Universidade Federal de São Carlos

# ESTUDO DA UTILIZAÇÃO DO EXABGP COMO ENGINE DE ROTEAMENTO NO ROUTEFLOW

**Alfredo Guilherme da Silva Souza**
**Denis Pereira de Lima**
**Gaio Belitardo de Oliveira**
**Manoel Aranda de Almeida**
**Roan Simões da Silva**

São Carlos
2013

# LISTA DE FIGURAS

# SUMÁRIO

# INTRODUÇÃO

RouteFlow [1] é uma solução baseada na tecnologia OpenFlow [2] que, por sua vez, surgiu do conceito de Redes Definidas por Software (SDN [3]), cujo propósito central é separar o plano de controle do plano de dados na internet, permitindo o tratamento de fluxos de dados transmitidos pela rede e a decisão sobre o seu encaminhamento.

O amadurecimento da tecnologia OpenFlow, permitiu a fabricação de equipamentos (*switches* OpenFlow) que suportam a utilização do protocolo OpenFlow, o que garante um controle dos dados, separado do equipamento, em um servidor remoto chamado de Controlador.

O papel do RouteFlow é criar uma abstração ou virtualização da infraestrutura física de um ou mais sistemas autônomos (*ASs*), utilizando máquinas "executando" protocolos de roteamento, para definir e propagar rotas de encaminhamento de fluxos de dados para o plano físico operando com *switches* OpenFlow.

Neste trabalho o RouteFlow será configurado para operar utilizando o protocolo BGP de modo que permita também a injeção de rotas.

O RouteFlow utiliza a suite de roteamento Quagga [4] executando em máquinas virtuais para processar rotas, operando com protocolos de roteamento como: RIP, OSPF, BABEL, BGP, entre outros [5].

O propósito inicial do trabalho era substituir a engine de roteamento Quagga pelo ExaBGP [6], entretanto, considerando que o mesmo não possui as funcionalidades necessárias que são realizadas pelo Quagga, o ExaBGP foi configurado para injetar rotas, mantendo-se a engine Quagga.

A *engine* ExaBGP é utilizada em redes definidas por *software* (SDN), e permite controlar a rede a partir de equipamentos servidores usuais, ou seja, utilizando hardware *commodity*, ao invés de caros equipamentos especifícos para rede. O ExaBGP é capaz de receber atualizações e dados do protocolo BGP e realizar as manipulações necessárias através da utilização de *scripts* escritos em *shell script*.

Na figura 1 é apresentado como seria o cenário utilizado neste trabalho no modo tradicional, onde seriam utilizados roteadores convencionais.

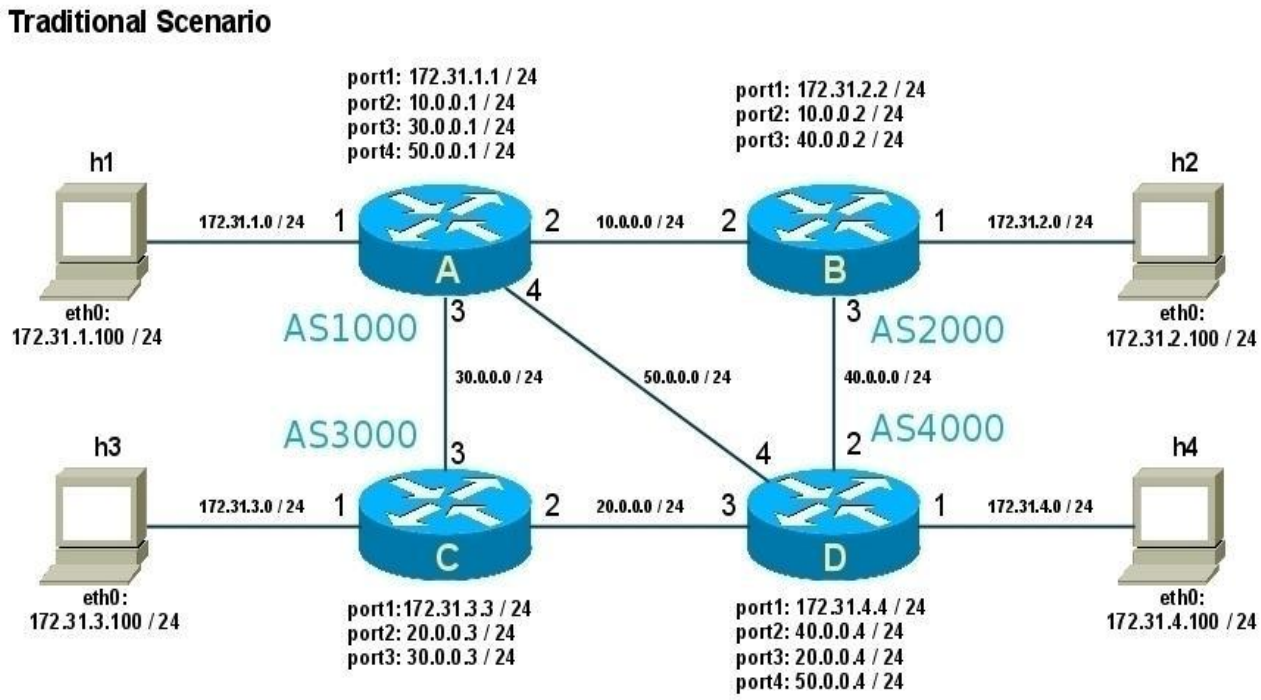


Figura 1: Cenário tradicional utilizando roteadores

Fonte: (adaptado de <https://sites.google.com/site/routeflow/documents/tutorial2-four-routers-with-ospf>, 16/06/2013).

Como é possível visualizar na figura, são criados quatro sistemas autônomos, representados pelos AS1000, AS2000, AS3000 e AS4000, que irão se interligar utilizando o protocolo de roteamento BGP.

Aplicando os conceitos de Redes Definidas por Software e utilizando switches OpenFlow, este cenário pode ser substituído, com o uso do RouteFlow, pelo cenário apresentado na Figura 2.

Neste novo cenário, os roteadores convencionais foram substituídos por switches Openflow, que serão virtualizados neste trabalho com o uso de uma máquina virtual Mininet[7] e o uso do RouteFlow.

É possível visualizar ainda que o ExaBGP será configurado para injetar rotas à partir da rf3vmD, nas demais máquinas virtuais, rf3vmA, rf3vmB e rf3vmC.

Nas seções seguintes, serão apresentados os passos necessários para a realização deste trabalho.

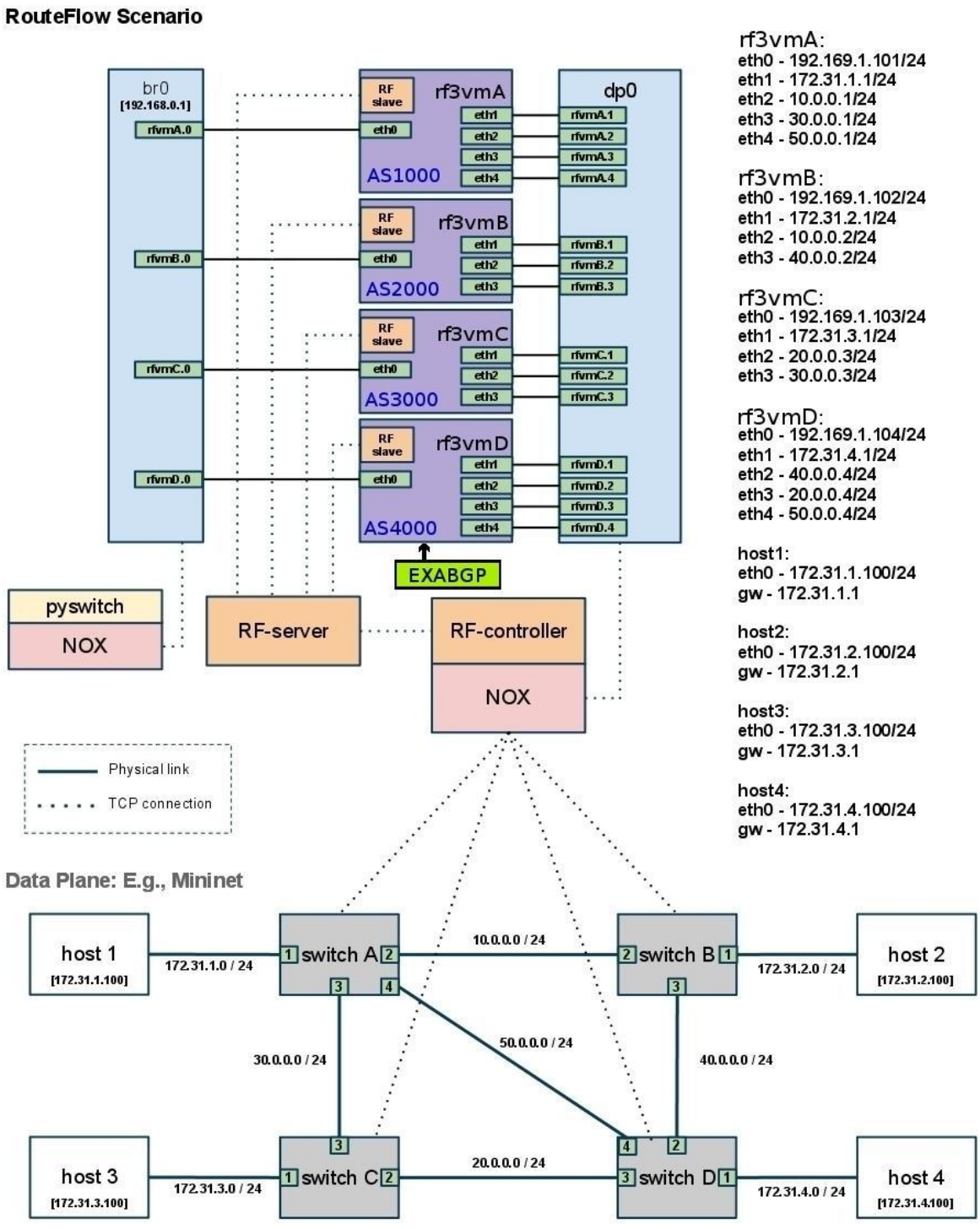


Figura 2: Cenário utilizando conceitos de redes definidas por software

Fonte: (adaptado de <https://sites.google.com/site/routeflow/documents/tutorial2-four-routers-with-ospf>, 16/06/2013).

# 1. INSTALAÇÃO DO UBUNTU SERVER 12.04

O Ubuntu Server, versão 12.04 de 32 bits, é um sistema operacional GNU/Linux e foi instalado de maneira padrão com os pacotes mínimos utilizando o sistema de virtualização *Oracle VirtualBox* [8].

Para este sistema, foi habilitada uma placa de rede, operando em modo NAT, permitindo realizar a instalação dos pacotes necessários, que serão descritos a seguir. Após a instalação dos pacotes e do *download* do RouteFlow, a placa de rede foi configurada para operar em modo *host only*.

Sem realizar alterações nos repositórios oficiais, foi realizado o *update* e o *upgrade* dos pacotes através do utilitário *apt-get*. Em seguida, utilizando novamente o *apt-get*, foram instalados os seguintes pacotes necessários para o funcionamento do RouteFlow:

```
$ sudo apt-get install build-essential git libboost-dev \
libboost-program-options-dev libboost-thread-dev \
libboost-filesystem-dev iproute-dev openvswitch-switch \
mongodb python-pymongo exabgp
```

Com esses pacotes instalados, foi possível realizar a instalação do RouteFlow, como descrito no próximo item.

# 2. INSTALAÇÃO DO ROUTEFLOW

Para instalar o RouteFlow foi realizado o *download* a partir do repositório Github [9] (Sistema de Controle de Versões Git), criado para a realização deste trabalho. Este repositório é um *fork* do projeto oficial do RouteFlow. O comando necessário para realizar a clonagem do conteúdo disponível no Github foi:

*$ git clone git://github.com/roansimoes/RouteFlow.git*

*Nota:* O git foi instalado previamente no Ubuntu Server, a partir do comando *apt-get* mencionado no item 2.

# 3. CONSTRUÇÃO DO COMPONENTE RFCLIENT DO SISTEMA ROUTEFLOW

Após a conclusão do *download* dos arquivos do RouteFlow é necessário a criação do componente RF-Client, para que seja possível realizar as simulações com o RouteFlow. Isso porque, o RF-Client é o responsável pelo anúncio das tabelas de roteamento ao componente RF-Server e representa, nesse cenário, a inteligência virtualizada de um *switch openflow* físico.

Dentro do diretório raiz do RouteFlow, é necessário digitar o seguinte comando:

*$ make rfclient*

# 4. CONSTRUÇÃO DO RFTEST 3 NO SISTEMA ROUTEFLOW

Para este trabalho, o rftest2 foi adaptado para que operasse utilizando o protocolo BGP, ao invés do protocolo OSPF. Deste modo, foi criado o rftest3 com base no rftest2 e com as alterações que serão apresentadas nas seções seguintes.

## 4.1. CÓPIA DE ARQUIVOS

Inicialmente é realizada a cópia dos arquivos do rftest2, para que seja possível alterá-los mantendo os originais. Os comandos realizados foram:

*$ cp rftest2 rftest3*
*$ cp create create_rftest3*
*$ cp -r config config_rftest3*
*$ cp rfteste2config.csv rftest3config.csv*

No novo diretório config_rftest3 é necessário manter apenas os diretórios das máquinas virtuais (VMs) rfvmA, rfvmB, rfvmC e rfvmD, que tiveram seus nomes alterados para rf3vmA, rf3vmB, rf3vmC, rf3vmD, de modo a serem separadas totalmente do rftest2. Cada um destes diretórios se refere a uma das quatro máquinas virtuais que serão criadas posteriormente utilizando o sistema *lightweight* de virtualização para GNU/Linux: LXC [10].

## 4.2. EDIÇÃO DE ARQUIVOS RF3VMX

Deve-se alterar os arquivos de nome “config”, que contém as configurações das VMs (também conhecidos como *containers*) LXC, localizados no diretório raiz de cada VM para substituir as ocorrências de rfvmX por rf3vmX, onde X é a identificação A, B, C ou D das VMs. Abaixo, o exemplo do arquivo config da máquina virtual rf3vmA, com destaque em vermelho dos itens alterados (basta realizar as mesmas modificações nos arquivos correspondentes às demais VMs):

```
lxc.utsname = rf3vmA
lxc.network.type = veth
lxc.network.flags = up
lxc.network.hwaddr = 02:a0:a0:a0:a0:a0
lxc.network.link=lxcbr0

lxc.network.type = veth
lxc.network.flags = up
lxc.network.veth.pair = rf3vmA.1
lxc.network.hwaddr = 02:a1:a1:a1:a1:a1

lxc.network.type = veth
lxc.network.flags = up
lxc.network.veth.pair = rf3vmA.2
lxc.network.hwaddr = 02:a2:a2:a2:a2:a2

lxc.network.type = veth
lxc.network.flags = up
lxc.network.veth.pair = rf3vmA.3
lxc.network.hwaddr = 02:a3:a3:a3:a3:a3

lxc.network.type = veth
lxc.network.flags = up
lxc.network.veth.pair = rf3vmA.4
lxc.network.hwaddr = 02:a4:a4:a4:a4:a4

lxc.devttydir = lxc
lxc.tty = 4
lxc.pts = 1024
lxc.rootfs = /var/lib/lxc/rf3vmA/rootfs
```

```
lxc.mount  = /var/lib/lxc/rf3vmA/fstab
lxc.arch = amd64
lxc.cap.drop = sys_module mac_admin
lxc.pivotdir = lxc_putold

# uncomment the next line to run the container unconfined:
#lxc.aa_profile = unconfined

lxc.cgroup.devices.deny = a
# Allow any mknod (but not using the node)
lxc.cgroup.devices.allow = c *:* m
lxc.cgroup.devices.allow = b *:* m
# /dev/null and zero
lxc.cgroup.devices.allow = c 1:3 rwm
lxc.cgroup.devices.allow = c 1:5 rwm
# consoles
lxc.cgroup.devices.allow = c 5:1 rwm
lxc.cgroup.devices.allow = c 5:0 rwm
#lxc.cgroup.devices.allow = c 4:0 rwm
#lxc.cgroup.devices.allow = c 4:1 rwm
# /dev/{,u}random
lxc.cgroup.devices.allow = c 1:9 rwm
lxc.cgroup.devices.allow = c 1:8 rwm
lxc.cgroup.devices.allow = c 136:* rwm
lxc.cgroup.devices.allow = c 5:2 rwm
# rtc
lxc.cgroup.devices.allow = c 254:0 rwm
#fuse
lxc.cgroup.devices.allow = c 10:229 rwm
#tun
lxc.cgroup.devices.allow = c 10:200 rwm
```

```
#full
lxc.cgroup.devices.allow = c 1:7 rwm
#hpet
lxc.cgroup.devices.allow = c 10:228 rwm
#kvm
lxc.cgroup.devices.allow = c 10:232 rwm
```

## 4.3. SCRIPT DESTROY_RF3VM

Durante o período de realização do projeto, era frequente a necessidade de recriar as máquinas virtuais após cada alteração, por isso, foi criado um *script*, localizado na pasta rftest, dentro do diretório raiz do RouteFlow, chamado destroy_vm_rftest3, que possui a função de destruir as VMs quando necessário. O arquivo de criação das máquina virtuais, está descrito no próximo item. O conteúdo do arquivo destroy_rf3vm é:

```
#!/bin/bash

if [ "$EUID" != "0" ]; then
  echo "You must be root to run this script. Sorry, dude!"
  exit 1
fi

LXCDIR="/var/lib/lxc"
CONFIG="config_rftest3"

# Clone the base container to make other containers based on config
cd $CONFIG
for VM in *
do
    lxc-destroy -n $VM
done
```

## 4.4. CRIAÇÃO DO CREATE_RFTEST3

As máquinas virtuais (VMs) utilizadas nesta simulação são virtualizadas através do sistema LXC, e podem também ser chamadas de *containers*. Neste trabalho são necessárias quatro VMs, chamadas rf3vmA, rf3vmB, rf3vmC e rf3vmD, todas criadas tendo como base a VM "base_rftest3". Cada VM executará uma instância da suite de roteamento, Quagga, e uma delas, a rf3vmD, também executará o injetor de rotas ExaBGP.

O script descrito abaixo é responsável por criar as VMs para o teste com o RouteFlow:

```
#!/bin/bash
 if [ "$EUID" != "0" ]; then
   echo "You must be root to run this script. Sorry, dude!"
   exit 1
 fi
 LXCDIR="/var/lib/lxc"
 CONFIG="config_rftest3"
 # Setup LXC and base container
 apt-get -y --force-yes install lxc
 mkdir -p $LXCDIR
 lxc-create -t ubuntu -n base_rftest3

 chroot $LXCDIR/base_rftest3/rootfs apt-get update
 # !!!
 # ADD THE PACKETS YOU NEED TO INSTALL HERE
 # !!!
 chroot $LXCDIR/base_rftest3/rootfs apt-get -y --force-yes install quagga libboost-thread-dev libboost-system-dev libboost-filesystem-dev libboost-program-options-dev rsyslog vlan tcpdump exabgp
 # !!!
```

```
# Clone the base container to make other containers based on config
cd $CONFIG
for VM in *
do
   lxc-clone -o base_rftest3 -n $VM
   cp -R $VM/* $LXCDIR/$VM
done
```

## 4.5. ALTERAÇÃO DOS DIRETÓRIOS DAS MÁQUINAS VIRTUAIS UTILIZADAS NA SIMULAÇÃO

Cada uma das quatro máquinas virtuais utilizadas na simulação precisa de alterações individuais, por isso são criados diretórios personalizados contendo os arquivos que serão alterados em relação ao rftest2.

Estes arquivos das 4 VM's utilizadas no rftest3, foram alterados da seguinte forma:

### 4.5.1. ARQUIVO DAEMON

Este arquivo ficou idêntico em todas as VMs, e está localizado em:
DIRETÓRIO_RAIZ_DO_ROUTEFLOW/config_rftest3/rf3vmX/rootfs/etc/quagga/daemon

```
zebra=yes
bgpd=yes
ospfd=no
ospf6d=no
ripd=no
ripngd=no
isisd=no
```

### 4.5.2. ARQUIVO BGPD.CONF

Este arquivo contém os parâmetros necessários para o RouteFlow, mais especificamente, para sua engine de roteamento Quagga funcionar utilizando o protocolo BGP, portanto, deve ser alterado para cada VM com os parâmetros personalizados de endereço IP da própria VM e dos vizinhos, além dos códigos dos sistemas autonômos (ASs). O conteúdo deste arquivo está em: DIRETÓRIO_RAIZ_DO_ROUTEFLOW/config_rftest3/rf3vmX/rootfs/etc/quagga/bgpd.conf, onde X deve ser substituído por A, B, C ou D:

### 4.5.3. VM RF3VMA

```
password routeflow
enable password routeflow
!
router bgp 1000
        bgp router-id 172.31.1.1
        network 172.31.1.0/24
        redistribute connected
        neighbor 10.0.0.2 remote-as 2000
        neighbor 30.0.0.3 remote-as 3000
        neighbor 50.0.0.4 remote-as 4000
        no auto-summary
!
log file /var/log/quagga/bgp.log
```

### 4.5.4. VM RF3VMB

```
password routeflow
enable password routeflow
!
router bgp 2000
```

```
        bgp router-id 172.31.2.1
        network 172.31.2.0/24
        redistribute connected
        neighbor 10.0.0.1 remote-as 1000
        neighbor 40.0.0.4 remote-as 4000
        no auto-summary
!
log file /var/log/quagga/bgp.log
```

#### 4.5.5. VM RF3VMC

```
password routeflow
enable password routeflow
!
router bgp 3000
        bgp router-id 172.31.3.1
        network 172.31.3.0/24
        redistribute connected
        neighbor 30.0.0.1 remote-as 1000
        neighbor 20.0.0.4 remote-as 4000
        no auto-summary
!
log file /var/log/quagga/bgp.log
```

#### 4.5.6. VM RF3VMD

```
password routeflow
enable password routeflow
!
router bgp 4000
        bgp router-id 172.31.4.1
```

```
        network 172.31.4.0/24
        redistribute connected
        neighbor 50.0.0.1 remote-as 1000
        neighbor 40.0.0.2 remote-as 2000
        neighbor 20.0.0.3 remote-as 3000
        no auto-summary
!
log file /var/log/quagga/bgp.log
```

### 4.5.7. O ARQUIVO EXABGP.CONF

O arquivo exabgp.conf, é responsável por definir os parâmetros para a injeção de rotas do ExaBGP. Para este trabalho o ExaBGP foi configurado na VM rf3vmD, para que injete rotas nas outras 3 VM's, indicando o caminho para alcançar o computador H4, virtualizado através da Mininet.

Como as rotas serão injetas apenas à partir da VM rf3vmD, este arquivo só foi alterado na mesma. O arquivo exabgp.conf se encontra em:

DIRETÓRIO_RAIZ_DO_ROUTEFLOW/config_rftest3/rf3vmD/rootfs/etc/exabgp/

E teve seu conteúdo alterado para:

```
neighbor 50.0.0.1{
        description "teste de injecao de rota em rf3vmA";
        router-id 50.0.0.4;
        local-address 50.0.0.4;
        local-as 4000;
        peer-as 1000;
        static{
                route 172.31.4.100/32 next-hop 50.0.0.4;
        }
```

```
}
neighbor 40.0.0.2{
        description "teste de injecao de rota em rf3vmB";
        router-id 40.0.0.4;
        local-address 40.0.0.4;
        local-as 4000;
        peer-as 2000;
        static{
                route 172.31.4.100/32 next-hop 40.0.0.4;
        }
}
neighbor 20.0.0.3{
        description "teste de injecao de rota em rf3vmC";
        router-id 20.0.0.4;
        local-address 20.0.0.4;
        local-as 4000;
        peer-as 3000;
        static{
                route 172.31.4.100/32 next-hop 20.0.0.4;
        }
}
```

### 4.5.8. ARQUIVO RFTEST3

Este arquivo, encontrado no diretório: DIRETÓRIO_RAIZ_DO_ROUTE_FLOW/rftest/ é responsável por iniciar a simulação e foi alterado - inclusive com a inclusão de comentários - para:

```
#!/bin/bash
#teste para ver se o usuário é o root.
if [ "$EUID" != "0" ]; then
```

```
  echo "You must be root to run this script."
  exit 1
fi

# definição de variáveis e valores.
SCRIPT_NAME="rftest3"
LXCDIR=/var/lib/lxc
MONGODB_CONF=/etc/mongodb.conf
MONGODB_PORT=27017
CONTROLLER_PORT=6633
RF_HOME=..
# atualização do path do sistema incluindo novos locais.
export PATH=$PATH:/usr/local/bin:/usr/local/sbin
export PYTHONPATH=$PYTHONPATH:$RF_HOME

# acesso ao diretório do RouteFlow.
cd $RF_HOME

#função que utiliza o comando nc para AGUARDAR conexões na porta passada
# como primeiro parâmetro.
# mais à frente nesse script, essa função é chamada para utilizar a porta do MONGODB.
wait_port_listen() {
        port=$1
        while ! `nc -z localhost $port` ; do
      echo -n .
      sleep 1
        done
}
# função que altera o texto passado como parâmetro em NEGRITO.
echo_bold() {
        echo -e "\033[1m${1}\033[0m"
```

```
}
# essa função é executada recursivamente para ir finalizando os processos mais internos até
        # os mais externos (árvore de processos).
kill_process_tree() {
        top=$1
        pid=$2
        children=`ps -o pid --no-headers --ppid ${pid}`
        for child in $children
        do
              kill_process_tree 0 $child
        done

        if [ $top -eq 0 ]; then
                kill -9 $pid &> /dev/null
        fi
}

reset() {
   init=$1;
   if [ $init -eq 1 ]; then
      echo_bold "-> Starting $SCRIPT_NAME";
   else
     # pressionando "^C" a qquer tempo, a função reset é chamada com o 1º parâmetro=0
     # e cai nesse fluxo.

     # a função kill_process_tree é chamada com o 1º parâmetro=1 e o
    # 2º parâmetro "$$", que refere-se ao PID do processo atual, ou seja, do script.
     echo_bold "-> Stopping child processes...";
      kill_process_tree 1 $$
   fi
   # finalização do data plane virtual criado com o openvswitch.
```

```
ovs-vsctl del-br dp0 &> /dev/null;
ovs-vsctl emer-reset &> /dev/null;

# encerramento das máquinas virtuais do lxc.
echo_bold "-> Stopping and resetting LXC VMs...";
      for vm in "rf3vmA" "rf3vmB" "rf3vmC" "rf3vmD"
      do
   lxc-shutdown -n "$vm";
   while true
      do
           if lxc-info -q -n "$vm" | grep -q "STOPPED"; then
         break;
      fi
      echo -n .
      sleep 1
   done
      done
#limpando a base de dados do MongoDB
echo_bold "-> Deleting (previous) run data...";
      mongo db --eval "
   db.getCollection('rftable').drop();
   db.getCollection('rfconfig').drop();
   db.getCollection('rfstats').drop();
   db.getCollection('rfclient<->rfserver').drop();
   db.getCollection('rfserver<->rfproxy').drop();
      "
   #excluindo os arquivos das máquinas virtuais
      rm -rf /var/lib/lxc/rf3vmA/rootfs/opt/rfclient;
      rm -rf /var/lib/lxc/rf3vmB/rootfs/opt/rfclient;
      rm -rf /var/lib/lxc/rf3vmC/rootfs/opt/rfclient;
      rm -rf /var/lib/lxc/rf3vmD/rootfs/opt/rfclient;
```

```
}

# Aqui é onde tudo começa!!!
# Ao chamar a função reset com o parâmetro "1", o script exibe a
# primeira mensagem da função reset. Ela indica o início da execução do script.
reset 1

#  o comando trap associa comandos passados como parâmetro entre aspas separados por ";"
#  ao sinal fora das aspas e em maiúsculo.
# INT equivale ao pressionamento de ^C (interrompe a execução de comandos ou scripts).
trap "reset 0; exit 0" INT

echo_bold "-> Setting up the management bridge (lxcbr0)..."
#configurando o IP para a brigde do lxc
ifconfig lxcbr0 192.169.1.1 up

echo_bold "-> Setting up MongoDB..."
sed -i "/bind_ip/c\bind_ip = 127.0.0.1,192.169.1.1" $MONGODB_CONF
service mongodb restart
wait_port_listen $MONGODB_PORT

echo_bold "-> Configuring the virtual machines..."
# Create the rfclient dir
mkdir /var/lib/lxc/rf3vmA/rootfs/opt/rfclient
mkdir /var/lib/lxc/rf3vmB/rootfs/opt/rfclient
mkdir /var/lib/lxc/rf3vmC/rootfs/opt/rfclient
mkdir /var/lib/lxc/rf3vmD/rootfs/opt/rfclient

# Copy the rfclient executable
cp build/rfclient /var/lib/lxc/rf3vmA/rootfs/opt/rfclient/rfclient
cp build/rfclient /var/lib/lxc/rf3vmB/rootfs/opt/rfclient/rfclient
```

```
cp build/rfclient /var/lib/lxc/rf3vmC/rootfs/opt/rfclient/rfclient
cp build/rfclient /var/lib/lxc/rf3vmD/rootfs/opt/rfclient/rfclient

# We sleep for a few seconds to wait for the interfaces to go up
# Grava as linhas a seguir no arquivo run_rfclient.sh (um script) em cada vm do routeflow.
echo "#!/bin/sh" > /var/lib/lxc/rf3vmA/rootfs/root/run_rfclient.sh
echo "sleep 3" >> /var/lib/lxc/rf3vmA/rootfs/root/run_rfclient.sh
echo "/etc/init.d/quagga start" >> /var/lib/lxc/rf3vmA/rootfs/root/run_rfclient.sh
echo "/opt/rfclient/rfclient > /var/log/rfclient.log" >>
/var/lib/lxc/rf3vmA/rootfs/root/run_rfclient.sh

echo "#!/bin/sh" > /var/lib/lxc/rf3vmB/rootfs/root/run_rfclient.sh
echo "sleep 3" >> /var/lib/lxc/rf3vmB/rootfs/root/run_rfclient.sh
echo "/etc/init.d/quagga start" >> /var/lib/lxc/rf3vmB/rootfs/root/run_rfclient.sh
echo "/opt/rfclient/rfclient > /var/log/rfclient.log" >>
/var/lib/lxc/rf3vmB/rootfs/root/run_rfclient.sh

echo "#!/bin/sh" > /var/lib/lxc/rf3vmC/rootfs/root/run_rfclient.sh
echo "sleep 3" >> /var/lib/lxc/rf3vmC/rootfs/root/run_rfclient.sh
echo "/etc/init.d/quagga start" >> /var/lib/lxc/rf3vmC/rootfs/root/run_rfclient.sh
echo "/opt/rfclient/rfclient > /var/log/rfclient.log" >>
/var/lib/lxc/rf3vmC/rootfs/root/run_rfclient.sh

echo "#!/bin/sh" > /var/lib/lxc/rf3vmD/rootfs/root/run_rfclient.sh
echo "sleep 3" >> /var/lib/lxc/rf3vmD/rootfs/root/run_rfclient.sh
echo "/etc/init.d/quagga start" >> /var/lib/lxc/rf3vmD/rootfs/root/run_rfclient.sh
echo "/opt/rfclient/rfclient > /var/log/rfclient.log" >>
/var/lib/lxc/rf3vmD/rootfs/root/run_rfclient.sh

#conceder permissão de execução aos scripts rfclient.sh.
chmod +x /var/lib/lxc/rf3vmA/rootfs/root/run_rfclient.sh
```

```
chmod +x /var/lib/lxc/rf3vmB/rootfs/root/run_rfclient.sh
chmod +x /var/lib/lxc/rf3vmC/rootfs/root/run_rfclient.sh
chmod +x /var/lib/lxc/rf3vmD/rootfs/root/run_rfclient.sh

echo_bold "-> Starting the virtual machines..."
lxc-start -n rf3vmA -d
lxc-start -n rf3vmB -d
lxc-start -n rf3vmC -d
lxc-start -n rf3vmD -d

# iniciando o controlador e aguardando conexões em sua porta...
echo_bold "-> Starting the controller and RFPRoxy..."
cd pox
./pox.py log.level --=INFO topology openflow.topology openflow.discovery rfproxy rfstats &
cd -
wait_port_listen $CONTROLLER_PORT

# iniciando o rfserver com a verificação de parâmetros em arquivo ".csv".
echo_bold "-> Starting RFServer..."
./rfserver/rfserver.py rftest/rftest3config.csv &

#criando a rede de controle do plano de dados
echo_bold "-> Starting the control plane network (dp0 VS)..."
ovs-vsctl add-br dp0
ovs-vsctl add-port dp0 rf3vmA.1
ovs-vsctl add-port dp0 rf3vmA.2
ovs-vsctl add-port dp0 rf3vmA.3
ovs-vsctl add-port dp0 rf3vmA.4
ovs-vsctl add-port dp0 rf3vmB.1
ovs-vsctl add-port dp0 rf3vmB.2
ovs-vsctl add-port dp0 rf3vmB.3
```

```
ovs-vsctl add-port dp0 rf3vmC.1
ovs-vsctl add-port dp0 rf3vmC.2
ovs-vsctl add-port dp0 rf3vmC.3
ovs-vsctl add-port dp0 rf3vmD.1
ovs-vsctl add-port dp0 rf3vmD.2
ovs-vsctl add-port dp0 rf3vmD.3
ovs-vsctl add-port dp0 rf3vmD.4
ovs-vsctl set Bridge dp0 other-config:datapath-id=7266767372667673
ovs-vsctl set-controller dp0 tcp:127.0.0.1:$CONTROLLER_PORT

#apenas mensagens de orientação após carregar o controlador.
echo_bold "---"
echo_bold "This test is up and running."
echo_bold "Start Mininet:"
echo_bold "  $ sudo mn --custom mininet/custom/topo-4sw-4host.py --topo=rftest2"
echo_bold "     --controller=remote,ip=[host address],port=6633 --pre=ipconf"
echo_bold "Replace [host address] with the address of this host's interface "
echo_bold "connected to the Mininet VM."
echo_bold "Then try pinging everything:"
echo_bold "  mininet> pingall"
echo_bold "You can stop this test by pressing CTRL+C."
echo_bold "---"
wait
exit 0
```

### 4.5.9. ARQUIVO RFTESTE3CONFIG.CSV

Este arquivo contém informações utilizadas pelo RFS-rever constituídas dos campos:

*vm_id* (identificador de cada VM que representa a abstração de cada swich OpenFlow, definido por endereço MAC);

*vm_port* (identificação das portas de cada VM, isto é, de cada switch virtualizado);

*ct_id* (identificador do controlador, pois pode haver mais de um);

dp_id (identificador de cada switch OpenFlow físico);

dp_port (identificador de cada porta de cada switch OpenFlow).

Esse arquivo possui conteúdo idêntico ao rftest2config.csv, tendo sido alterado apenas seu nome.

# 5. SIMULAÇÕES

Nas seções seguintes serão descritas as simulações de teste realizadas neste trabalho, com o intuito de verificar o funcionamento do RouteFlow operando com o protocolo BGP através do uso do sistema ExaBGP.

## 5.1. FUNCIONAMENTO DO BGP PADRÃO DO QUAGGA

Com as alterações realizadas anteriormente já há comunicação entre as 4 VM's utilizando o protocolo BGP com rotas distribuídas pelo QUAGGA.

Foi utilizada a Mininet, uma máquina virtual que cria redes virtuais, para realizar o teste de comunicação, e para isso é necessário a instalação padrão da Mininet, e a cópia dos arquivos ipconf e topo-4sw-4host.py, localizados no diretório rftest do RouteFlow para a Mininet. Para iniciar a simulação é necessário o comando:

*# mn –custom topo-4sw-4host.py –topo=rftopo –contoller=remote –ip=IP_DO_ROUTE_FLOW –port 6633*

No terminal da mininet é necessário carregar o arquivo ipconf, com o comando

*source ipconf*

Foi utilizada a Mininet para realizar o teste de comunicação, como visto na figura 3, onde podemos ver que houve comunicação entre todos os hosts virtuais.

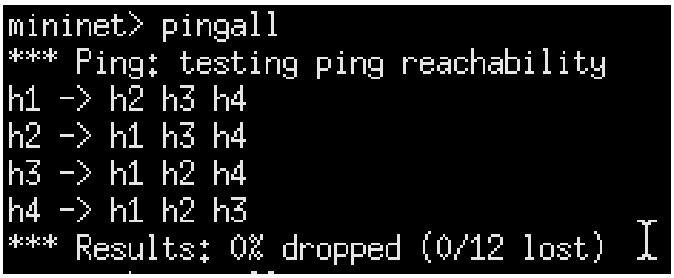

Figura 3: Teste de conectividade através da mininet

Outra maneira de testar, consiste em executar os terminais das VM's A, B, C ou D, executando o comando:

*# lxc-console -n rf3vmX*

Será necessário informar usuário e senha ("ubuntu" para ambos os campos), e após o login consultando a tabela de rotas com o comando:

$ route -n

Na figura 4 é possível ver a tabela de roteamento existente na rf3vmB.

Rotas distribuídas pelo Quagga

| Destination | Gateway | Genmask | Flags | Metric | Ref | Use | Iface |
|---|---|---|---|---|---|---|---|
| 10.0.0.0 | 0.0.0.0 | 255.255.255.0 | U | 0 | 0 | 0 | eth2 |
| 20.0.0.0 | 40.0.0.4 | 255.255.255.0 | UG | 1 | 0 | 0 | eth3 |
| 30.0.0.0 | 10.0.0.1 | 255.255.255.0 | UG | 1 | 0 | 0 | eth2 |
| 40.0.0.0 | 0.0.0.0 | 255.255.255.0 | U | 0 | 0 | 0 | eth3 |
| 50.0.0.0 | 40.0.0.4 | 255.255.255.0 | UG | 1 | 0 | 0 | eth3 |
| 172.31.1.0 | 10.0.0.1 | 255.255.255.0 | UG | 0 | 0 | 0 | eth2 |
| 172.31.2.0 | 0.0.0.0 | 255.255.255.0 | U | 0 | 0 | 0 | eth1 |
| 172.31.3.0 | 40.0.0.4 | 255.255.255.0 | UG | 0 | 0 | 0 | eth3 |
| 172.31.4.0 | 40.0.0.4 | 255.255.255.0 | UG | 0 | 0 | 0 | eth3 |
| 192.169.1.0 | 0.0.0.0 | 255.255.255.0 | U | 0 | 0 | 0 | eth0 |

Figura 4: Rotas presentes na rf3vmB

## 5.2. INJEÇÃO DE ROTAS COM O EXABGP

O ExaBGP pode ser utilizado para injetar rotas em outros roteadores. Esta injeção pode ocorrer mediante algum evento, como por exemplo, sendo executado em conjunto com um

sistema IDS operando de modo reativo, que ao identificar alguma tentativa de ataque, poderia injetar rotas de modo a impedir a ocorrência de um incidente.

Na simulação deste trabalho a VM rf3vmD injeta rotas nas outras 3 VM's para que consigam se comunicar com o host H4. Isso permite que haja comunicação com o host H4, virtualizado pela Mininet, mesmo se o Quagga não estiver funcionando na rf3vmD. Estas rotas são definidas no arquivo exabgp.conf, apresentado anteriormente.

Para injetar as rotas configuradas no EXABGP é necessário parar a execução do serviço Quagga na rf3vmD, com o comando:

```
# service quagga stop
```

Em seguida, é necessário alterar o arquivo /etc/default/exabgp, acrescentando o parâmetro que permite a execução do ExaBGP:

```
EXABGPRUN=yes
```

E por fim iniciar o serviço, com o comando:

```
#service exabgp start
```

Neste momento, as rotas configuradas para serem injetadas no arquivo exabgp.conf apresentado anteriormente já serão vistas nas tabelas de roteamento das máquinas alvo. Na figura 5, podemos ver a rota injetada na rf3vmB pelo ExaBGP que está em execução na rf3vmD.

É possível ainda notar que a rota para o endereço 172.31.4.0/24 (vide figura 4) não existe mais, pois a mesma era distribuída pelo Quagga, que não está mais executando na rf3vmD.

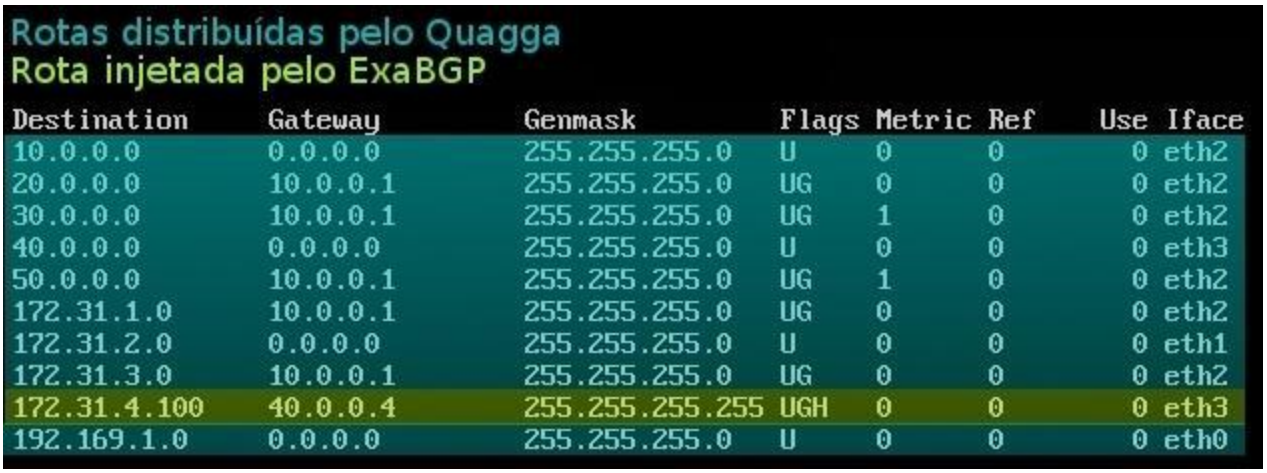


Figura 5: Rotas presentes na rf3vmB, em destaque a rota injetada pelo ExaBGP

# CONCLUSÃO

O Quagga é a engine de roteamento utilizada pelo RouteFlow que permite além do uso de BGP, outros protocolos, como já foi mencionado anteriormente. O ExaBGP é capaz de receber atualizações de dados sobre rotas BGP e realizar as manipulações necessárias através da utilização de *scripts* escritos em *shell script*.

Deste modo, após vários testes, conclui-se que o ExaBGP em suas funcionalidades atuais, não pode substituir totalmente o Quagga, por terem propósitos diferentes. Entretanto, o ExaBGP pode ser utilizado em conjunto ao Quagga, podendo trazer benefícios, especialmente em aspectos de segurança se combinado com sistemas de detecção de intrusão, por exemplo.

Neste trabalho, foi apresentado um cenário semelhante ao do rftest2, presente na documentação do sítio oficial do projeto do RouteFlow, substituindo-se o protocolo OSPF utilizado naquele teste pelo BGP, onde foi demonstrado como o ExaBGP poderia ser utilizado como um injetor de rotas.

As alterações realizadas neste trabalho foram sincronizadas no repositório Github, disponível no endereço https://github.com/roansimoes/RouteFlow, permitindo que após

o passo de download do RouteFlow, os testes sejam realizados sem a necessidade de alterar todos os arquivos novamente.

# REFERÊNCIAS BIBLIOGRÁFICAS

[1] <https://sites.google.com/site/routeflow/home>. Acesso em: 08 jun. 2013.

[2] <http://www.openflow.org/wp/documents/>. Acesso em: 10 jun. 2013.

[3] <https://www.opennetworking.org/sdn-resources/sdn-definition>. Acesso em: 10 jun. 2013.

[4] <http://www.nongnu.org/quagga/docs/quagga.html>. Acesso em: 14 jun. 2013.

[5] <http://www.nongnu.org/quagga/docs/quagga.html#Supported-RFCs>. Acesso em: 14 jun. 2013.

[6] <http://code.google.com/p/exabgp/>. Acesso em: 12 jun. 2013.

[7] < http://mininet.org/>. Acesso em 11 jun. 2013.

[8] <https://www.virtualbox.org/>. Acesso em: 15 jun. 2013.

[9] <https://github.com/>. Acesso em: 13 jun. 2013.

[10] <http://lxc.sourceforge.net/>. Acesso em: 20 jun. 2013.