务功能链(Service Function Chain, SFC)：定义了一组有序或部分有序的网络功能以及根据分类的结果必须应用于相应报文的有序约束。图中蓝线标识出的是一条由4个VNF组成的的SFC。
--RFC 7498

1、A survey on service function chaining [J]. Journal of Network and Computer Applications 2016.

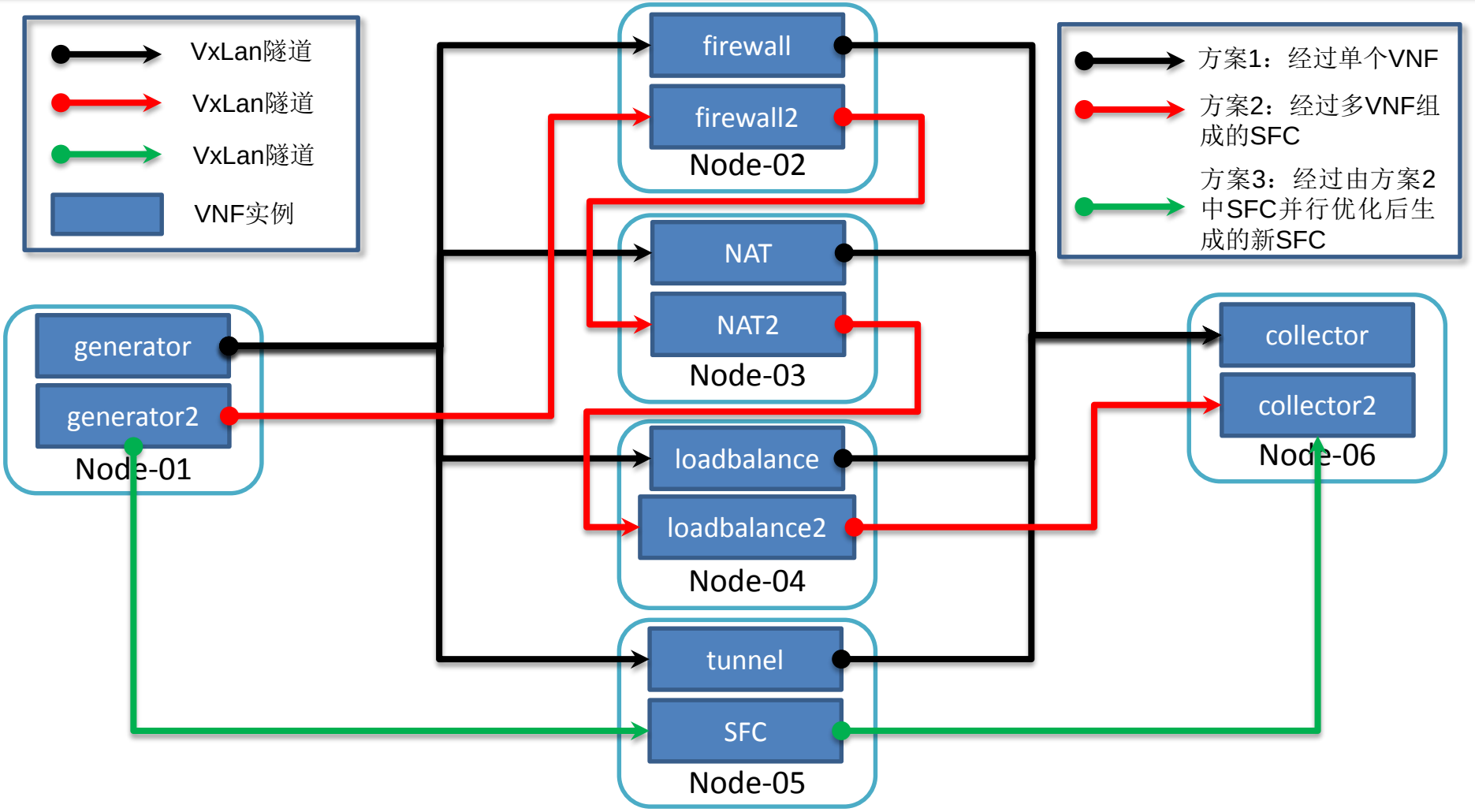
实验环境简介



为了服务VAS网络流分析以及我的毕业论文实验，我在370机房HPE集群上搭建了实验平台，HPE集群的基本情况如左图所示:

- 每台服务器节点均配置2颗Intel Xeon E5-2630 v4 2.2GHz CPU（每颗CPU有20个逻辑核）、64或128GB内存、4块Broadcom BCM5719 1Gbps网卡和2块Broadcom BCM57810 10 Gbps网卡
- 每个节点均通过1 Gbps与校园网络相连
- 每个节点均通过10 Gbps网卡与高性能交换机相连，构建了高速内部局域网

NFV实验平台简介


VxLan隧道
VxLan隧道
VxLan隧道
VNF实例
方案1：经过单个VNF
方案2：经过多VNF组成的SFC
方案3：经过由方案2中SFC并行优化后生成的新SFC
generator
generator2
Node-01
firewall
firewall2
Node-02
NAT
NAT2
Node-03
loadbalance
loadbalance2
Node-04
tunnel
SFC
Node-05
collector
collector2
Node-06

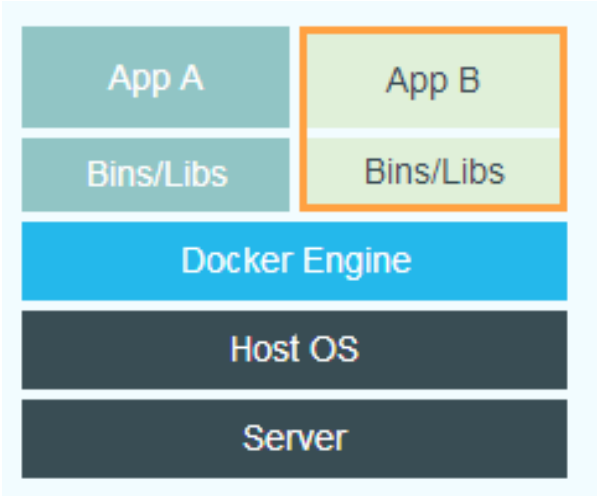
技术方案介绍

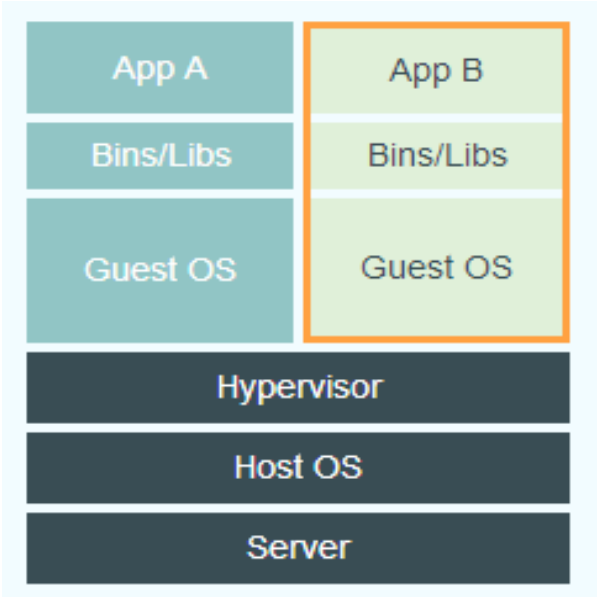
1、VNF实现方案

1.1 Docker容器技术简介

考虑到VNF的可扩展性以及对报文的处理性能等，本实验平台采用基于Click[1]的VNF软件实现方案，并将其运行在docker容器中以部署在服务器节点上。

Docker 是一个基于 Go 语言实现开源项目，其目标是实现轻量级的操作系统虚拟化解决方案。 它利用Linux提供的namespaces机制实现隔离、cgroups机制限制资源、AUFS提供存储服务等。下图展示了docker与传统虚拟机的不同之处。





1、The click modular router [C]. SOSP 1999.

技术方案介绍

1、VNF实现方案

1.2、Click Modular Router简介

Click Modular Router是麻省理工学院的研究人员于1999年研发的模块化路由工具包，时至今日，仍旧在被学术界与产业界的大量研究人员维护、更新与升级。

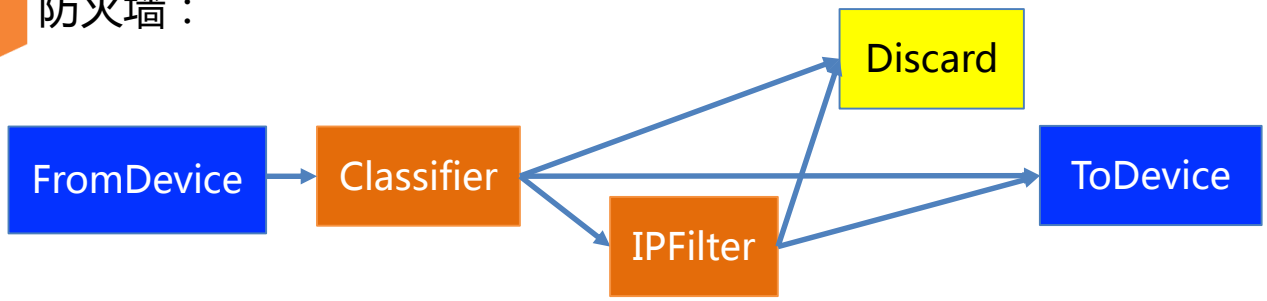
在Click中，每个click网络功能元素（即网元）实现基本报文处理功能，如报文分类、排队、调度、和网络设备交互等。一个虚拟网络功能（VNF）就是一个由click网元组成的有向图无环图（DAG），其中顶点是click网元，而边就是网元之间的连接，报文沿着图的边进行传输。

技术方案介绍

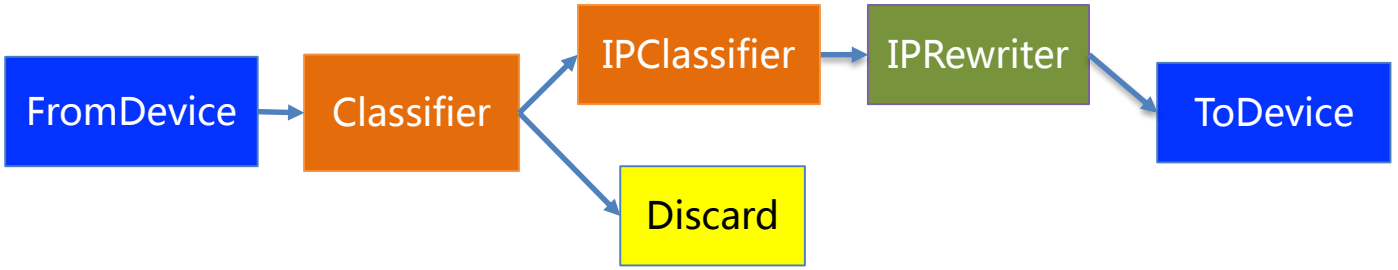
1、VNF实现方案

1.2、Click Modular Router简介

防火墙：



NAT：

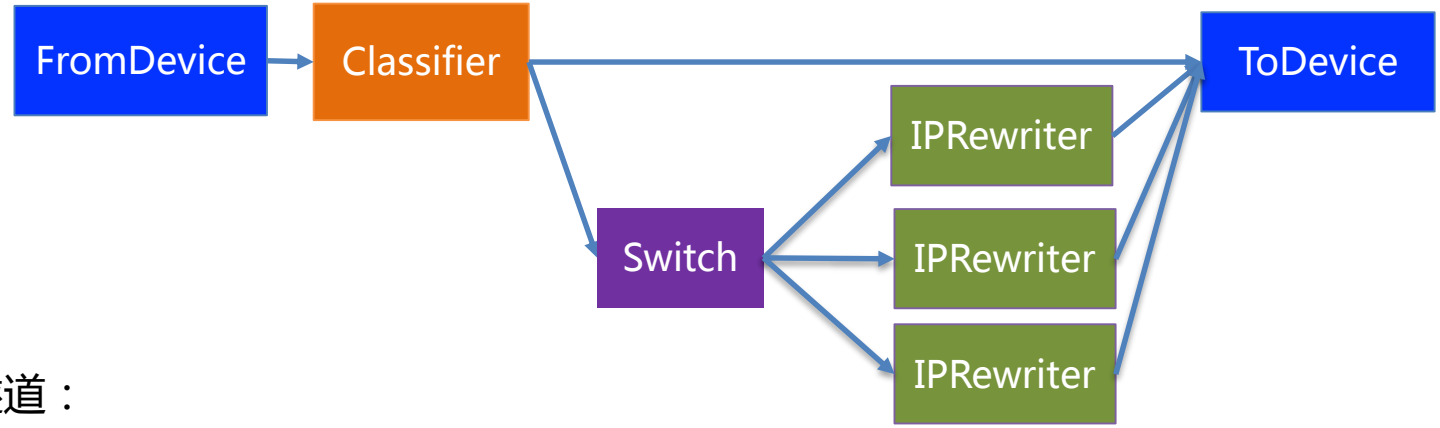


技术方案介绍

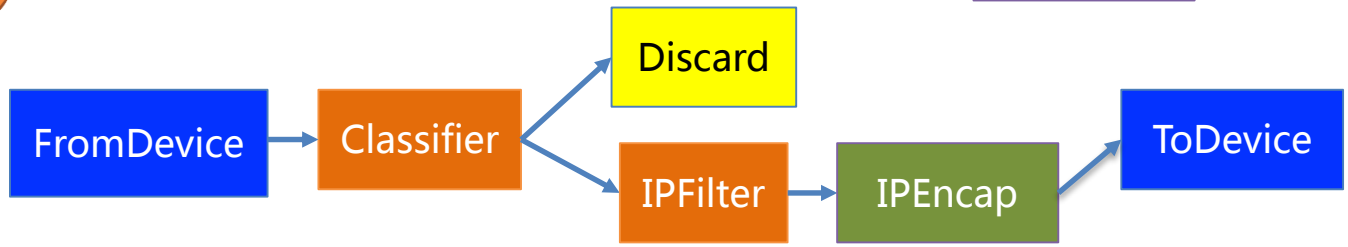
1、VNF实现方案

1.2、Click Modular Router简介

负载均衡：



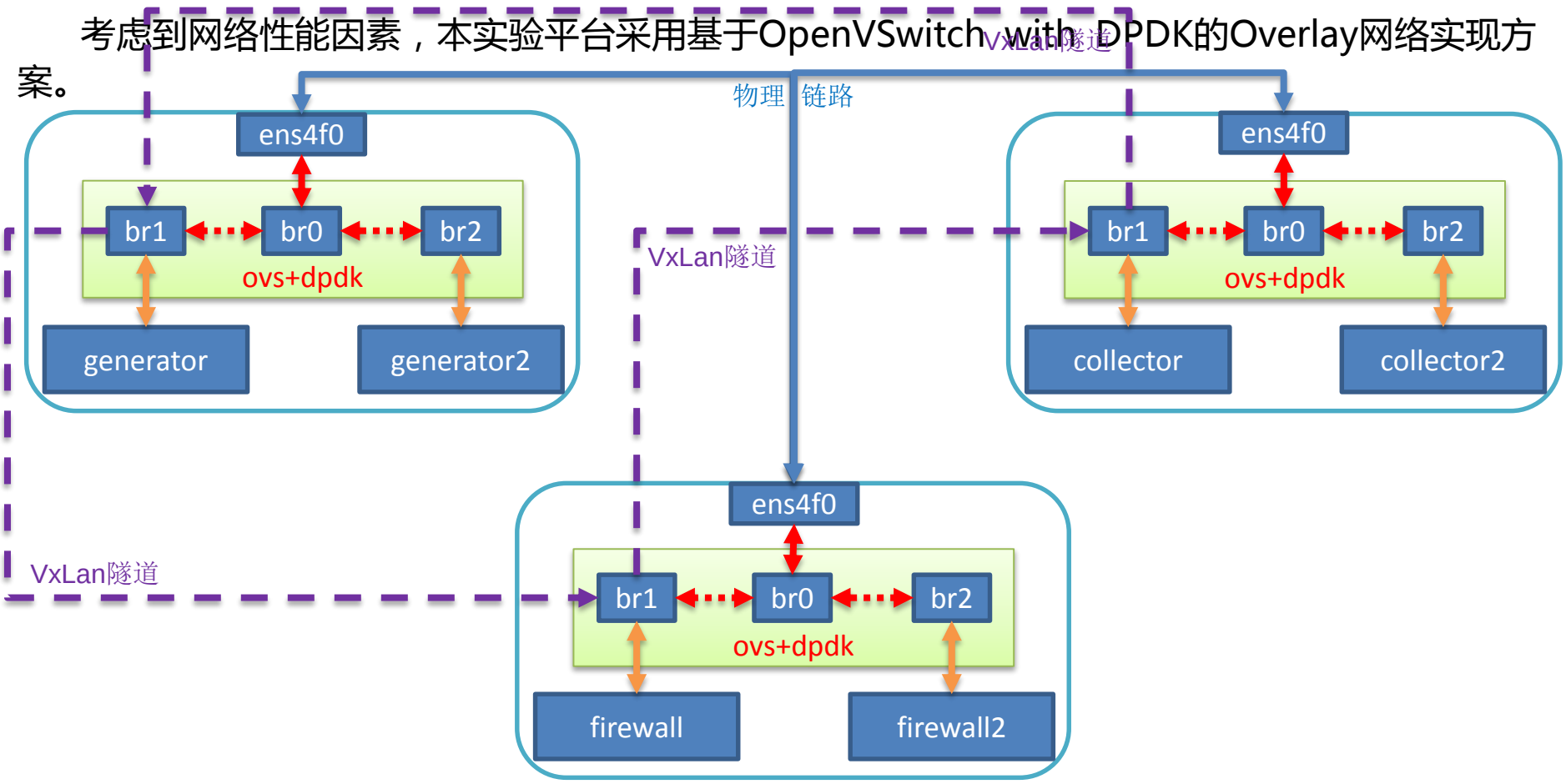
隧道：



实验平台简介

2、Overlay网络实现方案

考虑到网络性能因素，本实验平台采用基于OpenVSwitch with DPDK的Overlay网络实现方案。

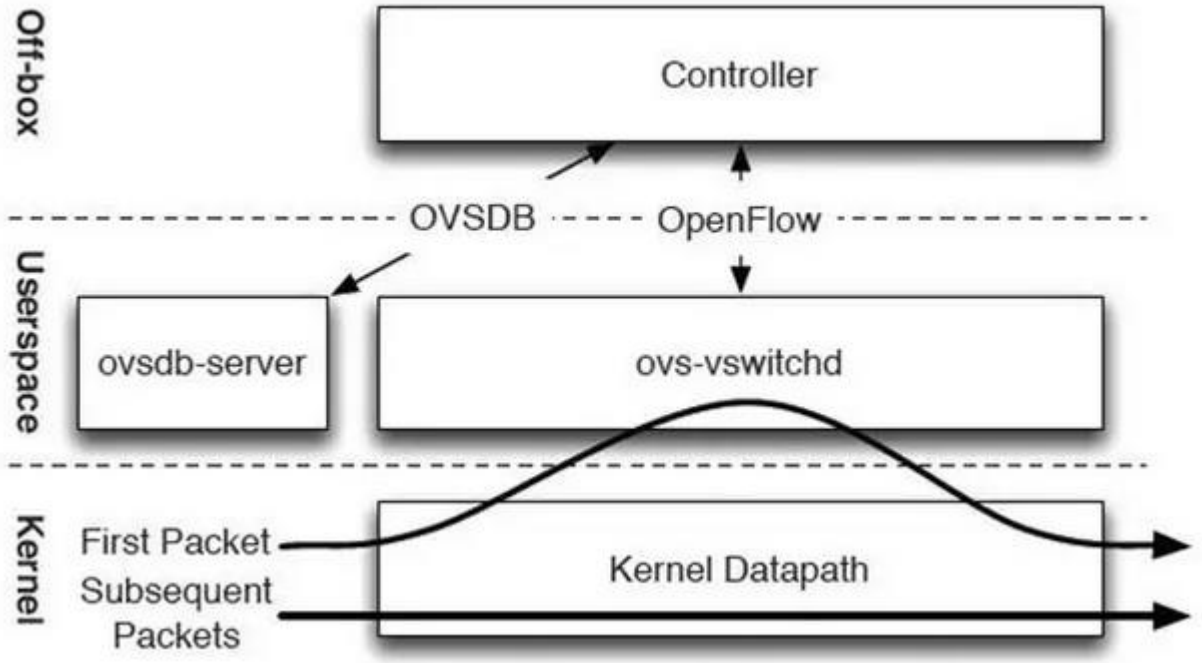


实验平台简介

2、Overlay网络实现方案

2.1、OpenVSwitch简介

OpenVSwitch是一个实现了OpenFlow协议的虚拟交换机，它主要由位于用户空间的ovsdb-server（OVS数据库服务器，存储OVS的配置信息）和ovs-vswitchd（实现交换功能的守护进程），以及位于内核空间的datapath组成。



实验平台简介

2、Overlay网络实现方案

2.1、OpenVSwitch简介

但是，一方面，datapath是利用操作系统内核提供的进程实现转发，操作系统会像对待其他进程一样，将CPU的部分时间片，分配给它，可用内存也会受操作系统的管理，因此，OVS并不能保证在需要进行报文转发时一定占有足够的资源。

另一方面，因为操作系统本身的设计，需要经过硬中断、软中断、内核空间和用户空间的切换等复杂的过程才能完成对报文的获取、处理与转发，极大影响转发的性能，如吞吐量、延迟等。

实验平台简介

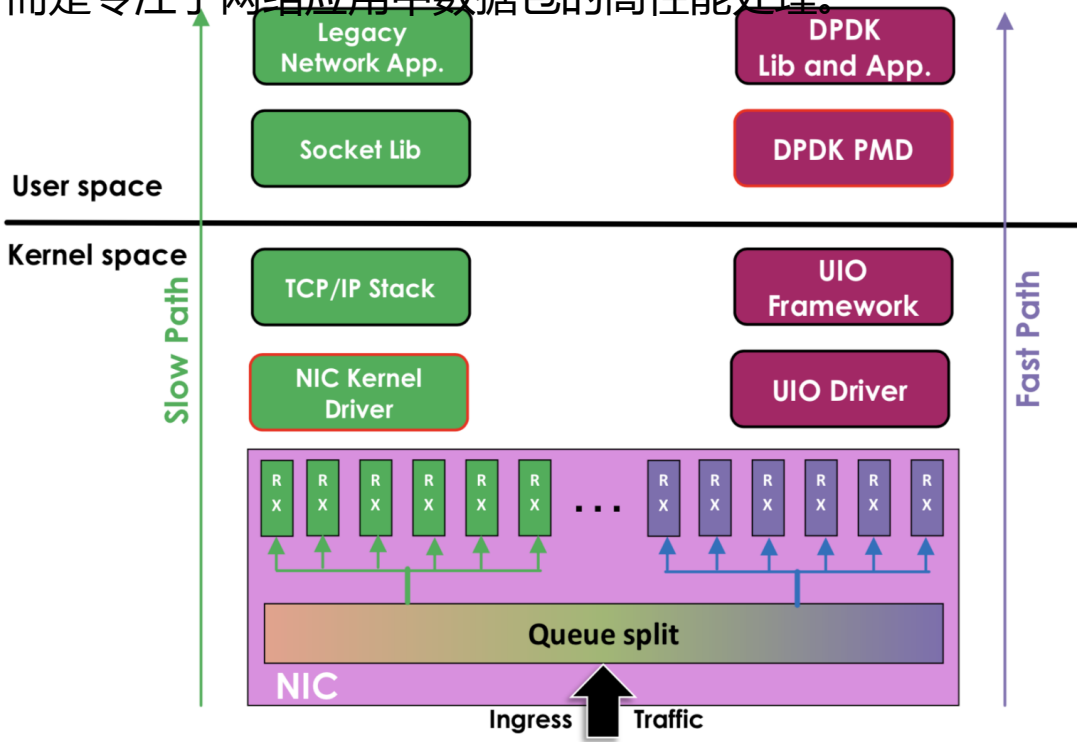
2、Overlay网络实现方案

2.2、NFV的基石--DPDK简介

Intel DPDK全称Intel Data Plane Development Kit，是intel提供的数据平面开发工具集，为Intel architecture（IA）处理器架构下用户空间高效的数据包处理提供库函数和驱动的支持，它不同于Linux系统以通用性设计为目的，而是专注于网络应用中数据包的高性能处理。

- Linux对报文处理过程：
  网卡 -> 内核驱动 -> 内核协议栈 -> 拷贝到用户Socket接口 -> 业务软件

- 基于UIO旁路数据，DPDK对报文处理过程：
  网卡 -> DPDK轮询驱动 -> DPDK基础库 -> 报文映射到业务软件

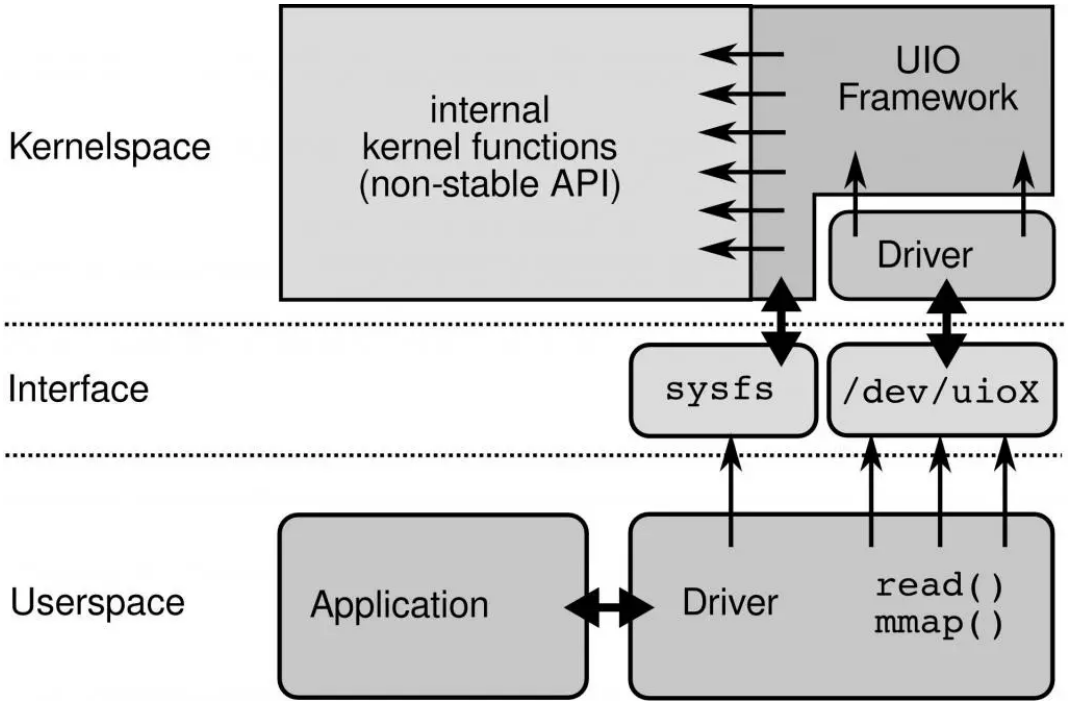


实验平台简介

2、Overlay网络实现方案

2.2、DPDK简介

2.2.1、DPDK旁路实现基础--UIO （Userspace I/O）机制



DPDK能够绕过内核协议栈，本质上是得益于 Linux提供的UIO 机制，它能够拦截中断，重设中断回调行为，从而绕过内核协议栈后续的处理流程并对用户空间暴露文件接口，将对文件的读写映射到对设备的操作。

如左图所示，当注册一个 UIO 设备 uioX，就会出现文件/dev/uioX，对该文件的读写就是对设备内存的读写。

实验平台简介

2、Overlay网络实现方案

2.2、DPDK简介

2.2.2、DPDK核心优化

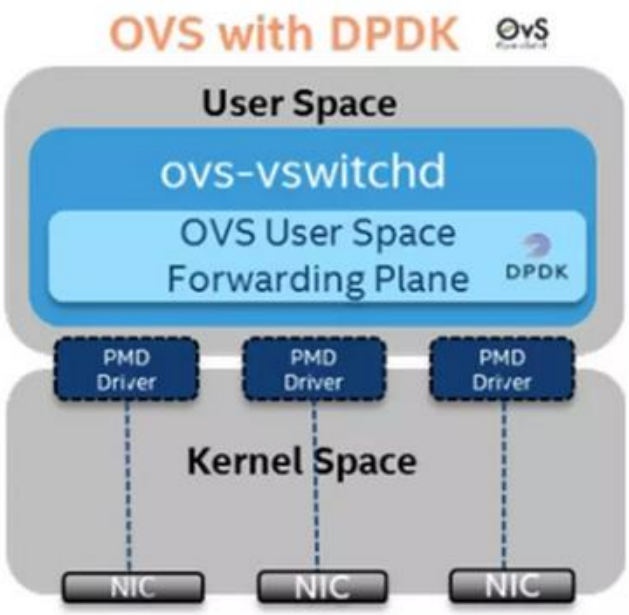
为了优化Linux系统的报文处理性能，DPDK进行了包括但不仅限于如下几方面的核心优化：

1. 采用PMD（Poll Mode Driver）与网卡交互，减少中断开销；
2. 采用HugePage，减少TLB Miss，降低访存开销；
3. 采用精巧的内存池技术，创建Mbuf直接映射到实际报文，内核空间和用户空间的内存交互不进行拷贝，只做控制权转移，避免拷贝开销；
4. 利用CPU 亲和性，将线程绑定到指定CPU上，一方面减少了CPU线程间切换的开销，另一方面避免了 CPU 缓存的局部失效性，增加了 CPU 缓存的命中率；
5. 采用基于Linux 内核的无锁环形缓冲 kfifo优化的无锁环形队列，针对单个或多个数据包生产者、单个数据包消费者的出入队列提供无锁机制，有效减少系统开销。

实验平台简介

2、Overlay网络实现方案

2.3、OVS with DPDK



基于DPDK加速，针对原始OVS存在的问题，一方面，因为PMD采用轮询方式与网卡交互，DPDK要独占部分CPU和内存，这样在任意时间都有足够的资源用于报文的处理、转发，避免了因操作系统调度带来的资源抢占的问题。

另一方面， OVS with DPDK绕过操作系统内核，在用户空间，通过PMD直接操作网卡的接收和发送队列。PMD会不断的轮询网卡，当从网卡上收到报文之后，会直接通过DMA将其传输到预分配的内存中，同时更新接收队列的指针，这样OVS很快能感知收到报文，大大减少了中断、拷贝等开销。

平台搭建过程及结果

1、各节点OVS with DPDK的安装过程

（1）安装DPDK

1. cd /data/wangrui/dpdk-stable-16.11.9
2. export DPDK_BUILD=/data/wangrui/dpdk-stable-16.11.9-installed
3. export DPDK_DIR=/data/wangrui/dpdk-stable-16.11.9
4. sysctl -w vm.nr_hugepages=1024
5. echo 1024 > /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
6. echo 1024 > /sys/devices/system/node/node1/hugepages/hugepages-2048kB/nr_hugepages
7. mkdir /mnt/huge
8. mount -t hugetlbfs nodev /mnt/huge
9. make install T=x86_64-native-linuxapp-gcc DESTDIR=$DPDK_BUILD
10. modprobe vfio-pci

平台搭建过程及结果

1、各节点OVS with DPDK的安装过程

（2）安装OVS with DPDK

```
1.  export OVS_DIR=/usr/src/openvswitch-2.7.7
2.  cd $OVS_DIR
3.  ./boot.sh && ./configure --with-dpdk=$DPDK_BUILD && make install
4.  mkdir -p /usr/local/etc/openvswitch
5.  mkdir -p /usr/local/var/run/openvswitch
6.  rm -f /usr/local/etc/openvswitch/conf.db
7.  ovsdb-tool create /usr/local/etc/openvswitch/conf.db \
8.      vswitchd/vswitch.ovsschema
9.  ovsdb-server --remote=punix:/usr/local/var/run/openvswitch/db.sock \
10.     --remote=db:Open_vSwitch,Open_vSwitch,manager_options \
11.     --private-key=db:Open_vSwitch,SSL,private_key \
12.     --certificate=db:Open_vSwitch,SSL,certificate \
13.     --bootstrap-ca-cert=db:Open_vSwitch,SSL,ca_cert \
14.     --pidfile --detach --log-file
15. ovs-vsctl --no-wait init
16. ovs-vsctl --no-wait set Open_vSwitch . other_config:dpdk-init=true
17. ovs-vswitchd --pidfile --detach
```

平台搭建过程及结果

2、Node-02节点OVS with DPDK配置

（1）br0 配置

1. ifconfig ens4f1 down
2. cd /data/wangrui/dpdk-stable-16.11.9/tools/
3. ./dpdk-devbind.py --bind=vfio-pci 0000:88:00.0
4. ovs-vsctl add-br br0 -- set bridge br0 datapath_type=netdev
5. ovs-vsctl add-port br0 dpdk0 -- set Interface dpdk0 type=dpdk options:dpdk-devargs=0000:88:00.0
6. ifconfig br0 192.168.0.102/24 up

平台搭建过程及结果

2、Node-02节点OVS with DPDK配置

（2）br1 && br2配置

1. ovs-vsctl add-br br1 -- set bridge br1 datapath_type=netdev
2. ovs-vsctl add-port br1 vxlan21 -- set interface vxlan21 type=vxlan options:remote_ip=192.168.0.101
3. ovs-vsctl add-port br1 vxlan26 -- set interface vxlan26 type=vxlan options:remote_ip=192.168.0.106
4. ifconfig br1 192.168.1.2/24 up

5. ovs-vsctl add-br br2 -- set bridge br2 datapath_type=netdev
6. ovs-vsctl add-port br2 vx21 -- set interface vx21 type=vxlan options:remote_ip=192.168.0.101 options:local_ip=192.168.0.102 options:key=124
7. ovs-vsctl add-port br2 vx23 -- set interface vx23 type=vxlan options:remote_ip=192.168.0.103 options:local_ip=192.168.0.102 options:key=124
8. ifconfig br2 192.168.2.2/24 up

平台搭建过程及结果

2、Node-02节点OVS with DPDK配置结果

（1）Hugepages文件系统和DPDK VFIO驱动

```
root@Node-02:/data/wangrui/dpdk-stable-16.11.9/tools# findmnt | grep hugepages
| └─/dev/hugepages                                                                hugetlbfs
            hugetlbfs       rw,relatime
```

```
root@Node-02:~# lsmod | grep vfio
vfio_pci               45056  0
vfio_iommu_type1       24576  0
vfio_virqfd            16384  1 vfio_pci
vfio                   32768  3 vfio_iommu_type1,vfio_pci
irqbypass              16384  2 kvm,vfio_pci
```

平台搭建过程及结果

2、Node-02节点OVS with DPDK配置结果

（2）网卡绑定DPDK VFIO驱动

```
root@Node-02:/data/wangrui/dpdk-stable-16.11.9/tools# ./dpdk-devbind.py --bind vfio-pci 0000:88:00.0
root@Node-02:/data/wangrui/dpdk-stable-16.11.9/tools# ./dpdk-devbind.py --status

Network devices using DPDK-compatible driver
============================================
0000:88:00.0 'NetXtreme II BCM57810 10 Gigabit Ethernet' drv=vfio-pci unused=bnx2x,igb_uio

Network devices using kernel driver
===================================
0000:02:00.0 'NetXtreme BCM5719 Gigabit Ethernet PCIe' if=eno1 drv=tg3 unused=igb_uio,vfio-pci *Active*
0000:02:00.1 'NetXtreme BCM5719 Gigabit Ethernet PCIe' if=eno2 drv=tg3 unused=igb_uio,vfio-pci
0000:02:00.2 'NetXtreme BCM5719 Gigabit Ethernet PCIe' if=eno3 drv=tg3 unused=igb_uio,vfio-pci
0000:02:00.3 'NetXtreme BCM5719 Gigabit Ethernet PCIe' if=eno4 drv=tg3 unused=igb_uio,vfio-pci
0000:88:00.1 'NetXtreme II BCM57810 10 Gigabit Ethernet' if=ens4f1 drv=bnx2x unused=igb_uio,vfio-pci

Other network devices
=====================
<none>

Crypto devices using DPDK-compatible driver
===========================================
<none>

Crypto devices using kernel driver
==================================
<none>

Other crypto devices
====================
<none>
```

平台搭建过程及结果展示

3、Node-02节点VNF实例配置

（1）firewall VNF配置

1. docker run --net=none --privileged=true --name=firewall -itd firewall
2. ovs-docker add-port br1 eth0 firewall --ipaddress=192.168.1.21/24 --gateway=192.168.1.2
3. ovs-docker add-port br1 eth1 firewall --ipaddress=192.168.1.22/24 --gateway=192.168.1.2
4. docker exec --privileged=true firewall iptables -F
5. docker exec --privileged=true firewall iptables -t nat -A PREROUTING -d 192.168.1.61 -i eth0 -j DNAT --to-destination 192.168.1.21
6. docker exec --privileged=true firewall iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
7. docker exec --privileged=true firewall click /home/Click-changed-for-ParaGraph/conf/firewall.click

平台搭建过程及结果展示

3、Node-02节点VNF实例配置

（2）firewall2 VNF配置

1. docker run --net=none --privileged=true --name=firewall2 -itd firewall2
2. ovs-docker add-port br2 eth0 firewall2 --ipaddress=192.168.2.21/24 --gateway=192.168.2.2
3. ovs-docker add-port br2 eth1 firewall2 --ipaddress=192.168.2.22/24 --gateway=192.168.2.2
4. docker exec --privileged=true firewall2 iptables -F
5. docker exec --privileged=true firewall2 iptables -t nat -A PREROUTING -d 192.168.2.61 -i eth0 -j DNAT --to-destination 192.168.2.21
6. docker exec --privileged=true firewall2 iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
7. docker exec --privileged=true firewall2 route add -host 192.168.2.61 gw 192.168.2.31
8. docker exec --privileged=true firewall2 click /home/Click-changed-for-ParaGraph/conf/firewall2.click

平台搭建过程及结果展示

3、Node-02节点VNF实例配置

```
root@Node-02:/data/wangrui/dpdk-stable-16.11.9/tools# docker ps -a
CONTAINER ID        IMAGE                                      COMMAND             CREATED             STATUS              PORTS               NAMES
dcc4e092e8cb        macwr/ubuntu_for_paragraph:latest-0.6      "/bin/bash"         47 hours ago        Up 47 hours                             firewall2
3462b00c38b2        macwr/ubuntu_for_paragraph:latest-0.6      "/bin/bash"         4 days ago          Up 4 days                               firewall
```

```
root@3462b00c38b2:/home/Click-changed-for-ParaGraph/conf# cat firewall.click
//firewall@node02
from@fw::FromDevice(eth0, SNIFFER false)
to@fw::ToDevice(eth1)
todump1@fw::ToDump(/home/in-fw)
todump2@fw::ToDump(/home/out-fw)
c0@fw::Classifier(12/0806,//ARP
                  12/0800,//IPv4
                  -)//others
checkarp@fw::CheckARPHeader
checkip@fw::CheckIPHeader(14)
ipf@fw::IPFilter(allow src 192.168.1.11 && dst 192.168.1.61, deny all)
qnq@fw::QuickNoteQueue

from@fw -> CheckLength(1500)-> todump1@fw -> Print(in) -> c0@fw

c0@fw[0] -> checkarp@fw -> qnq@fw
c0@fw[1] -> checkip@fw -> ipf@fw
c0@fw[2] -> Discard

ipf@fw[0] -> qnq@fw
ipf@fw[1] -> Discard

qnq@fw -> Print(out) -> todump2@fw -> to@fw
```

```
root@dcc4e092e8cb:/home/Click-changed-for-ParaGraph/conf# cat firewall2.click
from@fw::FromDevice(eth0, SNIFFER false)
to@fw::ToDevice(eth1)
todump1@fw::ToDump(/home/in-fw)
todump2@fw::ToDump(/home/out-fw)
c0@fw::Classifier(12/0806,//ARP
                  12/0800,//IPv4
                  -)//others
checkarp@fw::CheckARPHeader
checkip@fw::CheckIPHeader(14)
ipf@fw::IPFilter(allow src 192.168.2.11, deny all)
qnq@fw::QuickNoteQueue

from@fw -> CheckLength(1500)-> todump1@fw -> Print(in) -> c0@fw

c0@fw[0] -> checkarp@fw -> qnq@fw
c0@fw[1] -> checkip@fw -> ipf@fw
c0@fw[2] -> Discard

ipf@fw[0] -> qnq@fw
ipf@fw[1] -> Discard

qnq@fw -> Print(out) -> todump2@fw -> to@fw
```

平台搭建过程及结果

4、方案1各节点VNF路由及iptables配置信息

（1）Node-01节点generator路由信息

```
root@77ef12e2aea7:/# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.1.1     0.0.0.0         UG    0      0        0 eth0
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 eth1
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 eth2
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 eth3
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 eth4
192.168.1.61    192.168.1.51    255.255.255.255 UGH   0      0        0 eth3
192.168.1.61    192.168.1.41    255.255.255.255 UGH   0      0        0 eth2
192.168.1.61    192.168.1.31    255.255.255.255 UGH   0      0        0 eth1
192.168.1.61    192.168.1.21    255.255.255.255 UGH   0      0        0 eth0
```

平台搭建过程及结果

4、方案1各节点VNF路由及iptables配置信息

（2）Node-02节点firewall iptables配置信息

```
root@3462b00c38b2:/home/Click-changed-for-ParaGraph/conf# iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
DNAT       all  --  anywhere             192.168.1.61         to:192.168.1.21

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
MASQUERADE  all  --  192.168.1.11         192.168.1.61
```

平台搭建过程及结果

4、方案2各节点VNF路由及iptables配置信息

（1）Node-01节点generator2路由信息

```
root@b1abf53fa781:/# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.2.1     0.0.0.0         UG    0      0        0 eth0
192.168.2.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
192.168.2.0     0.0.0.0         255.255.255.0   U     0      0        0 eth1
192.168.2.61    192.168.2.51    255.255.255.255 UGH   0      0        0 eth1
192.168.2.61    192.168.2.21    255.255.255.255 UGH   0      0        0 eth0
```

平台搭建过程及结果

4、方案2各节点VNF路由及iptables配置信息

（2）Node-02节点firewall2 路由及iptables配置信息

```
root@dcc4e092e8cb:/home/Click-changed-for-ParaGraph/conf# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.2.2     0.0.0.0         UG    0      0        0 eth0
192.168.2.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
192.168.2.0     0.0.0.0         255.255.255.0   U     0      0        0 eth1
192.168.2.61    192.168.2.31    255.255.255.255 UGH   0      0        0 eth0
root@dcc4e092e8cb:/home/Click-changed-for-ParaGraph/conf# iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
DNAT       all  --  anywhere             192.168.2.61         to:192.168.2.21

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
MASQUERADE  all  --  anywhere             anywhere
```

平台搭建过程及结果

4、方案2各节点VNF路由及iptables配置信息

（3）Node-03节点nat2 路由及iptables配置信息

```
root@e85173741602:/home/Click-changed-for-ParaGraph/conf# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.2.3     0.0.0.0         UG    0      0        0 eth0
192.168.2.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
192.168.2.0     0.0.0.0         255.255.255.0   U     0      0        0 eth1
192.168.2.61    192.168.2.41    255.255.255.255 UGH   0      0        0 eth0
root@e85173741602:/home/Click-changed-for-ParaGraph/conf# iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
DNAT       all  --  anywhere             192.168.2.61         to:192.168.2.31

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
```

平台搭建过程及结果

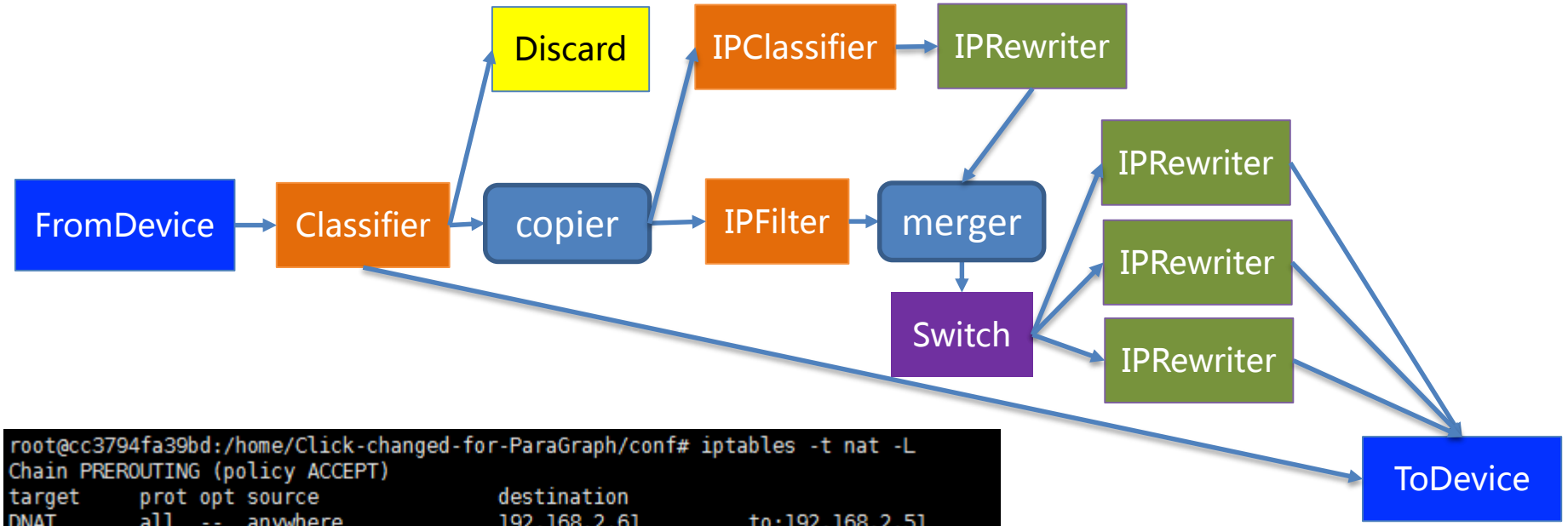
4、方案2各节点VNF路由及iptables配置信息

（4）Node-04节点loadbalance2 路由及iptables配置信息

```
root@6b3e4b8a2522:/home/Click-changed-for-ParaGraph/conf# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.2.4     0.0.0.0         UG    0      0        0 eth0
192.168.2.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
192.168.2.0     0.0.0.0         255.255.255.0   U     0      0        0 eth1
root@6b3e4b8a2522:/home/Click-changed-for-ParaGraph/conf# iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
DNAT       all  --  anywhere             192.168.2.61         to:192.168.2.41

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
MASQUERADE  all  --  anywhere             anywhere
```

平台搭建过程及结果

5、方案3 Node-05 节点SFC实现及其iptables配置信息




```
root@cc3794fa39bd:/home/Click-changed-for-ParaGraph/conf# iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
DNAT       all  --  anywhere             192.168.2.61         to:192.168.2.51

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
```