FreeBSD HA: Alta disponibilidade com CARP, Ifstated e pfsync

Autoria de Daniel Bristot de Oliveira
11/08/2006
Última Atualização 13/08/2006

Dando continuiade ao artigo sobre Balanceamento de carga e redundância para Servidores Web publicado aqui no FUG, este artigo irá abordar três ferramentas muito utilizadas nesta abordagem: A primeira ferramenta é o CARP, o Protocolo de Redundância de Endereço Comum, que surgiu para ser uma alternativa livre a o VRRP, no final da seção do carp(4) ainda é mostrado como funciona o balanceamento de carga em nivel ARP que o carp(4) oferece. Outro assunto muito discutido é o monitoramento de interfaces de rede, isto é, um vigia para fazer alguma ação caso algum evento associado a uma interface aconteça, como executar um script cada vez que uma interface se torne inativa, isto é feito utilizando o port ifstated(8). Ao final, o assunto abordado é a sincronização de tabela de estados de conexões, algo fundamental para fazer a redundância de roteadores e firewalls transparente para os clientes, isto é feito com o auxílio da interface pfsync(4).

O CARP

O CARP é o substituto para o VRRP, o CARP - Protocolo de Redundância de Endereço Comum (Common Address Redundancy Protocol) - Ele é um protocolo nativo do OpenBSD, e foi portado para o FreeBSD.

O CARP surgiu para substituir o VRRP, pois o Protocolo VRRP é um padrão do IETF e é patenteado pela CISCO, com isso o Projeto OpenBSD, seguindo sua ideologia, e como já feito com a criação do PF, pois o IPFilter não era um totalmente livre, criou um padrão livre, o CARP.

O Objetivo do CARP é o mesmo do VRRP, garantir a redundância de hosts, fazendo com que vários hosts sejam responsáveis por um único endereço em uma rede, caso um falhe, o outro assume. Porém o CARP age de forma diferente do VRRP, o VRRP cria um endereço virtual vinculado a uma interface de rede, como o criado com o parâmetro "ifconfig_interface_alias0" do rc.conf(5), assim os hosts pertencentes ao mesmo grupo compartilha somente o mesmo IP, desta maneira, cada vez que um host muda de estado, todo o cache ARP da rede deve ser atualizado, Já o CARP trabalha com uma interface virtual, nomeada carpN, onde N é o numero da interface, que se vincula a uma interface real, a interface virtual tem um endereço MAC virtual, e todos os hosts que estão compartilhando o mesmo endereço também, assim quando um host muda de estado, o cache não precisa ser atualizado, pois o novo host mestre irá assumir não somente o endereço de IP mas também o endereço MAC. O problema é que para utilizar o CARP, necessita de no mínimo 3 IPs na mesma rede para funcionar, o que não é necessário no VRRP.Configurando o CARP

Agora iremos configurar o CARP, seguindo o nosso esquema do artigo sobre HA. A única alteração é a adição de mais dois endereços de IP nos roteadores, que serão: 200.18.15.16 Que será o endereço da interface carp0 que escutará em fxp0, que é a interface ligada a Internet; 10.0.70.3 Que será o endereço do carp1 que escutará em fxp1, a interface ligada a rede local;Configurando o Sistema Operacional

Para utilizarmos o CARP, devemos adicionar o suporte a interface carp ao kernel, para isto, adicione a seguinte linha ao arquivo de configuração do kernel: device carpConfigurando a interface carp(4)

Como o carp(4) trabalha com o conceito de interface, ele não possui uma arquivo específico de configuração, pois sua configuração é feita como a de qualquer dispositivo de rede, utilizando o ifconfig. A sintax do ifconfig(8) para os dispositivos carp(4) é:

Para criar uma nova interface: ifconfig carpN create

Para configurar uma interface é: ifconfig carpN vhid ID [pass senha] [carpdev carpdev] [advbase advbase] [advskew advskew] endereço-ip [máscara]

OBS: Os campos fora de couchetes são obrigatórios.

Explicação das opções: carpN O dispositivo carp, onde N é o número do dispositivo create Cria um novo dispositivo vhid Virtual Host Id, a identificação do grupo em que a interface faz parte, ele deve ser único, e todos os hosts que fazem compartilham o mesmo endereço devem ter o vhid iguais. ele é equivalente a opção serverid = 1 do VRRP, o valor aceito é de 1 à 255. pass É a senha de autenticação que pode ser utilizada entre os membros de um grupo, todos os membros devem ter a mesma senha. carpdev O dispositivo que o carp(4) escutará as conexões, caso ele não seja informado, o carp irá utilzar o endereço de IP e máscara para definir em qual interface escutar. advbase Este é um parâmetro que especifica a freqüência, em segundos, em que os anúncios sobre status são enviados, um valor baixo, significa baixo tempo de resposta a uma queda do roteador mestre. advskew Com este parâmetro é definido quem é o host Mestre, quanto menor o valor maior a prioridade, quem tiver o menor advskew será o mestre, e ficará anunciado isto para a

rede, caso ele "cair", o outro host do grupo com menor advskew assumirá seu lugar, os valores aceitos são entre 0 e 254, se nenhum valor for atribuído, ele pegará o padrão 0 e será o host mestre. endereço-ip Este é o endereço de IP que será compartilhado por todos os hosts do mesmo grupo, este endereço não precisa estar na mesma sub-rede da interface real, porém ele deve ser o mesmo em todas as hosts do grupo máscara Especifica a mascara de sub-rede.

Além destas opções existem algumas sysctls net.inet.carp.allow: 1 Aceita pacotes CARP net.inet.carp.preempt: 1 Ativa preemptivismo, isso faz com que a maior prioridade sempre assumirá o estado de Mestre, por exemplo se o menor cair, um BACKUP assumir ele será o mestre, se o preemptvismo estiver ligado, e o mestre voltar o mestre assumirá novamente, caso não, ele ficará de BACKUP. net.inet.carp.log: 1 Ativa o log. net.inet.carp.arpbalance: 1 Ativa o balanceamento de carga no em nível de ARP, Veremos abaixo como faze isto. net.inet.carp.suppress_preempt: 0 Esta sysctl é "Somente leitura" e serve para contabilizar erros de preempção.Exemplo

Como exemplo iremos configurar os nossos dois roteadores, primeiramente iremos configurar o mestre, que terá as duas interfaces, como mostra a saída do ifconfig(8) abaixo: fxp0:
flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
options=8<VLAN_MTU>
inet6 fe80::202:a5ff:fea0:3c1b%fxp0 prefixlen 64 scopeid 0x1
inet 200.18.15.14 netmask 0xff000000 broadcast 200.18.15.255
ether 00:02:a5:a0:3c:1b
media: Ethernet autoselect (100baseTX <full-duplex>)
status: active
fxp1: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
options=8<VLAN_MTU>
inet6 fe80::202:a5ff:fea0:3c1b%fxp0 prefixlen 64 scopeid 0x1
inet 10.0.70.1 netmask 0xff000000 broadcast 10.255.255.255
ether 00:02:a5:a0:3c:1c
media: Ethernet autoselect (100baseTX <full-duplex>)
status: active

Para nosso exemplo precisaremos de duas interfaces CARP, uma para a fxp0, que terá o endereço 200.18.15.16, e outra para fxp1, que terá o endereço 10.0.70.3. As interfaces no nó mestre terão a prioridade mais alta, para que sejam escolhidas como mestre, então iremos dar a elas o valor mínimo de advskew, já para as interfaces do nó escravo daremos uma prioridade mais baixa, então iremos dar um valor alto de advskew, por exemplo 100. Veremos abaixo como serão criadas as interfaces.

Criando as interfaces, Antes de configurarmos as interfaces CARP, devemos cria-las, elas podem ser criadas de duas maneiras, através do console, com o comando ifconfig(8) carpN create, ou na inicialização do sistema, com a opção cloned_interfaces no rc.conf, veremos as duas opçoes abaixo.

Criando a partir do console: # ifconfig carp0 create
ifconfig carp1 create

A partir do rc.conf cloned_interfaces="carp0 carp1"

Pronto as interfaces já estão criadas, agora vamos configura-las.

Aqui também é possível cria-las das duas formas, eu irei colocar apenas o exemplo do host mestre, porém a única diferença entre o mestre e o escravo é o valor de advskew, que no mestre será 1 e no escravo será 100.

vamos configurar.

Via console: # ifconfig carp0 vhid 1 advskew 1 pass senha1 200.18.15.16
ifconfig carp1 vhid 2 advskew 1 pass senha2 10.0.70.3

Via rc.conf: ifconfig_carp0="vhid 1 advskew 1 pass senha1 200.18.15.16"
ifconfig_carp1="vhid 2 advskew 1 pass senha2 10.0.70.3"

Pronto as interfaces estão configuradas.Como o CARP trabalha

O host mestre envia regularmente, via broadcast, anúncios sobre o seu estado, assim os hosts de backup ficam sabendo quem é o host mestre. Se os hosts de backup não ouvirem mais estes anúncios do nó mestre, o host de maior prioridade, ou seja, menor advskew, caso todos estejam utilizando preemptvismo, ou o host que responder primeiro, provavelmente o que tiver menor advbase, caso não estejam utilizando preemptivismo, irá assumir o status de mestre.

Estes anúncios são enviados utilizando o protocolo carp(4), e podemos ver os pacotes com o tcpdump(8), com o seguinte comando. # tcpdump proto carp

Se você está utilizando um firewall, você deve liberar o trafego para este protocolo, por exemplo com o PF seria:
pass out on $carp_dev proto carp keep state

Onde $carp_dev deve ser a interface física que o carp(4) está utilizando.Balanceamento de carga com carp(4)

Com o carp(4) também é possível fazermos balanceamento de carga, em nível ARP, isto é, o balanceamento será feito com o endereçamento ARP. Como sabemos o host mestre é definido por quem envia os anúncios para rede, então se fizermos isto em dois roteadores:

Roteador 1: Uma interface no grupo 1 com advskew baixo Uma interface no grupo 2 com advskew alto

Roteador 2: Uma interface no grupo 1 com advskew alto Uma interface no grupo 2 com advskew baixo

O que irá acontecer? o mestre do grupo 1 será o Roteador 1 e o mestre do grupo 2 será o roteador 2, e se estes dois grupos utilizassem o mesmo endereço de IP? teriamos dois mestres anunciando, mas aconteceria algo de errado, os dois aceitariam os pacotes, porém para resolver isto, se habilitarmos a sysctl(8) net.inet.carp.arpbalance, os roteadores utilizarão o endereço de IP do cliente para definir se ele aceita o pacote ou descarta, assim, o balanceamento é feito.

Veja o exemplo de configuração abaixo:

Roteador1 # ifconfig carp0 vhid 1 pass senha 10.0.60.61
ifconfig carp1 vhid 1 pass senha advskew 100 10.0.60.61

Roteador2 # ifconfig carp0 vhid 1 pass senha advskew 10010.0.60.61
ifconfig carp1 vhid 1 pass senha 10.0.60.61

Vejamos parte da saída do ifconfig dos dois roteadores. Roteador1 carp0: flags=49<UP,LOOPBACK,RUNNING> mtu 1500
	inet 10.0.60.61 netmask 0xff000000
	carp: MASTER vhid 1 advbase 1 advskew 0
carp1: flags=49<UP,LOOPBACK,RUNNING> mtu 1500
	inet 10.0.60.61 netmask 0xff000000
	carp: BACKUP vhid 2 advbase 1 advskew 100

Roteador2 carp0: flags=49<UP,LOOPBACK,RUNNING> mtu 1500
	inet 10.0.60.61 netmask 0xff000000
	carp: MASTER vhid 1 advbase 1 advskew 100
carp1: flags=49<UP,LOOPBACK,RUNNING> mtu 1500
	inet 10.0.60.61 netmask 0xff000000
	carp: BACKUP vhid 2 advbase 1 advskew 0

Agora precisamos habilitar a sysctl(8) net.inet.carp.arpbalance nos dois roteadores: # sysctl net.inet.carp.arpbalance=1

O Balanceamento está configurado. Vamos ver a saída do tcpdump(8). # tcpdump proto carp
[...]
15:09:56.674705 IP 10.0.60.2 > VRRP.MCAST.NET: VRRPv2, Advertisement, vrid 1, prio 0, authtype none, intvl 1s, length 36
15:09:57.104002 IP 10.0.60.4 > VRRP.MCAST.NET: VRRPv2, Advertisement, vrid 2, prio 0, authtype none, intvl 1s, length 36
15:09:57.675689 IP 10.0.60.2 > VRRP.MCAST.NET: VRRPv2, Advertisement, vrid 1, prio 0, authtype none, intvl 1s,

length 36
15:09:58.104852 IP 10.0.60.4 > VRRP.MCAST.NET: VRRPv2, Advertisement, vrid 2, prio 0, authtype none, intvl 1s, length 36

Como podemos ver os dois hosts, 10.0.60.2 e 10.0.60.4, estão anunciado, com priordade 0 e com seus distintos vhid's.

ATENÇÂO: O balanceamento de carga só funciona no segmento local, e não terá efeito em balanceamento onde todo o tráfego passa por apenas um roteador , pois o endereço do roteador sempre fará com o que o mesmo host responda a requisição, assim ele é inviável para utilizarmos nos servidores Web do artigo sobre "balanceamento de carga em servidores Web"O problema da dependência de interfaces

Porém nos surge um problema, se só uma interface de nossos roteadores cair? a outra continuaria UP, e isso quebraria a rede certo? Bom no VRRP nós temos a opção vridsdep, que cria uma dependência entre interfaces, assim, as duas estarão UP, se as duas estiverem UP, caso uma caia, o grupo inteiro cai também. Para resolver este problemas surge o ifstated(8).ifstated(8)

O ifstated(8) é um daemon de monitoramento de interfaces, com ele é possível examinar as interfaces e executar ações caso algum evento aconteça, em nosso caso ficaremos examinado as interfaces fxp0 e fxp1. O ifstated está disponível no ports(8) net/ifstated.

O arquivo de configuração tem uma sintaxe um pouco confusa no início, mas ele é extremamente poderoso. O arquivo de configuração fica em /usr/local/etc/ifstated.conf .

Neste artigo vou apenas dar uma pincelada no ifstated.conf(5), quem tiver maior interesse nele, aconselho a ler o man. ifstated.conf

Este arquivo de configuração se parece um pouco com programação, então eu farei uma analogia ao C.

O arquivo de configuração é dividido em três seções,

A primeira seção é denominada global, onde vão as opções para o início do sistema, ele apenas aceita a configuração de log e o estado inicial(Estado Inicial? espere um pouco que você vai intender). Aquí seria como dizer para o ifstated(8) quem é o void() do programa.

A segunda seção é chamada Macros, nela são criadas a variáveis do usuário, em uma analogia, seria as variáveis globais do programa.

A terceira e última seção é chamada Definições de estados, em uma analogia, estes estados são como as funções do programa (Agora você intendeu o que era aquele estado inicial), nele são feito testes com variáves, executados comandos. etc.

Bom vamos ver o exemplo de configuração que fiz para monitorar as interfaces, com ele fica mais fácil de intender. init-state one

```
if_up="fxp0.link.up && fxp1.link.up "

state one {
	if ( ! $if_up ) {
		run "ifconfig carp0 advskew 200"
		run "ifconfig carp1 advskew 200"
		set-state two
	}
}

state two {
	if ( $if_up ) {
		run "ifconfig carp0 advskew 1"
		run "ifconfig carp1 advskew 1"
		set-state one

	}
```

}

Vamos examina-lo:

A primeira linha é a seção global, nela apenas definimos que o estado inicial é o estado one.

A segunda linha é uma variável, que verifica se a interface fxp1 e a fxp0 estão up, caso estejam, a variável se torna verdadeira, caso uma não esteja, ela se torna falsa. Este teste pode ser feito com qualquer interface, a sintax é Interface.link.Estado, e os estados das interfaces podem ser: up : Ativa, ou para o carp(4) MASTER unknown: Desconhecido. ou para o carp(4) INIT down: Desativo, ou para carp(4) BACKUP

Existem outros testes, que fogem o escopo deste artigo, para maiores informações, leia o man 5 ifstated.conf

A terceira linha podemos ver o início de um bloco, isto é um estado, ou como na analogia, uma função. O estado one é o inicial, ele checa: Se a $if_up for falso, isto é se alguma iterface não está ativa, ele aumenta o advskew, através da a opção run executa o comando que lhe foi passado, assim fazendo com que as interface carp que ainda está ativa vá pra o estado de BACKUP. A última linha muda para o estado two, em uma analogia, seria a chamada de uma outra função.

O estato two faz o contrário do estado one, as interfaces aquí chegarão como BACKUP, então ele fica checando se: $if_up for verdade, isto é as duas interfaces estiverem ativas, ele diminui os advskew das interfaces carp, assim tornando elas interfaces MESTRE novamente.

ATENÇÂO: Neste exemplo estamos trabalhando apenas com dois roteaores, o ifstated irá executar apenas no servidor mestre, pois de nada adianta colocarmos ele no servidor escravo, se o escravo também cair a rede vai cair, se tivessemos mais um roteador, os dois primeiros executarão o ifstate e o último não. Em nosso exemplo, o que devemos prestar atênção é: o advskew do escravo deve estar entre o advskew menor e o maior do nó mestre, no nó escravo estamos utilizando o advskew como 100, se o mestre está UP ele terá o advskew de 1 e o mestre estará UP, caso alguma interface do mestre cair, o seu advskew será de 200, assim o escravo se tornará o mestre, pois sua prioridade é maior.

ATENÇÂO 2: Para isto funcionar, devemos habilitar a preempção no carp, para isto mude o valor da sysctl net.inet.carp.preempt para 1, isto pode ser feito adicionando o a seguinte linha no /etc/sysctl.conf. net.inet.carp.preempt=1Executando o ifstated

O ifstated pode ser executado de duas formas, manualmente, executando ifstated, ou durante a inicialização do sistema, pra isto adicione a linha abaixo no /etc/rc.conf: ifstated_enable="YES"

Também é possível executar o ifstated(8) em primeiro plano, e em modo debug, isto é ótimo para analisar se o arquivo de configuração está trabalhando corretamente. Para isto execute ifstated(8) manualmente com as flags -d para ficar em primeiro plano, -v para modo verbose e -f para especificar outro arquivo de configuração, como no exemplo abaixo. # ifstated -dv -f /home/usuario/ifstated.conf.teste
Pfsync

Um tema que não abordei na primeira parte do artigo, foi o firewall. Como estes roteadores estarão de frente para a Internet, é fundamental a presensa de um firewall.

Bom, como todos podemos ver o firewall ou Packet Filter que mais vem se destacando é o Pacaket Filter do Projeto OpenBSD, ele trás várias vantagens como listas e tabelas, integra várias tecnologias com QoS e NAT, e tem uma sintaxe limpa e clara. Ele surgiu pelo mesmo motivo do CARP, substituir um software protegido por patente e direitos autorais e patente.

Neste artigo não abordarei como fazer um firewall, mas sim uma das propriedades mais interessantes dele, o pfsync(4).

Neste artigo estamos trabalhando com dois firewalls, um em cada roteador, suponhamos que um cliente está em uma tranzação que está passando pelo nosso roteador, e o nosso roteador cair, bom, o roteador de backup irá assumir o seu lugar, só que irá acontecer um problema, o firewall so backup não irá reconhecer a conexão já existente e irá trancar o cliente. Para resolver esta situação surgiu o pfsync(4). Ele irá sincronizar as tabelas de estados de conexões entre os firewall de uma rede.

O pfsync(4) é como o carp(4), uma interface de rede, que é utilizada para enviar e receber, via multicast ou em unicast com o auxílio do ipsec(4), informações sobre estados de conexões. Vamos ver como ela funciona. O sistema operacional

Para utilizarmos os pfsync(4) devemos ter suporte ao PF e a interface pfsyc, para isto adicione as seguintes linhas no arquivo de configuração do seu kernel. device pf
device pfsync

Compile e instale o seu kernel.

Após reiniciar o seu novo kernel, você verá que aparecerá mais uma interface a saída do ifconfig(8), como no exemplo abaixo. pfsync0: flags=41<UP,RUNNING> mtu 1348
pfsync: syncdev: fxp0 maxupd: 128Configurando o pfsync(8)

Como o pfsync é uma interface virtual, a sua configuração é feita através do ifconfig(8), a sintaxe de configuração é: ifconfig pfsyncN syncdev dispositivo [syncpeer EndereçoDeIp]
pfsyncN O nome da interface pfsync(4) syncdev O nome da interface fisica que o pfsync utilizará para enviar as mensagens. syncpeer Este parâmetro opcional específica o endereço IP de um host para trocar atualizações pfsync. Por padrão atualizações pfsync são multicast na rede local. Esta opção cancela este comportamento e em vez disso envia atualizações unicasts para o syncpeer especificado.

O método mais correto e eficiênte de fazer estas atualizações, seria adicionarmos mais um interface aos nossos roteadores, ligar os dois via cabo crossover e configurar o pfsync para estas interfaces, porém nós só estamos utilizando duas interfaces, então iremos utilizar uma delas.

A nossa interface fxp1 está em um ambiênte seguro que é nossa rede, então poderemos utiliza-la.Configurando o pfsync(8)

Veja o exemplo abaixo: # ifconfig pfsync0 sysncdev fxp1

Nossas interfaces estão conectadas.Como o pfsync(8) funciona

O pfsync envia suas mensagens, em modo multicast utilizando o protocolo pfsync, com o auxílio do tcpdump(8) podemos ver o tráfego dos pacotes co pfsync, como no exemplo abaixo. # tcpdump proto pfsync
[...]
10:26:19.309892 IP dcc.unesc.net > 224.0.0.240: pfsync 20
10:26:24.309820 IP dcc.unesc.net > 224.0.0.240: pfsync 20
10:26:29.309737 IP dcc.unesc.net > 224.0.0.240: pfsync 20

Não esqueca de abrir o firewall para o pfsync, no PF seria assim: pass on $sync_if proto pfsync

Onde $sync_if é a interface real que o pfsync está ligado.Considerações Finais

Chegou ao fim mais um artigo, este ficou um pouco menor que o artigo Monstro do GEOM, mas espero ter clareado a mente dos leitores sobre a utilização do CARP, do ifstated e do pfsync. Não posso deixar de agradecer ao Projeto OpenBSD por essas duas grandes contribuições. Um Obrigado também ao meu Mestre, ídolo e pai, Angelo de Oliveira.

Referências bibliográficas:

Failover Firewalls with OpenBSD and CARP. Disponível em: http://www.samag.com/documents/s=9658/sam0505e/

PF: Redundância de Firewall com CARP e pfsync Disponível em: http://www.openbsd.org/faq/pf/pt/carp.html

Páginas do Man

Até na próxima no fantástico mundo de Bob. Retirado de "http://dbristot.info/wiki/index.php?title=Carp"