FIREWALL EN ALTA DISPONIBILIDAD

Norberto Altalef
naltalef@redklee.com.ar

RedKlee - Argentina

1- Alcances

Se describirá en este documento las ventajas y la configuración de una estructura de firewall en alta disponibilidad, utilizando herramientas OpenSource.

2- Ventajas

Una estructura de firewall en alta disponibilidad presenta las siguientes ventajas:

- Continuidad de servicio frente a fallas de hardware.
- Posibilidad de usar hardware standard (PC clone) para funciones críticas.
- Permitir actualizaciones de software sin interrupción del servicio.
- Si además se puede contar con más de una conexión a internet, puede continuarse el servicio aún frente a una caída de uno de los enlaces.

Firewall stateful. Sincronización de estados de conexiones.

Un *firewall stateful* es aquel que lleva un control de las conexiones establecidas, de manera que puede determinar si un paquete pertenece o no a una conexión.
Si se determina que pertenece a una conexión, se lo acepta sin necesidad de validarlo contra las reglas. Esto brinda una notable mejora de performance.

Cuando se quiere establecer una configuración de alta disponibilidad, es necesario contar con un mecanismo para que el firewall que está inactivo (backup) tenga información de las conexiones ya establecidas.
Si esto no ocurre, en caso de falla del firewall principal cuando el de backup tome su lugar cancelará todas las conexiones ya establecidas.

3- Propuesta

La propuesta contemplará los siguientes aspectos:

1- Instalar una estructura de firewall con equipos redundantes.
2- Contar con al menos dos accesos internet, provistos por distintos proveedores (ISP).
3- Frente a una falla el usuario no debe tener ninguna interrupción de su conexión.
4- Posibilidad de usar los dos enlaces internet uno como activo y el otro como backup o tenerlos ambos activos.

4- Implementación

Para la estructura de firewall redundantes la propuesta es realizarlo con una instalación basada en el sistema operativo *OpenBSD.*

La decisión de usar *OpenBSD* se basa en varios aspectos:

- Sistema operativo muy estable y totalmente orientado desde su concepción hacia la seguridad.
- Software de filtrado de paquetes *pf*, muy sencillo de usar y con muchas funcionalidades integradas.
- Configuración de alta disponibilidad usando el protocolo *CARP*, estabilizada desde hace varias versiones.
- Pseudo-interface *pfsync* que permite configurar un firewall "stateful" en alta disponibilidad.
- Instalación muy sencilla, que ocupa poco espacio en disco y con requerimientos de hardware mínimos.

Ampliando los puntos anteriores:

1. ***OpenBSD*** es uno de los sistemas operativos más seguros y bien diseñados disponibles actualmente.
   Pese a que las raíces de este sistema operativo se remontan a los años 70 en los primeros días de *UNIX*, *OpenBSD* ha evolucionado siendo en la actualidad un sistema con un fuerte desarrollo y muy usado en gran cantidad de instalaciones.
   *BSD (**B**erkeley **S**oftware **D**istribution)* fue una derivación del *Unix* de *AT&T* a fines de los años 70. A partir de *BSD* a principios de los años 90 se derivaron *NetBSD* y *FreeBSD*.
   Theo de Raadt, que es todavía el lider del proyecto, se separó del proyecto *NetBSD* en Octubre de 1995 y liberó la versión 2.0 de *OpenBSD* a mediados de 1997.
   El objetivo del proyecto es disponer de un sistema operativo compacto, muy bien documentado y con extremo cuidado en temas de seguridad.

2. ***pf*** es el software de filtrado de paquetes (packet filter) de *OpenBSD*.
   Además de filtrar tráfico TCP/IP puede hacer también traslación de direcciones (*NAT - **N**etwork **A**ddress **T**raslation*), tambien incluye la función de normalización de tráfico TCP/IP, así como control de ancho de banda.
   La sintaxis es muy intuitiva y permite realizar firewall potentes con archivos de configuración sencillos.
   Así como todo el proyecto *OpenBSD*, tiene una excelente documentación.
   Está disponible en el kernel genérico desde el año 2001 (versión *OpenBSD 3.0*) por lo que es muy estable y probado. A través de los años se le han agregado muchas funcionalidades.

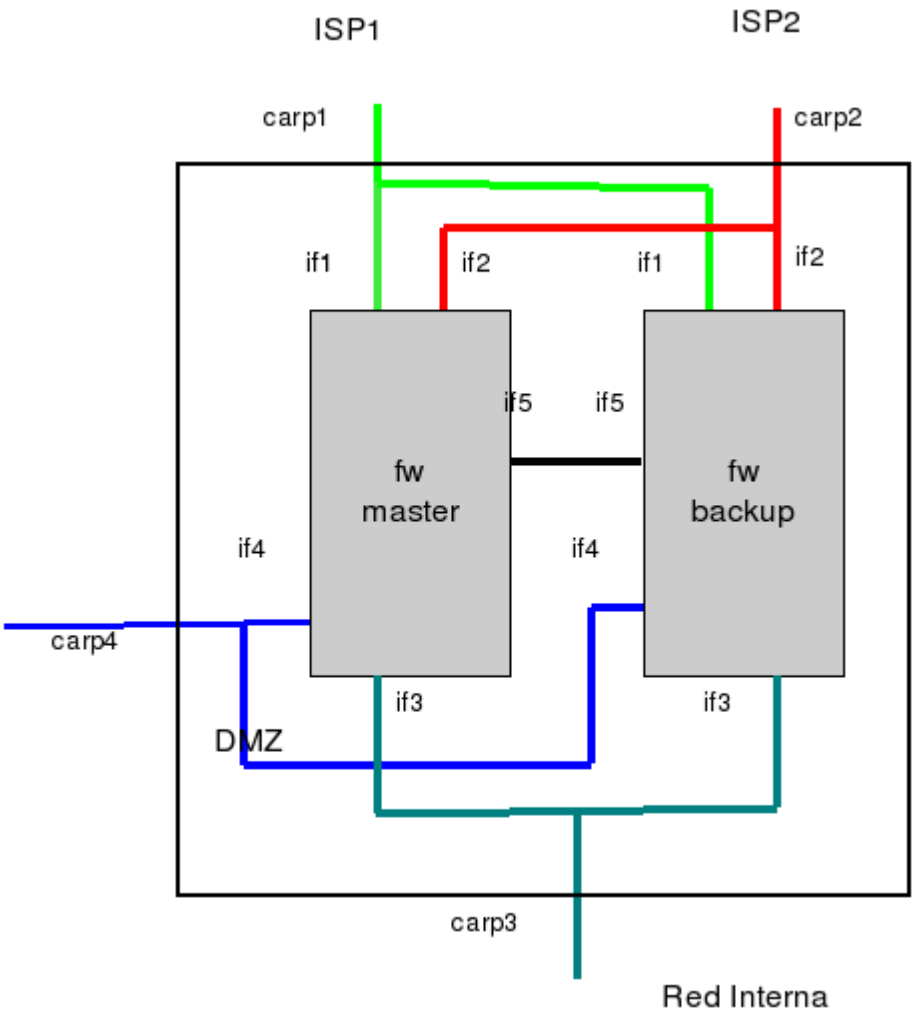
3. El propósito de ***CARP*** (***C**ommon **A**ddress **R**edundancy **P**rotocol*) es el de permitir que varios hosts en el mismo segmento de red, compartan una dirección IP.
   CARP es una alternativa gratis a *VRRP* (***V**irtual **R**outer **R**edundancy **P**rotocol*) y *HSRP (**H**ot **S**tandby **R**outer **P**rotocol)*
   El grupo de hosts que comparte una dirección IP se denomina "**grupo de redundancia**".
   Dentro de un grupo un host se designa como **master** y los demás como **backups**.
   El host master es quien responde al tráfico dirigido a la dirección IP compartida. En caso de falla del master, el host backup de mayor prioridad es quien pasa a tener identidad de master.

4. La interface virtual **pfsync** expone los cambios en la tabla de estados usada por *pf*.
   Si esta interface virtual se asocia a una interface física estos cambios de estado

pueden ser transmitidos a otros hosts, de manera que todos los hosts de un grupo puedan tener la información de la tabla de estados del host master.
En caso de tener que tomar su lugar las conexiones pueden seguir sin interrupción.
Esto permite configurar *firewalls stateful* en alta disponibilidad.

5. La instalación de OpenBSD que permite configurar todas estas opciones es muy rápida, ocupa muy poco espacio en disco y tiene bajos requerimientos de hardware.

Detalles de la implementación
En el siguiente gráfico se representan los esquemas de conexión de una instalación de un firewall que tiene:



- Dos equipos en alta disponibilidad
- Dos proveedores de internet
- Una zona desprotegida. EXTERNA
- Una zona desmilitarizada. DMZ
- Una zona protegida. INTERNA

En cada una de las interfaces, cada equipo tiene su propia dirección IP y a su vez hay una dirección IP de cada interface carp. Esta es la dirección IP que "ven" los demás equipos conectados.

Esto es:

if1= interface de cada equipo en la red del ISP1.
carp1 = interface del conjunto en la red del ISP1.

if2 = interface de cada equipo en la red del ISP2
carp2 = interface del conjunto en la red del ISP2

if3 = interface de cada equipo en la DMZ
carp3 = interface del conjunto en la DMZ

if4 = interface de cada equipo en la red interna
carp4 = interface del conjunto en la red interna

if5 = interface entre cada equipo

Las interfaces **if** tienen una IP distinta en cada equipo.
Las interfaces **carp** tienen una ip única.

Como decíamos antes, en esta configuración uno de los equipos se configura como **master** y es quien *"maneja"* la IP virtual de cada interface.

El otro equipo se configura como backup y solo empieza a funcionar ante la falla del master.

Se utiliza una interface que vincula ambos equipos, que para tener baja latencia y seguridad suele ser simplemente un cable de red cruzado, es por donde se transfieren los paquetes de **pfsync**, para mantener las tablas de estado sincronizadas.

¿Cómo se entera el host backup que el host master está activo?
La característica del protocolo *carp* es que los equipos que son master se anuncian como activos enviando a intervalos regulares paquetes de datos.

Para usar *carp*, se necesita configurar en cada interface de cada host un *Virtual Host ID (VHID)*, una dirección IP virtual y además dos parámetros que son **advbase** y **advskew**, que controlan cuan frecuentemente se envía un anuncio.

Cuando un host recibe anuncios con un advbase menor al propio, supone que otro host está actuando como master.
Cuando deja de recibirlos, comienza a enviar los anuncios, indicando que ahora ese host tiene la función de master.
Si se quieren tener varios host backup, se define la prioridad con que deben actuar como master asignando valores *crecientes* del parámetro *advbase*.

Todos los parámetros de configuración de *carp*, se asignan con el comando *ifconfig* que en este sistema operativo maneja todos los tipos de interfaces tanto sean reales como virtuales. Esto simplifica muchísimo la configuración.
Para más detalles ver: **man (8) ifconfig**
A modo de ejemplo, veamos la configuración de la interface con uno de los dos ISP.

En el equipo *master*:

```
ifconfig if1 201.202.203.204 netmask 255.255.255.0
ifconfig carp1 201.202.203.210 vhid 1 advsbase 0 pass 1234 carpdev if1
```

En el equipo *backup*:

```
ifconfig if1 201.202.203.205 netmask 255.255.255.0
ifconfig carp1 201.202.203.210 vhid 1 advsbase 10 pass 1234 carpdev if1
```

El significado de los parámetros es el siguiente:

- **if1:** nombre de la interface física (ver Nota)
- **carp1:** nombre de la interface virtual
- **201.202.203.204:** dirección IP del host master
- **201.202.203.205:** dirección IP del host backup
- **201.202.203.210:** dirección IP de la interface virtual
- **vhid:** virtual host id
- **advbase:** valor base que determina si el host es master o backup
- **pass:** password que se usa para la autenticación de los mensajes
- **carpdev:** interface física asociada

Nota: en OpenBSD a diferencia de Linux el nombre de una interface depende del driver del kernel que la maneje. O sea la primer interface no será eth0 siempre, sino que variará dependiendo de la marca y modelo de la tarjeta de red. (Ejemplo: **em0** si es una tarjeta Intel 10/100/1000, **xl0** si es una tarjeta 3COM 3c9xx).

Para que los comandos de configuración de cada interface se tomen en el boot, debe existir por cada una de las interfaces un archivo */etc/hostname.if,* donde se *deben* incluir los parámetros que recibe el comando *ifconfig*.
Se pueden incluir parámetros adicionales que fijen por ejemplo el modo de operación de la interface (100 MB FullDuplex, etc) y tambien ejecutar comandos en el momento de la inicialización de la interface.
Para más detalles ver: ***man (5) hostname.if***

firewall stateful y pfsync
Una de las grandes ventajas de utilizar OpenBSD, es que está resuelto en forma muy confiable y sencilla el mecanismo para que el o los firewall de backup tengan conocimiento de los estados establecidos en el firewall master.
Esto, como dijimos antes, permite configurar reglas que generen estado y que en caso de falla las conexiones establecidas no se vean afectadas.
Para esto se usa la pseudo-interface **pfsync**.
Puede fácilmente asociarse *pfsync* con una interface física.
Para que los firewall mantengan sincronizadas sus tablas de estado, será necesario solamente crear en cada equipo un archivo */etc/hostname.pfsync0,* con el siguiente contenido:

/etc/hostname.pfsync0:

```
up syncdev if5
```

if5 es el nombre de la interface que se dedica a esta sincronización.

¿Cómo manejar dos proveedores de internet?
Una alternativa es tener un proveedor principal y otro de backup.
Mientras el proveedor principal está activo se envía el tráfico por esta conexión y si se detecta una falla se comienza a enviar por el de backup.
Otra alternativa es que ambas conexiones estén activas simultáneamente. Si bien es perfectamente posible enviar el mismo tipo de tráfico por ambas interfaces, esto a veces trae inconvenientes con sitios seguros en los cuales se validan las direcciones IP de origen.
Una alternativa interesante es manejar por cada enlace distintos tipos de tráfico. Por ejemplo, tráfico de navegación (*http y https*) por un ISP y tráfico de mail y otros protocolos por otro ISP. En caso de falla de cualquiera de los enlaces, se puede manejar todo el tráfico por el restante.

¿Cómo puede hacerse este ruteo variable?
Todas estas funcionalidades pueden manejarse directamente con pocas reglas del sistema de filtrado de paquetes pf.
Esta capacidad del filtrado de paquetes de OpenBSD es lo que brinda muchas posibilidades de implementación.

Supongamos los siguientes requerimientos:

- Permitir desde la red interna acceso hacia internet de los puertos de *http* y *https*. Rutear este tráfico por el ISP1.
- Permitir desde la red interna acceso hacia internet de los puertos de *pop3* y *smtp*. Rutear este tráfico por el ISP2.
- No permitir tráfico iniciado en internet hacia la red interna.

Veamos algunas reglas para obtener esta funcionalidad.

```
# Variables
ext_if1 = "if1"                          # interface donde se conecta el ISP1
ext_if2 = "if2"                          # interface donde se conecta el ISP2
int_if = "if3"                           # interface donde se conecta la red interna
www_ports = "{" 80 443 "}"               # ports permitidos para navegación
mail_ports = "{" 25 110 "}"              # ports permitidos para mail

# NAT de cualquier server de la red interna con la IP de la interface externa
# correspondiente a cada ISP
nat on $ext_if1 from $int_if:network to any -> $ext_if1
nat on $ext_if2 from $int_if:network to any -> $ext_if2

# Inicialmente se bloquea todo el tráfico de entrada.
block in log

# Pese a que pf puede bloquear tanto el tráfico de entrada o de salida, es más
# simple filtrar en una dirección.
# Elegimos filtrar el tráfico de entrada. Una vez que se permite el tráfico de entrada
# en una interface, no se bloquea su salida. La opción keep state, indica que se
# genere un estado para las conexiones (firewall stateful)
```

pass out keep state

 # Aceptamos en la interface interna tráfico http y https y se rutea por ISP1
pass in on $int_if route-to ($ext_if1) inet proto tcp from $int_if:network to any port \ $www_ports keep state

 # Aceptamos en la interface interna tráfico smtp y pop3 y se rutea por ISP2
pass in on $int_if route-to ($ext_if2) inet proto tcp from $int_if:network to any port \ $mail_ports keep state

Se puede observar que con muy pocas reglas (6 en total) se obtiene la funcionalidad deseada.

¿Cómo detectar fallas de un ISP y como cambiar las reglas de filtrado?
Para detectar la caída de una conexión se dispone de **ifstated**. Este daemon puede monitorear las interfaces y a su vez ejecutar comandos externos para monitorear conexiones. Dependerá de cada tipo de conexión como se monitoree.
Una vez que se detectó la caída de un ISP, es necesario cambiar las reglas de filtrado para redireccionar el tráfico. Esto nuevamente puede hacerse utilizando una funcionalidad de pf, que en este caso es el concepto de **anchor**, que permite tener reglas definidas que pueden cargarse y descargarse en forma dinámica sin necesidad de hacerlo con el juego completo de reglas, ya que esto afectaría las conexiones existentes, que se están cursando por el ISP que no falló.

Veamos esto en más detalle:

El daemon *ifstated* puede ejecutar comandos en respuesta a cambios en el estado de las interfaces. El estado de las interfaces puede determinarse monitoreando el estado del link o ejecutando comandos externos.
Tiene un archivo de configuración que es */etc/ifstated.conf*

ifstated tiene definidos 3 estados posibles para un link, que son:

up: La interface física tiene conexión activa. Para una interface carp, representa que interface está como *master*.
down: La interface física no tiene conexión activa. Para una interface carp, representa que la interface está como *backup*.
unknown: No se puede determinar el estado de la conexón física. Para una interface carp esto representa el estado *init*.

En muchos casos esta información no es suficiente.
En nuestro caso en particular, perfectamente podemos tener link hacia el router que nos conecta con un ISP y este router no tener conexión hacia internet. Se necesita alguna forma externa de verificar esta conexión.
ifstated permite ejecutar comandos externos y tomar acciones verificando el status que estos comandos retornen.
Por ejemplo haciendo un ping de unos pocos paquetes hacia un server conocido, puede indicar si la conexión está activa.

Dentro de ifstated.conf, se pueden definir variables como las siguientes:

```
carp_up = "carp0.link.up && carp1.link.up"

net = '( "ping -q -c 1 -w 1 192.168.0.1 > /dev/null" every 10 && \
        "ping -q -c 1 -w 1 192.168.0.2 > /dev/null" every 10 )'
```

La variable **carp_up**, depende de los estados de dos interfaces carp. En este caso en que sea master en ambas interfaces.

La variable **net,** depende de la conectividad con dos IP. Para verificar esto se envía un ping con una frecuencia de 10 seg (palabra clave *every).*

Estados en ifstated
ifstated opera con estados y transiciones.

Cada estado tiene una bloque de inicio y un cuerpo.
El bloque de inicio, se utiliza para inicializar el estado y se ejecuta cada vez que se ingresa en este estado.
El cuerpo del estado solo se ejecuta cuando el estado es el actual y ocurre un evento.

Las acciones a tomar dentro de un cierto estado normalmente implican uno o más sentencias **if.** Las posibles acciones incluyen ejecutar un comando usando **run** o disparar una transición con **set-state**.

Como ejemplo mostramos del archivo de configuración usado en la implementación.

```
# Test de ISP1 e ISP2
ISP1 = '( "/etc/ifstated.scripts/test_isp isp1" every 30 )'
ISP2 = '( "/etc/ifstated.scripts/test_isp isp2" every 30 )'

# Ambos ISP activos
state bothup {
        init {
                run "/etc/ifstated.scripts/start"
        }
        if (! $ISP1 && $ISP2)
                set-state ISP1down
        if ! $ISP2 && $ISP1
                set-state ISP2down
        if ! ($ISP1 || $ISP2)
                set-state bothdown
}

# Falla ISP1
state ISP1down {
        init {
                run "/etc/ifstated.scripts/isp2"
        }
        if ($ISP1 && $ISP2)
                set-state bothup
```

```
        if ! ($ISP1 || $ISP2)
                set-state bothdown
        if ! $ISP2 && $ISP1
                set-state ISP2down
}

# Falla ISP2
state ISP2down {
        init {
                run "/etc/ifstated.scripts/isp1"
        }
        if ($ISP2 && $ISP1)
                set-state bothup
        if ! ($ISP1 || $ISP2)
                set-state bothdown
        if ! $ISP1 && $ISP2
                set-state ISP1down
}

# Ambos ISP con falla
state bothdown {
        init {
                run "/etc/ifstated.scripts/start"
        }
        if ($ISP2 && $ISP1)
                set-state bothup
        if ($ISP1 && ! $ISP2)
                set-state ISP2down
        if ($ISP2 && ! $ISP1)
                set-state ISP1down
}
```

En el script **test_isp** se ejecuta para cada ISP un ping a dos direcciones conocidas. Previamente se setearon las rutas necesarias.
Con que al menos una de las dos direcciones responda el ping, se considera que el ISP está activo.

En los scripts **isp1** e **isp2** se cambian las reglas de pf, de manera de enviar el tipo de tráfico normalmente enviado por el ISP que falló, por el que quedó activo.
Para evitar cargar todas las reglas nuevamente, se usa el concepto de **anchor** que tiene pf.

Más allá del juego principal de reglas, pf puede cargar reglas adicionales, en puntos denominados *anchors.*
Un *anchor* puede incluir reglas, variables e incluso otros anchor.
La ventaja de este esquema es que las reglas que tiene el anchor pueden estar fijas con anterioridad o modificarse en forma dinámica, pero en cualquier caso pueden cargarse y descargarse sin afectar las otras reglas, ni los estados ya establecidos.

Para el ejemplo anterior, podrían agregarse 2 anchor luego de las reglas de filtrado, de manera que quedaran como sigue:

```
# Aceptamos en la interface interna tráfico http y https y se rutea por ISP1
pass in on $int_if route-to ($ext_if1) inet proto tcp from $int_if:network to any port \
$www_ports keep state

# Aceptamos en la interface interna tráfico smtp y pop3 y se rutea por ISP2
pass in on $int_if route-to ($ext_if2) inet proto tcp from $int_if:network to any port \
$mail_ports keep state

# Anchor por falla de isp1
anchor isp1

# Anchor por falla de isp2
anchor isp2
```

En este caso tendríamos dos archivos, con las reglas que debe cargar cada anchor. Por ejemplo:

/**etc**/**anchor.isp1**, debería tener el siguiente contenido:

```
# Variables

ext_if1 = "if1"                          # interface donde se conecta el ISP1
ext_if2 = "if2"                          # interface donde se conecta el ISP2
int_if = "if3"                           # interface donde se conecta la red interna
www_ports = "{" 80 443 "}"               # ports permitidos para navegación
mail_ports = "{" 25 110 "}"              # ports permitidos para mail

# Aceptamos en la interface interna tráfico http y https y se rutea por ISP2
pass in on $int_if route-to ($ext_if2) inet proto tcp from $int_if:network to any port \
$www_ports keep state
```

/**etc**/**anchor.isp2**, debería tener el siguiente contenido:

```
# Variables

ext_if1 = "if1"                          # interface donde se conecta el ISP1
ext_if2 = "if2"                          # interface donde se conecta el ISP2
int_if = "if3"                           # interface donde se conecta la red interna
www_ports = "{" 80 443 "}"               # ports permitidos para navegación
mail_ports = "{" 25 110 "}"              # ports permitidos para mail

# Aceptamos en la interface interna tráfico smtp y pop3 y se rutea por ISP1
pass in on $int_if route-to ($ext_if1) inet proto tcp from $int_if:network to any port \
$mail_ports keep state
```

Un par de observaciones:

- En los anchor no se trasladan las definiciones de MACROS (variables) que se hayan hecho en la regla principal, por lo que debe ser incluídas en cada definición.

- En pf es la última regla que valide la que se toma en cuenta.

De esta manera los archivos */etc/ifstated.scripts/isp1 y /etc/ifstated.scripts/isp2,* que se ejecutan en /etc/ifstated.conf, tendrán las siguientes instrucciones para cambiar los reglas de filtrado:

isp1: *pfctl -a isp1 -f /etc/anchor.isp1*
isp2: *pfctl -a isp2 -f /etc/anchor.isp2*

A esto puede ser necesario cambiar rutas o algúna otra configuración asociada con un dado ISP.

El esquema es muy flexible y el mostrado es solo una de las opciones posibles.

Con este artículo se muestra las posibilidades de implementación de un firewall redundante, usando OpenBSD.
Como adicional puede configurarse el firewall como concentrador de VPNs usando IPSec. El sistema operativo tiene soporte IPSec nativo, usando el daemon **isakmpd** y el comando de configuración **ipsecctl.** Valga aclarar que tambien puede hacerse que este concentrador funcione en alta disponibilidad, ya que se dispone del daemon **sasyncd** que permite mantener la información de las VPNs establecidas en el firewall de backup.

Se recomienda consultar el sitio de OpenBSD (www.openbsd.org) donde se encontrará excelente documentación e incluso los manuales de los respectivos comandos y archivos de configuración.
Cabe aclarar que todo lo descripto ha sido probado en la práctica y está funcionando en la actualidad en más de una organización y ha mostrado sus capacidades reales.

Norberto Altalef
RedKlee

Soluciones basadas en
Software Libre
www.redklee.com.ar
naltalef@redklee.com.ar
Sanchez de Bustamante 635
Buenos Aires - Argentina
Tel: 5411-4863-3074 / Cel: 5411-15-5055-9853