dpdklvs使用手册

# 基本操作

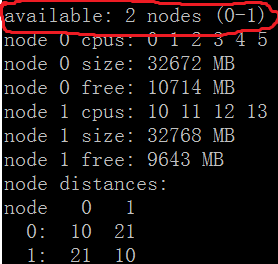
Dpdklvs是用户态lvs，它包含4个部分：

* dlvs: 转发报文的进程。
* dlvsadm: 命令行工具，dlvs配置工具，可以配置和查看dlvs的所有信息。
  + 使用方法、配置项，与标准LVS的ipvsadm基本一致。
* keepalived: lvs的配置管理、健康检查工具。
  + 必须使用dpdklvs的定制版的keepalived
  + 配置方法与标准的keepalived完全一致。
* bgpd: 标准的bgpd，对外发布路由。
  + 使用标准版的bgpd即可。

## 机器环境的配置

* 检查numa，确认numa已经开启

numactl --hardware



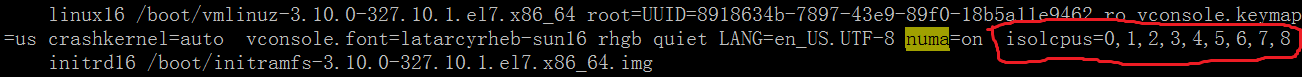
* 检查超线程，确认超线程已关闭。

lscpu | grep "Thread(s) per core"

C:\Users\wangyan50\Documents\JDdongdong\JIMEnterprise\wangyan627\Temp\JdOnline20170217121258.png

* Cpu隔离

cat /proc/grub2/grub.cfg



Isolcpus，隔离cpu core，运行dlvs，可以获取更好的性能。

* 确保标准的ipvs已经卸载
  + 删除标准LVS的ipvsadm命令,避免误操作

rpm -qa | grep ipvsadm | xargs rpm –e

* linux网络环境配置与检查

检查业务网口在linux环境下，与网关、后端服务器的连通情况。

* + 与标准的lvs不同，部署dpdklvs的机器需要增加一个网口，用于ssh连接。
  + 假设机器有eno1\eno2两个网口。ssh登录，网口eno1；其它所有流量，包括Lvs、keepalived、bgp的报文，经过网口eno2。为了达到这个目的，需要配置策略路由。

Example:

ip route add default via 172.20.18.142 dev eno2 table 100//策略路由

ip rule add from 172.20.18.128/28 table 100 //策略路由

ip route add default via 172.20.18.63 dev eno1 //机器的默认路由

在centos7上策略路由，可以参考<http://git.jd.care/wangyan/dpdklvs-rpm>里面的80-ruleroute文件.

* + - 编辑80-ruleroute修改好网卡名和IP地址，
    - chmod +x 80-ruleroute
    - cp 80-ruleroute /etc/NetworkManager/dispatcher.d/
    - service network restart
    - 查看策略路由的情况：ip rule; ip route show table 100

## Dpdklvs的安装与卸载

Dpdklvs相关的安装包的路径：

<http://git.jd.care/wangyan/dpdklvs-rpm>

卸载：

* 删除旧的dpdklvs

rpm –e dlvs-7.2-1.0-1.x86\_64

* 删除keepalived，dpdklvs需要定制的keepalived，标准的不可用

rpm -qa | grep keepalived | xargs rpm –e

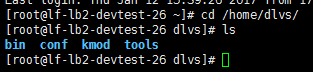
安装：

* 安装dlvs

rpm –ivh dlvs-7.2-1.0-1.x86\_64.rpm

备注：每个组合(机型+os)，需要编译一份rpm安装包。或者直接在目标机器上编译。

安装在/export/servers/dlvs/目录：



* 安装dpdklvs定制的keepalived

rpm –ivh keepalived-1.2.2-5.x86\_64.rpm

## dlvs的配置

dpdklvs启动后，原来的网络接口在linux不存在了，会生成一个vEth0的interface。

网络配置，比如路由、ip地址、vlan，都是在这个vEth0操作，

bgp、ospf、健康检查的报文，都是通过这个vEth0收发。

下面重点描述dpdklvs与标准LVS不同之处。

### 配置策略路由

编辑并执行/etc/NetworkManager/dispatcher.d/80-ruleroute

### 配置lo、bgp、keepalived

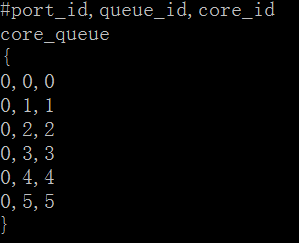
配置方法与标准的LVS一致。

### 3.3配置dlvs

* 配置文件

需要定义3类配置文件，分别配置网络、工作的cpu

* + 配置1：dpdk\_core.conf



核的选取，优先选择与网卡同numa。

./cpu\_layout.py 可以查看core与numa的关系

./sys/class/net/$eth\_dev /device/numa\_node， 可以查看网卡与numa的关系。

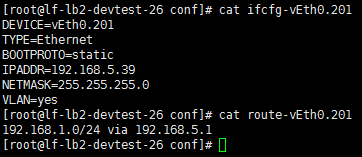
* + 配置2：网卡和路由配置文件。

比如，ifcfg-vEth0 route-vEth0.201等。

分两部分，路由和interface地址配置。

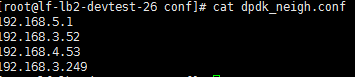
采用与linux一样的配置文件，只需要把linux下相关网卡和路由的配置文件，拷到/export/servers/dlvs/conf目录中，修改文件名字、网卡名字即可。

网卡的名字必须是vEth0。



* + 配置3：配置文件dpdk\_neigh.conf，2层邻居的ip地址。

把网关的ip、dr模式下realserver的ip，填写在这个文件中。



## 启动、停止服务

### 启动服务

进入/export/servers/dlvs/tools操作。

可以通过一个命令：

* + ./dlvs\_start.sh $eth\_name

也可以逐步执行。

* + 设置dpdk运行环境。
    - ./0\_dlvs\_init.sh $eth\_name

打印” 0\_dlvs\_init.sh ok!”，表明这一步成功。

* + 启动dlvs程序
    - python 1\_dlvs\_run.py

ifconfig –a , 能看见网卡vEth0，表明这一步成功。

* + - 这一步如果出现错误，一般是因为配置给dpdk的内存不足导致。可以调整下面的值，增加NODENAME\_A分配给dpdk的内存。

C:\Users\wangyan50\Documents\JDdongdong\JIMEnterprise\wangyan627\Temp\JdOnline20170314183154.png

* + 完成配置网络
    - 运行python 2\_dlvs\_if.py

打印信息，” 2\_dlvs\_if.py ok!”,表明成功。

* + 完成配置arp
    - 运行python 3\_dlvs\_arp.py

打印信息“3\_dlvs\_arp.py ok”，表明成功。

* + 完成配置Lvs的业务逻辑
    - 运行 service keepalived restart
  + 启动对网关的arp的检查，监控网关mac地址的变化。
    - 运行 ./check\_gw\_dpdklvs.sh 实现。

### 停止服务

./dlvs\_stop.sh

## 命令行工具dlvsadm

标准的LVS提供了命令ipvsadm，用于查看LVS的运行状态。

dpdklvs也开发了对应的工具dlvsadm，功能与ipvsadm基本一样，参数也一致，

具体的用法可以参考标准LVS的ipvsadm的手册。

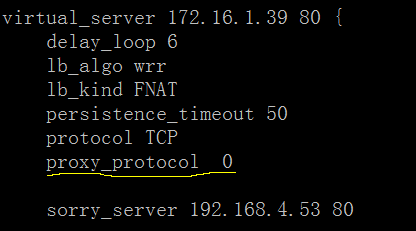
此外，dpdklvs还有其它功能，下面详细描述。

### 配置lvs服务

* Vip , Realserver , Dr和fullnat

目前dpdklvs配置lvs的逻辑时，统一使用keepalived来配置。

* + 相对于标准的keepalived，增加了proxy\_protocol的配置，0关闭， 1开启， 默认关闭。配置在virtual\_server级别。如果后端服务器需要获取真实的client ip，需要开启。



* 动态增加配置

可以通过dlvsadm动态添加配置，即时生效。

* Lvs配置察看

dlvsadm -Sn > dpdk\_lvs.conf

### 配置协议栈

标准的lvs通过 linux的/proc目录来配置，

Dpdk版本的, 通过dlvsadm命令来配置

dlvsadm --setproc timeout\_established 90

dlvsadm --setproc timeout\_synsent 3

dlvsadm --setproc timeout\_synrecv 3

dlvsadm –setproc timeout\_finwait 7

dlvsadm –setproc timeout\_timewait 7

dlvsadm --setproc timeout\_close 3

通过dlvsadm –getproc 来查看系统中的配置。

### 配置dpdk的路由和arp

dlvsadm --setroute <net/masklen> <gw> <dev>

dlvsadm --setarp <ip> <mac> //手动配置arp

dlvsadm --queryarp <ip> //查询并配置arp

dlvsadm --showarp [ip] //获取（某个ip对应的）arp

dlvsadm --showroute //获取路由

## 监控相关

* 进程监控
  + 包括察看dlvs、bgp、keepalived进程是否存在。
* CPU利用率

按照linux的perf工具，dpdk利用率一直是100%，就算不收发报文时依然100%。

dlvsadm –L –resource，查看dlvs的cpu利用率、lvs收发报文速率；

* 内存监控

Dpdk启动时，会从系统中分配32G内存，然后自己管理。Free无法准确看出dpdk管理的内存使用率。

dlvsadm – –mempool，查看dlvs的内存的使用情况。如果某一项的

* 报文监控

可以通过:

dlvsadm –L –resource，lvs收发报文速率；

dlvsadm --ethtool，查看网卡收发丢弃报文、丢弃的报文累计值和速率、网卡各个队列的值；

dlvsadm --netfilter，查看dlvs收发、丢弃的报文的累计值

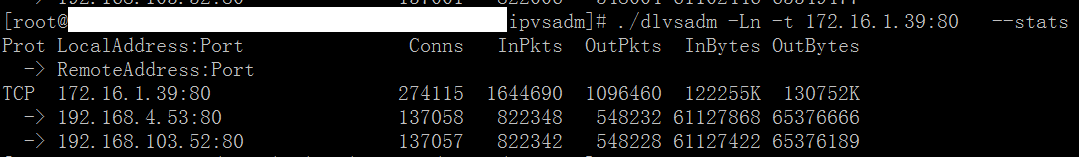
* 日志监控
  + /export/servers/dlvs/log/dlvs.log中，会记录程序运行的信息。遇到比较严重的问题，会添加上”DPDKLVS\_ERR”标识，并打印出来。
  + Keepalived 的日志记录(/var/log/messages)中，有realserver 增加、删除的记录。
* 事件监控

业务口的vlan子接口(VEth0.xx),存在且”state UP”

* 服务监控

通过命令：dlvsadm -Ln -t 172.16.1.39:80 –stats 查看各个服务的收发的累积值。

* + 如果该服务的Conns、InPkts 和 OutPkts没有变化，可以认为整个服务有问题；
  + 如果某个后端服务器的相关值有异常，可以认为该服务器出现异常

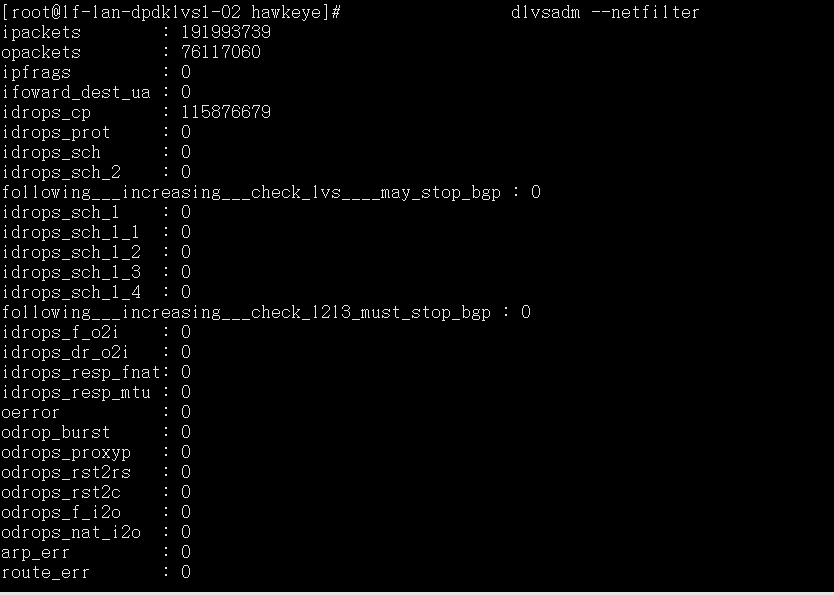


注意：与标准lvs不同之处，无法提供：dlvsadm -Ln -t 172.16.1.39:80 –rate

* dlvsadm –netfilter

这个监控，从第6项开始的统计都为0，表明dlvs进程转发没有问题；

如果不为0，表明可能有错误，需要结合其它的指标，看是否有异常出现。



各个计数，是系统启动至今的累计值，的意义如下

|  |  |
| --- | --- |
| 这一部分，无需报警 | |
| Ipackets | 接收的报文 |
| Opackets | 发送的报文 |
| Ipfrags | 接收的Frag报文 |
| ifoward\_dest\_ua | 发送的报文，dest状态不对 |
| idrops\_cp | 接收了报文，在当前配置下，没有规则可以转发这个报文 |
| idrops\_prot | 来自客户端，非tcp报文，丢弃 |
| idrops\_sch | 来自客户端，虚拟服务存在，无法调度到后端 |
| idrops\_sch\_2 | 来自客户端，vip匹配、vport不匹配，丢弃 |
| 下面的非零，报警，需要确认是否有问题 | |
| idrops\_sch\_1 | 来自客户端，虚拟服务存在，无法创建连接 |
| idrops\_sch\_1\_1 | 虚拟服务的port为0、非persistent，丢弃报文 |
| idrops\_sch\_1\_2 | 来自客户端，虚拟服务存在，找不到后端服务器 |
| idrops\_sch\_1\_3 | 来自客户端，虚拟服务存在，没有内存connection memory |
| idrops\_sch\_1\_4 | 来自客户端，虚拟服务存在，local ip+port不足 |
| 下面的非零，出问题了，需要关bgp切走流量 | |
| idrops\_f\_o2i | 来自客户端，fullnat，外面到里面，3层发送失败 |
| idrops\_dr\_o2i | 来自客户端，dr，外面到里面，3层发送失败(路由\arp不存在) |
| idrops\_resp\_fnat | 来自后端，fullnat，修改报文失败 |
| idrops\_resp\_mtu | 来自后端，报文超过mtu(1500)，丢弃 |
| Oerror | 3层发送失败 |
| odrop\_burst | 2层发送失败，链路忙或者出问题了 |
| odrops\_proxyp | 发送proxy protocol报文失败 |
| odrops\_rst2rs | 向后端发rst，3层发送失败 |
| odrops\_rst2c | 向client发送rst，3层发送失败 |
| odrops\_f\_i2o | 来自后端，fullnat，3层发送失败(路由、arp不存在) |
| odrops\_nat\_i2o | 来自后端，nat，3层发送失败(路由、arp不存在) |
| arp\_err | 3层发送失败，找不到arp |
| route\_err | 3层发送失败，找不到路由 |

* 连接监控

// (参数的使用见7.2节)

dlvsadm  -Lnc --vip 172.16.1.39 --vport 80 --cip 192.168.1.50 --lip 172.16.1.133  --dip 192.168.4.53 --dport 80 --state TIME\_WAIT

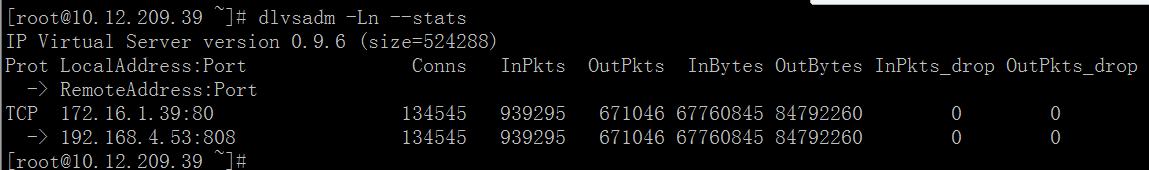
dlvsadm –Lnc | grep “172.16.1.39” | grep “EST” | wc –l

* 转发监控

相对于标准的lvs，增加了丢包统计，统计维度为vs级别、后端服务器级别、session级别。

* + vs级别、后端服务器级别，转发的成功、失败次数。

dlvs –Ln --stats



InPkts : client发进来的报文

OutPkts : 后端发出去的报文

InPkts\_drop ： client进来的报文被dlvs丢掉

OutPkts\_drop : 后端发出去的报文被dlvs丢掉

* + 连接级别、报文收发情况的详细信息。dlvs –Lnc 某条连接的输出：

C:\Users\WANGYA~1\AppData\Local\Temp\1490005575(1).png

i 客户端进来的报文

o 后端发出去的报文

id 客户端进来的报文，被dlvs丢弃

od 后端发出去的报文，被dlvs丢弃

* 业务日志

在日志级别是pre时，每个报文输出3条日志, dlvs每秒打印当前的时间。在系统不繁忙时，可以使用这个功能调试问题。

(dlvsadm –setproc dlvs\_set\_log\_level pre)

## 调试

### 抓包工具

dpdk-pdump可以抓取dpdk收发的所有报文，可以针对每个queue来抓包。

./tools/dcap.sh start 开始抓包

./tools/dcap.sh stop 停止抓包

抓网卡所收发的所有报文，存储在/tmp/tx.pcap 和 /tmp/rx.pcap文件，收和发存在不同的文件。

可以根据需求调整dcap.sh，增加过滤参数。

--pdump'port=0,queue=\*,sip=1.1.1.1.dip=3.3.3.3,sport=101,dport=103,rx-dev=/tmp/rx.pcap'

// queue=\*,表明所有的队列，

// queue=3,表明3号队列，

// sip/dip/sport/dport,表明tcp的源目ip和源目port。

// tx-dev= 表明发送的报文

// rx-dev= 表明接收的报文

// /tmp/rx.pcap,文件名和内容可以修改

### 系统当前连接展示

dlvs –Lnc

dlvsadm  -Lnc --vip 172.16.1.39 --vport 80 --cip 192.168.1.50 --lip 172.16.1.133  --dip 192.168.4.53 --dport 80 --state TIME\_WAIT

可以使用的过滤项项如下：

vip/vport

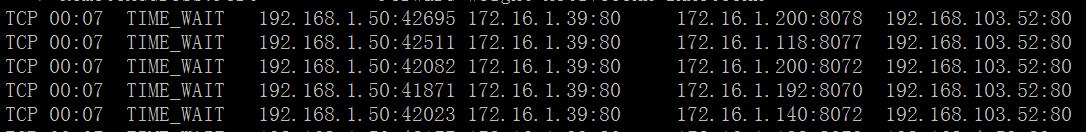
cip/cport //client ip/port

lip/lport //local ip /port

dip/dport //后端服务器ip/port

state //连接的状态

State可以选择：“ESTABLISHED SYN\_SENT SYN\_RECV FIN\_WAIT TIME\_WAIT CLOSE CLOSE\_WAIT LAST\_ACK LISTEN SYNACK”



### 日志

Dpdklvs运行时的日志，存放在dlvs.log文件，路径在/export/servers/dlvs/log/目录。

日志开关：

dlvsadm --setproc dlvs\_set\_log\_onoff 1 // 1,on ; 0,off， 默认是打开的。

日志级别

dlvsadm --setproc dlvs\_set\_log\_level debug // debug/pre/info/err,

下面的顺序，记录信息由多到少，info为线上的默认级别。

* + debug级别，输出详细的处理流程，仅用于线下调试用。
  + pre级别，在负载很小时，可用于线上debug，每个报文大概有3条日志，另外，每秒系统输出时间。
  + info 是默认级别，除了错误信息，还记录vs、dest、lip、route、arp的增减信息。
  + Err级别，只输出严重错误。

### misc，查看状态

dlvsadm --ethtool

查看网卡的收发包统计。

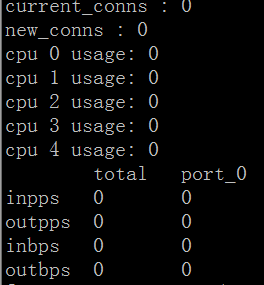
dlvsadm –mempool

查看dpdk的内存使用情况。

Dlvsadm –netfilter

查看dpdk的lvs收发包情况。//进一步细化，需要统计出每个异常点

dlvsadm –L –resource //获取cpu利用率、连接数、网卡等信息



### 排查步骤

部署和运维时，排查步骤：

* 第一步，查看进程是否启动成功。

ps axu | grep dlvs

如果不存在dlvs进程，表明启动失败。

一般是内存设置的问题，或者conf/dpdk\_core.conf设置错误。

* 第二步：检查本机的静态路由

如果能从dlvs机器上，ping 和 curl 后端的server，表明本机的静态路由正确。

否则需要检查静态路由的设置。

相关的文件为conf/route\* / 和 /etc/NetworkManager/dispatcher.d/80-ruleroute

如果调整后，问题依旧，运行./dlvs\_stop.sh，此时网卡由linux驱动，在linux环境下，排查网络是否正常。

另外这一步不成功，很可能是系统没有按照vconfig导致，yum install vconfig之后，再重新启动dlvs即可。

* 第三步：检查本机的动态路由情况

如果从client能ping 通vip，表明vip发布成功。

如果从后端server能ping 通local ip，表明local ip发布成功。

如果失败，则需要调整bgp的配置；如果依然失败，则运行./dlvs\_stop.sh，在linux环境下排查网络问题。

* 第四步：检查本机的lvs逻辑

通过dlvsadm –Ln 查看本机lvs的逻辑，确保keepalived的配置生效。

如果与预期不一致，需要检查dlvs机器与后端服务器的连通性，确认后端服务器是否up。

* 第五步：检查服务是否正常

从客户端 curl http：//vip:port，如果不通，表明服务有故障，

接下来，从dlvs本机上curl http：//real\_server\_ip:port/

结果分两种情况：

* + - 如果不通，需要检查realserver；
    - 如果通，需要检查client与dlvs间的连通情况，
      * 可以通过./dcap.sh start抓包排查
      * 命令dlvsadm -Ln -t 172.16.1.39:80 –stats
      * 命令dlvsadm –Lnc | grep 172.16.1.39，查看连接的状态
      * 通过dlvsadm –mempool查看dpdk的内存是否不足
      * 通过dlvsadm –L –resource查看是否cpu不足
      * 通过dlvsadm –-netfilter，察看丢包的细节
      * 查看log/dlvs.log是否有ERR\_DLVS信息

# dpdklvs架构图



# dpdklvs的运维步骤

## 机器环境的配置。

## 卸载标准的keepalived，安装dpdk版本的keepalived，同时配置lvs的逻辑，配置方式与标准的lvs一致。

## 配置bgp，发布vip; fullnat时，还需要发布local ip; bgp用标准的版本。

## 安装dpdklvs，配置网络、core等信息。细节参考本文档第一章。

## 启动dpdklvs。进入/export/servers/dlvs/tools/做相关操作。细节见本文档第一章。

## 运维与监控

通过dlvsadm查看dlvs运行的状态

通过/export/servers/dlvs/log/dlvs.log，查看运行时的日志、错误。

具体参考第一章第6节。

# Dpdk相关知识

## dpdk的模块图(dpdklvs的应用)



实线方框，表明dpdklvs运行时的各个组件；

虚线方框，表明linux驱动下，网卡相关的组件，dpdklvs运行时，虚线部分就不存在了。

dlvs的逻辑伪码如下：

dlvs\_process() {

while (1) {

read packets from nic;

vip+port匹配的报文，修改并发送；

其它报文，发送到kni\_fifo;

read packets from kni\_fifo，并发送出去;

}

}

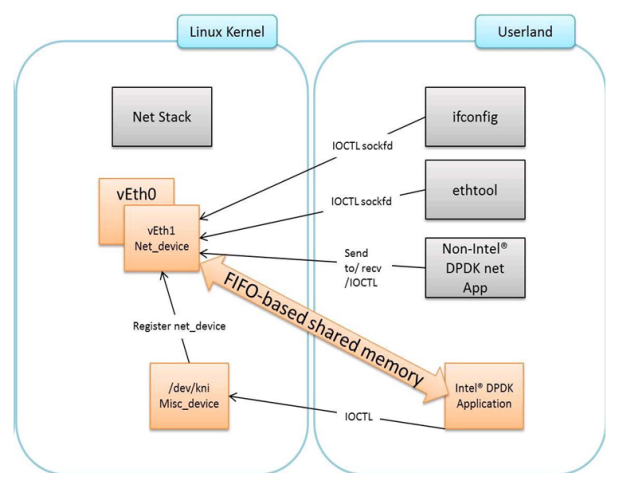
### Dpdk的原理

* Dpdk，是一种运行在用户态的网卡驱动程序。通过dpdk，应用程序可以在用户态，直接读写网卡的寄存器，收发报文
* 在常规的系统中，网卡是通过linux驱动程序来管理的，见上图的虚线框eth0和ixgbe.ko。
* 为了让dpdk接管网卡驱动，必须通过dpdk的工具，来解除ixgbe.ko和网卡直接的关联，这样interface eth0就不存在了。(Dpdk接管网卡之后，会生成vEth0这个interface)
* Dpdk包含两部分：用户态api和内核态的模块(uio.ko、ixgbe\_uio.ko、rte\_kni.ko)。在安装dpdk的应用时，需要先加载内核模块，之后用户态api才能正常工作。
  + uio.ko和ixgbe\_uio.ko，在这两个内核模块的支持下，可以才能在用户态收发报文。
  + 应用程序(以dlvs为例)，能够通过dpdk的api，接收网卡中的所有报文。

对于vip+port匹配的报文，则经过lvs逻辑修改报文后，再通过dpdk的api发送到网络中去。

* + 加载内核模块rte\_kni.ko之后，才能在用户态生成网卡vEth0。
  + 基于rte\_kni.ko，可以创建一个双通道的fifo，用于用户和内核协议栈通信。
    - dlvs从网卡收到报文后，把不感兴趣的报文(vip+port不匹配)，通过这个fifo,递交给linux内核协议栈；
    - 内核协议栈需要通过这个网卡向外发送报文时，就把报文填入这个fifo中，然后dlvs从这个fifo中读取报文，通过uio发送到网络中去。

### Dpdk与linux的关系



### dpdklvs的方案

* 报文的流程

获取网卡的所有报文，截取vip+port匹配的报文，转发出去；

其它的所有报文，都递交给linux协议栈处理。

* vip+port匹配的报文，转发时的arp和route表

在应用程序dlvs转发Vip+port匹配的报文时，需要有arp\路由表.但是linux的这些表在内核中，用户态无法直接使用。Dlvs采取的方案是，dlvs从配置文件中读取配置，自己维护arp表和路由表。

* 其他的报文

其它的报文，递交给linux的协议栈，与标准的lvs的处理流程一致，只是网卡名字换成了vEth0。

* 网络的配置

包含两部分，linux下的配置、dlvs的配置。

* + Linux下的配置，与标准LVS一样，只是网卡名换成了vEth0。包括vEth0的ip地址、vlan\路由\策略路由的配置。
  + dlvs的配置，包括vip+port匹配报文转发需要的arp和route。

### dpdk引入的问题

* 问题1：dpdk启动时，dpdk网卡无法收发报文且时间长。

dpdk启动的时间较长，1分钟左右，在这一分钟，这个网卡无法收发报文。

如果服务器只有一个网卡，启动的过程中，服务器无法登陆；如果由于配置错误，dpdk程序启动失败，服务器就失联了。

* + 为了解决这个问题，需要使用另外的一个网卡，用于ssh登陆服务器，在服务器上启动dpdk、配置dpdk。
  + 新加的网卡，只用于ssh登陆，其他的所有流量，都和标准的lvs一样，走原来的网卡。
* 问题2：加入新网卡引入了路由问题

既然增加了一个网卡，又引入了一个新问题，两个网卡，默认路由从哪个网卡走？

* + 采取的方案时，添加一条源ip策略路由，如果报文的源ip是新增网卡的ip，就从新网卡发送出去；其他的报文，路由不变。
* 问题3：常规的网络抓包、统计工具无法使用

由于网卡收到的报文，有一部分(vip+port匹配)，被dlvs通过dpdk直接转发出去了， linux协议栈不知道这些报文存在过，所以tcpdump\sar\ifconfig\netstat\ss等工具，都统计不到这些报文。

* + 采取的方案，dlvsadm增加一些统计信息；dcap.sh从dlvs中抓取报文。
* 问题4：cpu利用率统计

dpdk线程，独占cpu，while循环不sleep，通过linux的工具，cpu利用率永远为100%，无法统计出真实负载。

* + 统计出处理报文的cpu周期，剖出空转的cpu周期，算出真实的cpu利用率。
* 问题5：内存统计

Dpdk程序，在启动时预分配内存，运行结束才释放这些内存。通过free查看时，无论负载如何，dpdk占用的内存都是一致的，无法查看出内存的真实使用率。

* + Dpdk应用的关键数据，通过内存池来使用内存。通过工具dlvsadm获取各个内存池的使用情况，获取出内存的真实使用率。

## dlvs数据流

### dlvs进程收发的报文

* dlvs从网卡收到的报文



* linux内核通过vEth0发送的报文，最终通过dpdk的dlvs发送到网络。



### dpdklvs的数据流详解

假设：

eth0  ip0  gw0 分别是ssh网口的名字、ip地址和网关；

eth1  ip1  gw1 分别是业务口的名字、ip地址和网关；

路由配置如下:

* 策略路由： 源ip是ip0的，走eth0；
* 默认路由设置为gw1，走eth1口；

相对于标准的lvs配置，就是新加了一个网卡(网线)，配置了一个ip用于ssh登录。

为了保证out方向的流量从这个新网卡出去，添加一条源ip的策略路由。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 流量类型 | Dr | | fullnat | |
| In | out | in | Out |
| Ssh访问：  工程师对ip0的ssh访问 | gw0到eth0 | eth0到gw0。  根据ssh\_ip，配置源策略路由，走eth0 | gw0到eth0 | eth0到gw0。  根据ssh\_ip，配置源策略路由，走eth0 |
| 业务数据：  用户对vip+port的访问 | gw1到eth1 | eth1到realserver，  二层通信。 | gw1到eth1 | eth1到gw1  这一步不查linux路由表 |
| BGP:  本机通过eth1发布bgp信息 | gw1到eth1 | eth1到gw1。  二层通信。 | eth1到gw1 | eth1到gw1。  二层通信。 |
| 本机对外的探测：  本机对realserver的健康检查 | gw1到eth1 | eth1到gw1。  二层通信。 | 无论是从eth0还是eth1进入 | 默认路由，  Eth1到gw1 |
| 外界对本机的探测：  监控、管控对本机的访问 | gw1到eth1 | eth1到gw1。  查linux的路由表，走默认路由。 | gw1到eth1 | Eth1到gw1。  查linux的路由表，走默认路由。 |

## 为什么要配置dpdk？

### 内存预分配

dpdk应用程序启动时，预先分配内存。分配的多少内存，从下面的值中读取：

/sys/devices/system/node/$NODENAME\_A/hugepages/hugepages-2048kB/nr\_hugepages

用户可以根据机器的内存大小，设置该内核变量。

dpdklvs在tools/0\_dlvs\_init.sh中设置的。

### cpu分配与独占

dpdk应用程序启动时，需要指定哪几个core运行dpdk程序，通过conf/dpdk\_core.conf来指定。

为了避免别的程序干扰dpdk程序的运行，需要设置boot/gruf.conf，设置cpu isolcpus

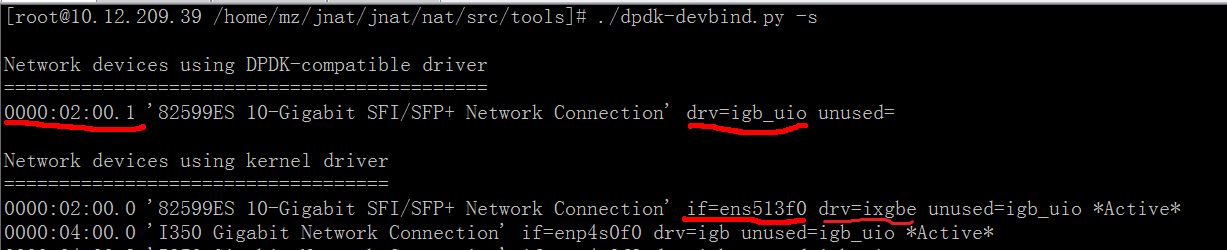
### dpdk内核模块

igb\_uio.ko rte\_kni.ko

在启动dpdk应用程序之前，确保加载这些内核模块，并且网卡已经绑定到igb\_uio.ko上。

这个步骤，在tools/0\_dlvs\_init.sh中完成的。

通过下面的命令，可以查看是否绑定成功。



见上图，0000:02:00.1绑定dpdk成功，drv=igb\_uio。

与之对比的是，0000:02:00.0 drv=ixgbe，驱动是ixgbe.ko，网卡名是ens513f0

### dpdk应用程序启动后的状态

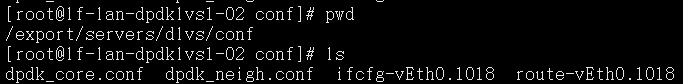
dpdk应用程序(dlvs)启动后，生成了一个新网卡vEth0，就是dpdk绑定的网卡。

可以简单的理解为，除了vip+port匹配的报文，所有的报文，都是经过vEth0收发。

需要检查vEth0上面的ip地址、vlan、路由。

另外，可以通过dlvsadm –showarp/showroute 来查看dpdk的路由arp，vip+port匹配的报文的转发是否正确。

### dpdklvs的配置文件



dpdk\_core.conf，指定运行dpdk的core

dpdk\_neigh.conf，用于设置vip\_port匹配报文的arp

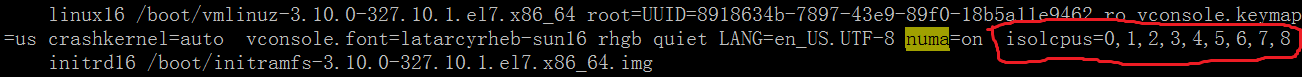
ifcfg-vEth0.1018，设置ip地址

route-vEth0.1018，设置所有流量的路由

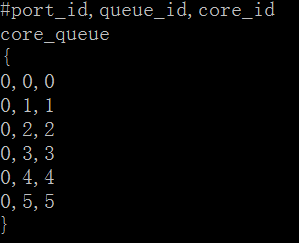
## 如何配置dpdklvs？

### cpu

先设置grub，隔离cpu，然后重启：



设置哪几个核运行dlvs：



### 内存

tools/0\_dlvs\_init.sh

C:\Users\wangyan50\Documents\JDdongdong\JIMEnterprise\wangyan627\Temp\JdOnline20170508153213.png

### 网络

添加vlan, 设置ip, 设置路由，这些配置，既在linux网络配置，又在dlvs的转发平面配置。

C:\Users\wangyan50\Documents\JDdongdong\JIMEnterprise\wangyan627\Temp\JdOnline20170508153437.png

设置arp，用于设置dlvs转发平面的arp

C:\Users\wangyan50\Documents\JDdongdong\JIMEnterprise\wangyan627\Temp\JdOnline20170508153653.png

配置bgp，与标准的lvs配置一致，不再描述。

### 应用lvs

通过配置keepalived，来配置lvs；

通过dlvsadm，设置dlvs转发时的参数，包括timeout\log\_level等。

## fullnat基础知识

Fullnat的原理：

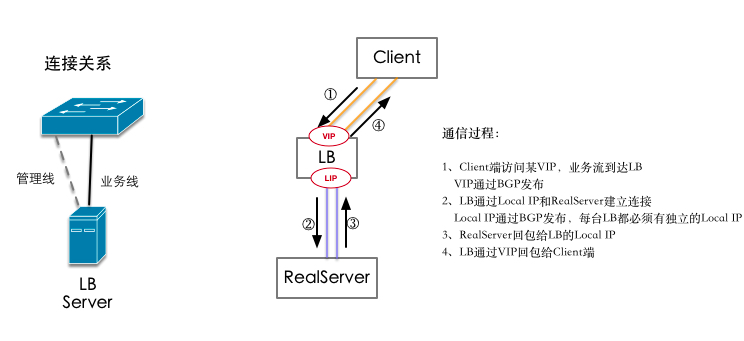
in： cip : vip 🡪 lvs处理 🡪 lip : rip

out：rip : lip 🡪 lvs处理 🡪 vip : cip

lvs上的配置fullnat：

* lo: vip, lip,
* 业务网卡：互联ip
* 管理网卡：管理ip
* 对网络的要求：3层可达。
* 本机路由: 默认路由设置在业务网口；管理网口上，设置本网口源ip的策略路由，确保正确回包。
* 配置bgp、keepalived

## dpdklvs疑问：



Question:

* ***为什么LB Server的Local IP要通过BGP发布呢？***
  + *我的理解*
    - *每台LB Server有唯一的、固定不变的Local IP*
    - *这种情况下，通过BGP动态发布是不必要的。*

答案：为了应对高并发的情况，每个lvs server需要配置几十个local ip。

之所以选择动态bgp，为了缩小二层的规模。

* ***管理流量与业务流量分离，是如何实现的***

**答：**严格的讲，管理流量和业务流量没有分开。

只是增加的一个网口，用于ssh登录。其它的流量依旧，都是经过业务口。

这样做的目的，业务相关流量路径，与内核版本的lvs完全一致，避免引发新问题。

