Roteamento Virtual em Linux para roteamento de borda

Henrique de Moraes Holschuh
hmh@ima.sp.gov.br

Informática de Municípios Associados – Im@
Gerência de Engenharia de Telecomunicações

GTER 32
São Paulo, 02/12/2011

Conteúdo

- Sobre essa apresentação
- Conceitos
- *Containers, Network Namespaces, veth*
- Usando roteamento virtual na borda
- Performance

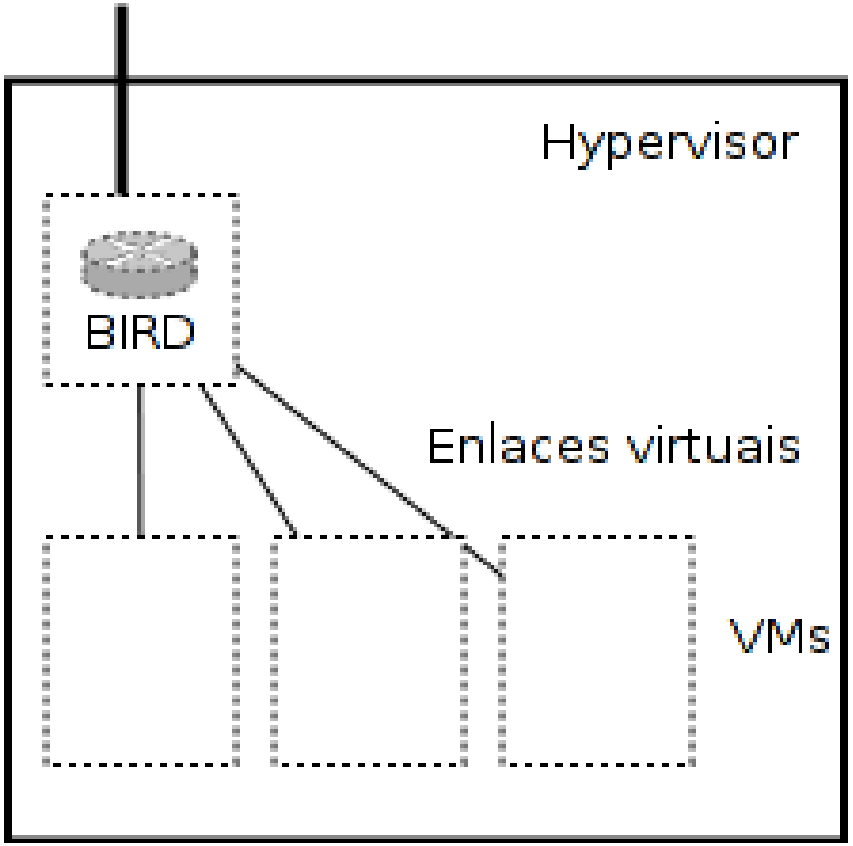
Sobre esta apresentação

- Terá como objetivo introduzir o roteamento virtual em Linux e apresentar os *network namespaces*
- Originalmente planejada para conter forte componentes quantitativos e rigor estatístico
- Infelizmente estes objetivos não puderam ser alcançados: a apresentação perdeu o direito de utilizar a palavra “alta performance” no título

Roteador Virtual (VR)

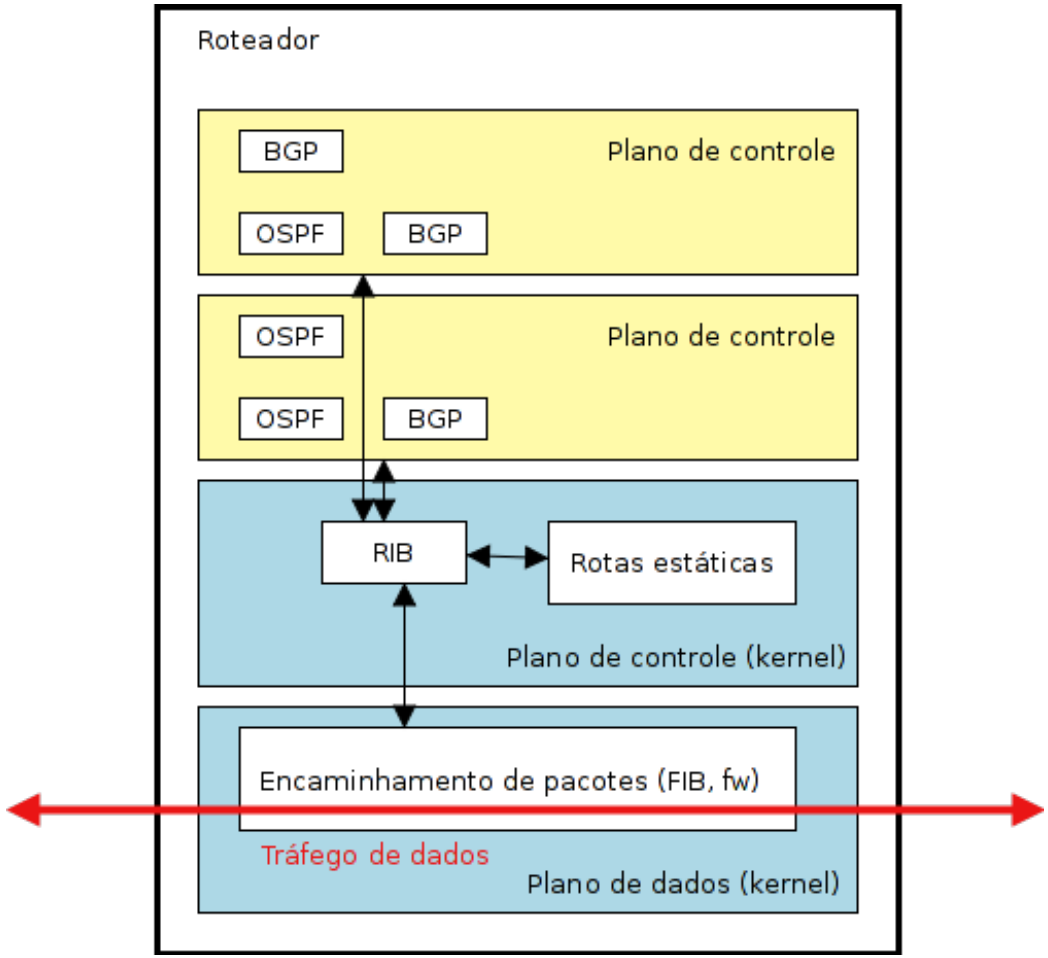
- Para alguns, é o roteador em software rodando em caixa genérica, ou um IP virtual sob VRRP...
- Definição: partição de um roteador (caixa) em diversas unidades de roteamento independentes:
  - Roteador com diversos planos de controle (RIBs, protocolos) distintos e isolados (VR)
  - Roteador com diversos planos de controle (RIBs, protocolos) e de dados (FIBs, interfaces) distintos e isolados (VRF)

Roteador “Virtual”: roteador para VMs

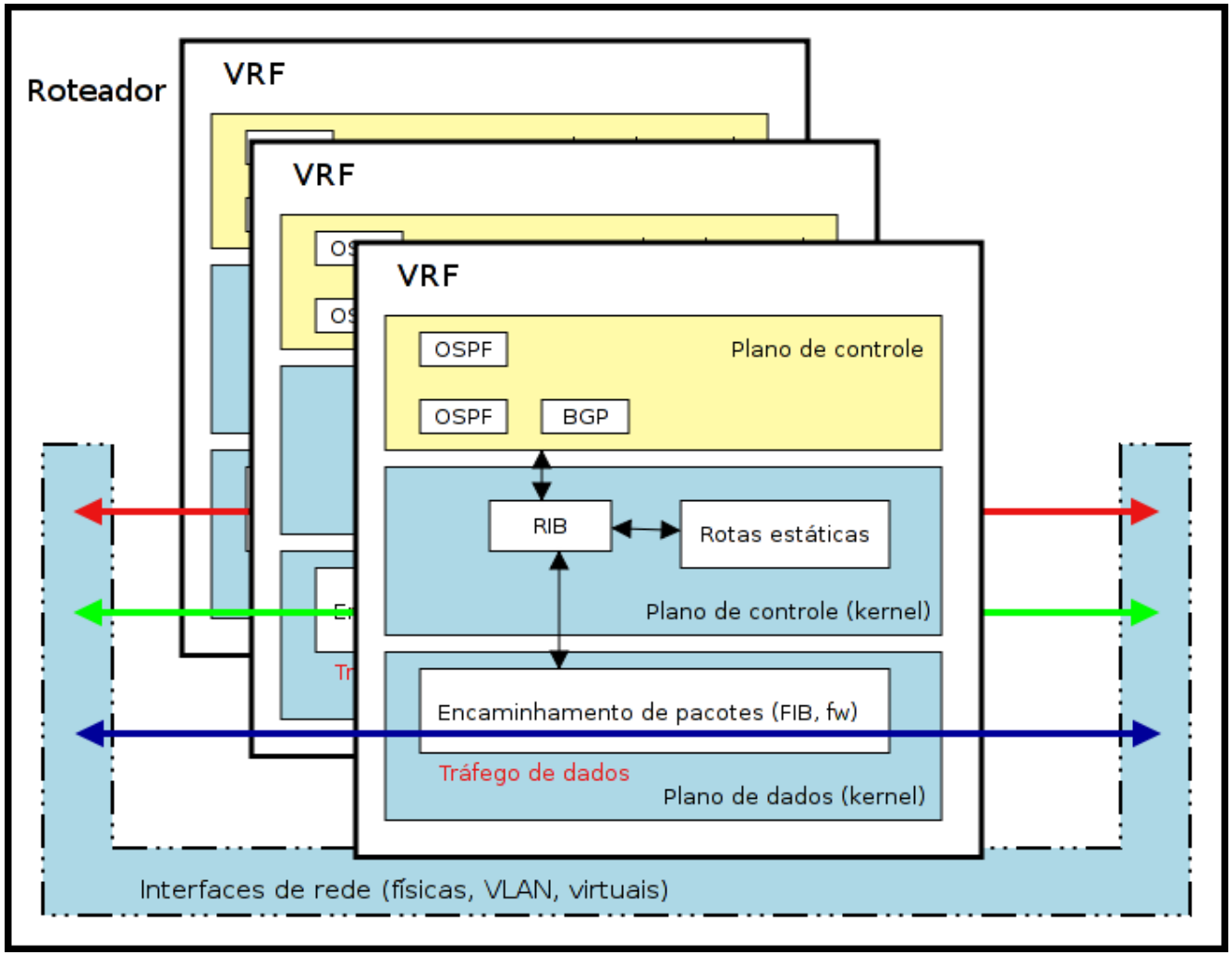


- Virtual é só uma “*buzzword*”: roteador por *software*, dentro de uma VM ou direto no hypervisor
- Não é o foco desta apresentação

VR: múltiplos planos de controle



VRF: múltiplas RIBs e FIBs



Routing engine: Linux

- Capacidade e performance variam muito com a versão do kernel
- Muita atividade nesta área no momento:
  - Tentativa de remoção do *route cache*
  - Aumento na performance e escalabilidade
  - Particionamento (*containers*)
  - Preço a pagar: *bugs!*
- **Requer** configuração otimizada para roteamento para melhor performance

FIB e RIB em Linux

- A RIB do kernel é efetivamente a FIB quando não existe algum ASIC de roteamento
  - **Infelizmente** este é o único caso de interesse prático para ISPs no momento...
- Aplicações de roteamento (Quagga, BIRD, etc) implementam suas próprias RIBs, e a RIB do kernel funciona como a FIB do roteador
- Para evitar confusão, nesta apresentação chamaremos a RIB/FIB do kernel de FIB, e as RIBs das aplicações de roteamento, de RIB

Network Namespaces (NETNS/NNS)

- Particionamento de **todo** o *network stack (VRF):*
  - *FIB, policy-routing, iptables, interfaces...*
- Criado para virtualização leve, baseada em *containers*. O objetivo principal é o isolamento de rede completo do *container*
- Menor perda de performance se comparado com virtualização completa (KVM, Xen) e UML
- Independente de outros isolamentos: não exige isolamento das PID, sistemas de arquivo...

Routing engine: v2.6.32

- Versão mais estável, indicada para roteamento “simples”:
  - Apenas uma FIB (mas possui policy routing)
  - Kernel LTS upstream
  - Sem surpresas, performance aceitável em caixas SMP/NUMA “pequenas”
  - Não é muito interessante (no sentido das antigas maldições chinesas)
- Não indicado para VRF, falta funcionalidade e tem problemas de escalabilidade na procfs

Routing engine: v3.0 / v3.1

- *May you live in interesting times*
- Versão LTS ainda não definida (3.0? 3.1? 3.2?)
- *Network namespaces* “atualizado”: VRF
  - Múltiplas FIBs, completamente isoladas
  - Completo isolamento das interfaces (física ou VLAN)
  - Escalável (*cgroups* não implodem a procfs)
  - NNS ancorável (via *bind mount*): existência independente do processo criador

Routing engine: utilitários *userspace*

- *iproute*
  - Versão v3.0 compilada com *kernel headers* ≥3.0
  - “ip netns ...”
- *Linux containers (lxc)*:
  - Solução de virtualização leve, suporte completo a *containers*
  - Use versão mais recente que v0.7.5
- Libvirt
  - Solução de virtualização pesada, mas tem algum suporte a *containers*

Performance: nada é de graça...

- Elevado PPS somente com muitos *cores,* com placas-mãe otimizadas para IO PCIe, e restrito a alguns NICs (por razões de driver e de hardware)
- *Tunning* do kernel e da plataforma é absolutamente indispensável
- Necessário restringir funcionalidade que interfere com o *fast-path*:
  - *ebtables*
  - *iptables*
  - *conntrack...*

Administração: nada é de graça...

- Nenhum suporte “nativo” a VRF nas *suites* de roteamento: Quagga, BIRD...
- Utilizando lxc e *containers* para isolamento total:
  - Fácil de entender: são VMs
  - Duplicação de trabalho, atualizações, etc (usar *cfengine, fai, puppet*, etc)
- Utilizando ip netns para isolamento parcial:
  - Mais complexo para entender e documentar
  - Visualização não é óbvia (administração difícil)

Criando um NNS com lxc

- Para kernel 3.x, recomendada versão da *suite lxc* mais recente que v0.7.5
- “lxc-unshare -s NETWORK /usr/bin/screen”
- A PID do processo é o ID do NNS
- Somente filhos do processo criador podem usar aquele NNS (já resolvido via /proc/<pid>/ns/net, mas *userspace* ainda não disponível)
- O NNS é destruído quando o processo que o criou é encerrado, a menos que esteja ancorado. Interfaces de rede são movidas para o NNS geral

Ancorando NNS

- *Namespaces* podem ser ancorados:
  - Abrindo o arquivo /proc/<pid>/ns/net e mantendo o mesmo aberto, ou
  - Via bind-mount (usado pelo iproute)
- Necessário, a menos que a aplicação seja capaz de configurar *completamente* o NNS
- Ideal para implementar VRF sem utilizar VMs (evitar isolamento da *filesystem,* PIDs, etc)

Criando um NNS ancorado com iproute

“ip netns add nns1”

“ip netns exec nns1 ip link set dev lo up”

“ip link add name veth10 type veth peer name veth11 netns nns1”

…

“ip netns exec nns1 /etc/init.d/bird_nns1 start”

Usando NNS

- Utilize *containers* completos como se fossem VMs, usando *libvirt* ou *lxc*
- Ou use *lxc-unshare* para rodar um script de longa duração (complexo)
- Ou use *ip netns* e scripts (fácil devido à ancoragem)
- Para o futuro:
  - Tentar adicionar suporte nativo a NNS e usá-lo para implementar VRF no BIRD e Quagga
  - Votar para que Vyatta implemente VRF...

Conectando NNSes

- Utilizaremos um par de dispositivos veth
  - Enlace “ethernet” virtual ponto a ponto
- “ip link add name veth00 type veth peer name veth01”
  - Cria veth00 ligado a veth01
- “ip link add name veth00 type veth peer name veth01 netns <nome/id>”
  - Cria par, e coloca veth01 em outro NNS

Conectando NNSes (2)

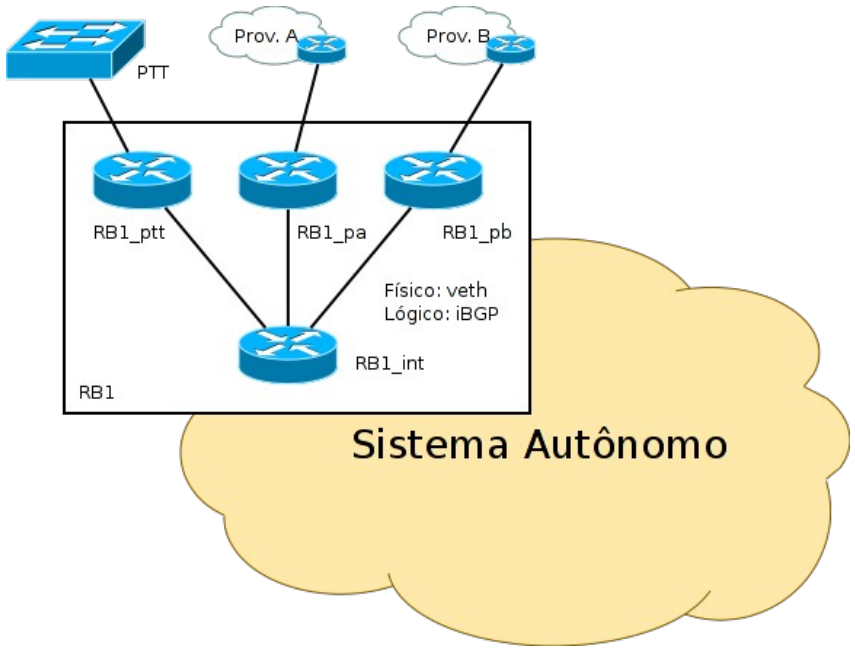
- “ip link set <dev> netns <id>” move dispositivo deste NNS para outro NNS
- Especifique o MTU de cada ponta se não quiser o padrão (1500 bytes)
- Especifique os MACs se não quiser que eles variem cada vez que o par de dispositivos for criado
- “ip link add name veth40 mtu 3500 address 12:89:bd:e5:cf:01 type veth peer name veth41 address 12:89:bd:e5:cf:02 mtu 3500 netns exemplo”

Dispositivos tipo *veth*

- Enlace *ethernet* virtual ponto a ponto
  - Funcionamento equivalente a dois NICs interligados diretamente via cabo
  - Permite VLANs, bonding, bridges...
- Criado aos pares: ip link add … type veth peer …
- MTUs **independentes**, máximo 16436
- Performance escalável por core, mas inferior a da *loopback*
- As pontas de um enlace *veth* podem estar em *network namespaces* diferentes

VRF: roteamento de borda

- Única caixa com 4 instâncias VRF
- uRPF pode ser utilizado
- ACLs/*firewalls* mais simples (em cada VRF)
- Muito mais simples de entender e gerenciar do ponto de vista do NOC
- **Menos resiliente que 4 caixas distintas**



Outras formas de conectar NNSes

- MACVLAN (conexão multiponto entre NNSes)
- OpenVZ (venet/vbus) – ainda não aceito no kernel *upstream,* mas disponível e suportado em Debian
- Externo à caixa:
  - NIC SR-IOV ligado em *switch* 1GbE/10GbE
- Assunto para discussões futuras e testes

Referências

- LWN, *Namespace file descriptors*, https://lwn.net/Articles/407495/
- Kadlecsik, Pásztor, *Netfilter performance testing*, http://people.netfilter.org/kadlec/nftest.pdf
- David Lamparter, vrf-tools
- Elio, D, *Linux Containers and Networking,* http://blog.flameeyes.eu/2010/09/04/linux-containers-and-networking

Hack: setns no iproute (glibc velha)

```
--- iproute-20111117.orig/ip/ipnetns.c
+++ iproute-20111117/ip/ipnetns.c
@@ -31,6 +31,19 @@
 #ifndef HAVE_SETNS
 static int setns(int fd, int nstype)
 {
+#ifndef __NR_setns
+#  warning please update your kernel headers, this is a major kludge
+#  if defined(__x86_64__)
+#    define __NR_setns 308
+#  elif defined(__i386__)
+#    define __NR_setns 346
+#  elif defined(__arm__)
+#    define __NR_setns 375
+#  warning assuming ARM EABI for setns() call
+#  else
+#    error setns syscall number not known
+#  endif
+#endif
 #ifdef __NR_setns
       return syscall(__NR_setns, fd, nstype);
 #else
```