Configuración de un punto de acceso WiFi en su nueva instalación de Slackware

Debe leer esto, independientemente de si es un entusiasta de Slackware o no, porque encontrará que la mayor parte de esto también es aplicable a otras distribuciones.

1 El preámbulo

Existen infinitas razones para querer ejecutar un AP desde un entorno estándar de Linux en lugar de linux con funciones limitadas, que vienen en la mayoría de los dispositivos de AP, no estamos aquí para discutirlos todos, pero si estás leyendo esto, has encontrado tu propia motivación para hacerlo.

La nueva instalación de Slackware no se puede utilizar para crear un punto de acceso inalámbrico, pero no te preocupes, casi todo tiene la solución que necesitas, con solo una cosa que debe obtenerse de fuera de los paquetes oficiales de Slackware y un poco de configuración, puede lograr lo que la mayoría de los AP de gama baja ofrecen. Después de todo, la mayoría del AP de casa/pequeña oficina en el comercio ejecuta un Linux realmente limitado sobre un hardware ridículamente con poca potencia, ¡solo es cuestión de obtener el software necesario y configurarlo!
Echemos un vistazo a cómo podría hacerlo.

2 requisitos de hardware

Supongo que casi cualquier hardware compatible con Linux tiene la capacidad de lidiar con NICs cableadas e inalámbricas,ya que eso es lo que necesitaremos en términos de hardware para hacer un AP y porque realmente no importa a qué bus están conectadas las NIC (pci, pcie, pcix o incluso usb), casi cualquier hardware compatible con Linux estará bien. Elegirá el hardware que puede manejar la carga de tráfico que desea dirigir.

Una cosa que debe comprobar es que su dispositivo inalámbrico es compatible con el software AP mirando la salida de:

```
iw list
```

Mira la sección:

```
 software interface modes:
              * AP/VLAN
              * monitor
```

La AP debe estar entre los modos de interfaz de software.

Bueno, todavía hay algunas NIC inalámbricas que admiten el modo maestro de hardware, pero eso es 3 contra todo lo demás. No me voy a molestar en entrar en detalles con esas 3 excepciones. Google

es tu amigo si quieres insistir en el modo maestro de hardware.

3 requisitos de software

Aquí su millaje puede variar según el tipo de configuración que desee, pero seamos sencillos para comenzar ofreciendo cosas similares a las que tienen los puntos de acceso de gama más baja: dhcp, dns forwarder, WPA PSK, firewall y puente de interfaces cableadas e inalámbricas. Incluso con eso hay varias posibilidades con los paquetes Slackware enviados: podría ejecutar bind y full dhcp server, pero para simplificar, combinemos esos 2 en un software que maneja ambos (dnsmasq). Si ya ejecuta el servidor bind o dhcp, probablemente sepa cómo hacer que funcionen para el AP de todos modos.

Los puntos de acceso de gama baja también son compatibles con WEP, pero hay un problema con su implementación que lo hace muy vulnerable (busque en la vulnerabilidad de Wep y lea más acerca de él o simplemente créame que incluso con una máquina de rendimiento extremadamente bajo, puede descifrar las claves WEP). El uso de un wifi protegido por WEP es básicamente lo mismo que no tener ninguna protección, por lo que no voy a considerar esa opción. Más adelante podría hablar sobre cómo no tener ninguna protección, pero es algo que realmente debería evitar, así que por el momento, hagamos las cosas con la seguridad WPA-PSK, que es la seguridad de nivel de entrada de WiFi.

Otro error común es hacer redes SSID WiFi ocultas: en términos de seguridad, en realidad es menos seguro porque hacer que a los clientes les resulte más difícil encontrar el AP hace que sea más fácil para un usuario malintencionado acceder a la LAN inalámbrica. No hagamos nuestro WiFi con un ssid oculto.

3.1 Opciones de kernel requeridas

El kernel que se incluye con Slackware tiene todo lo que necesita, pero en caso de que construya el suyo, deberá verificar estas opciones junto con lo que sea necesario para que su hardware funcione correctamente:

- 802.1d Puente Ethernet
- Marco de filtrado de paquetes de red (Netfilter) [y una gran cantidad de sub opciones allí] (*)
- cfg80211 - API de configuración inalámbrica
- cfg80211 compatibilidad de extensiones inalámbricas
- Rutinas comunes para controladores IEEE802.11
- Generic IEEE 802.11 Networking Stack (mac80211)

(*) Si no está familiarizado con iptables, lo mejor es verificar todas las subopciones que no sean experimentales.

3.1.1 Parámetros del kernel en tiempo de ejecución

Aquí están los parámetros del kernel en tiempo de ejecución que he agrupado en un solo lugar, como

algunas otras distribuciones, en /etc/sysctl.conf

```
net.ipv6.conf.all.disable_ipv6=1
net.ipv4.ip_forward=1
net.ipv4.conf.all.rp_filter=1
net.ipv4.conf.default.rp_filter=1
#usa esto si obtienes un comportamiento extraño que se parece vagamente a no
encontrar pmtu
#net.ipv4.tcp_sack=0
net.ipv4.ip_dynaddr=1
net.ipv4.conf.all.accept_source_route=0
net.ipv4.conf.default.accept_source_route=0
net.ipv4.conf.all.accept_redirects=0
net.ipv4.conf.default.accept_redirects=0
#Use esto si los agujeros negros icmp se convierten en un problema para
usted
#net.ipv4.tcp_mtu_probing=1
#net.ipv4.tcp_base_mss=1024
```

3.2 Paquetes de Slackware

- dnsmasq
- bridge-utils
- iptables
- wireless-tools
- iw
- libnl
- openssl
- ppp (*)
- rp-pppoe (*)

(*)solo si desea que su AP administre realmente su conexión a Internet.

3.3 Otro software

- hostapd

Hostapd no se incluye con los paquetes de Slackware y es el único software adicional que necesita para una configuración básica. Para obtener hostapd, puede descargar una versión binaria de alguna fuente confiable o compilarla desde las fuentes. Puede ser que sea un poco mejor compilando desde fuentes, de modo que obtenga una versión bastante reciente, pero depende de usted. Simplemente no intentes usar versiones muy antiguas de hostapd como 0.6.7 en 2.6+ kernels:hostapd ha seguido los controladores de pila inalámbrica del kernel y ha pasado de ser compatible con dispositivos FullMAC a SoftMAC en casi todos los dispositivos a través del nl80211. Hoy en día, la combinación de hardware y controlador compatible con FullMAC es difícil de conseguir (Prism2/2.5/3 y Atheros ar521x) todos los demás tienen que usar SoftMAC, por lo que sugiero ignorar los 2 que aún podrían hacerlo con FullMAC y solo ir a SoftMAC para todos, lo que le permite usar casi cualquier tarjeta WiFi para crear un AP. Si lo desea, puede leer más sobre mac80211 aquí.

Las fuentes de Hostapd se pueden descargar desde aquí, debería ver la versión estable reciente (2.5 la última vez que se editó este artículo) y evitar el desarrollo/las ramas antiguas. Compilar hostapd es realmente simple:

1. extraer las fuentes
2. ingrese al árbol de origen (debería ver algo como esto: CONTRIBUCIONES DE COPIA DE README hostapd/ patches/ src/)
3. ingrese al subdirectorio hostapd (de aquí en adelante nos referiremos a esta carpeta como HOSTAPD_SOURCE_TREE)
4. edite el archivo defconfig y habilite las funciones opcionales que necesite (la configuración predeterminada es buena para una configuración simple)
5. Copie el archivo defconfig a .config
6. make
7. make install (u opcionalmente solo coloque en /usr /local /bin solo el binario de hostapd)

Si está utilizando un compilador cruzado para apuntar al hardware que no es x86, asegúrese de haberlo configurado correctamente: si tiene problemas, puede intentar una compilación nativa ya que la construcción de hostapd es realmente simple y no requiere mucho hardware.

3.3.1 Integridad de archivos

Si su AP también actúa como un enrutador, probablemente estará expuesto a malware e incluso si hace todo lo posible por mantener a los usuarios malintencionados fuera de su trabajo, es posible que todavía encuentren una forma de ingresar. Si su enrutador está expuesto a Internet, es posible que desee considerar algún tipo de herramienta de integridad de archivos que le avise si los archivos han sido manipulados. La verificación de la integridad de los archivos podría ser un artículo completo, por lo que no voy a entrar en detalles más allá de recomendar que lea más sobre las comunidades orientadas a la seguridad como security focus o tal vez solo hacer una búsqueda de google en 'herramienta de integridad de archivos'.

4 Configurando

Ahora veamos cómo configurar todo para que funcione correctamente.

4.1 Hostapd

Cree una carpeta hostapd dentro de /etc y copie en ella algunos archivos:

```
mkdir /etc/hostapd
cp HOSTAPD_SOURCE_TREE/hostapd.conf /etc/hostapd/hostapd.template
cp HOSTAPD_SOURCE_TREE/hostapd.wpa_psk /etc/hostapd/
```

Ahora, suponiendo que wlan0 es la interfaz que utilizará para crear el AP, le sugiero que copie hostapd.template a wlan0.conf o br0.conf si está enlazado.

```
cd /etc/hostapd
cp hostapd.template wlan0.conf
```

como alternativa, simplemente cópielo en hostapd.conf ... pero si desea ejecutar varios AP, es posible que prefiera diferenciar sus archivos de configuración.

Estas son las cosas que usted tiene que editar en wlan0-conf

- interface=wlan0
- driver=nl80211
- ssid=your_ssid
- hw_mode=g #to keep it simple don't attempt n mode right away even if your hardware supports it
- channel=6 #or whatever other channel you prefer
- macaddr_acl=1 # see notes below
- # 0 = accept unless in deny list (iptables mac filtering and optionally have a ban list)
- # 1 = deny unless in accept list (mandatory to have a mac address ACL)
- accept_mac_file=/etc/hostapd/wlan0.accept
- # deny_mac_file=/etc/hostapd/wlan0.deny
- wpa=2
- wpa_psk_file=/etc/hostapd/wlan0.wpa_psk
- wpa_key_mgmt=WPA-PSK WPA-PSK-SHA256
- wpa_pairwise=TKIP
- rsn_pairwise=CCMP
- wpa_group_rekey=600
- wpa_gmk_rekey=86400
- wpa_ptk_rekey=600

Esto utilizará autenticación WPA-PSK/WPA2-PSK. Hecha un vistaso here en la sección “Un buen punto de inicio para un punto de acceso habilitado para wpa y wpa2 es:” para más detalles sobre la autenticación.

Existen ventajas y desventajas al tratar con el filtrado de direcciones mac, ya sea con iptables o con mac ACL, para ponerlo en términos realmente breves, la principal diferencia es que hacerlo con iptables le permite mantener todas las cosas de filtrado en un solo lugar, pero tiene una tendencia a prolongarse si su red tiene muchos clientes mientras lo hace con mac ACL requerirá la edición de un archivo separado (que contiene solo una lista de direcciones mac separadas en una nueva línea) pero realmente puede simplificar las cosas si tiene varios clientes cableados e inalámbricas.

La configuración de hostapd anterior tiene mac ACL, por lo que si la sigue de cerca, ahora debe crear su /etc/hostapd/wlan0.accept que contiene una nueva lista de macs separados por líneas que desea permitir en su red inalámbrica. No hacerlo no permitirá que ninguno de sus clientes inalámbricos se autentique, incluso con las credenciales correctas.

También hay una opción para decirle a hostapd sobre el puente, pero el comentario en hostapd.conf dice: “Si el parámetro bridge no está establecido, los controladores descubrirán automáticamente la interfaz del bridge (suponiendo que sysfs esté habilitado y montado en /sys) y este parámetro puede no ser necesario”. Lo seguí al pie de la letra y lo ignore por completo.

```
#bridge=br0
```

y además de eso estaremos armando las interfaces después de haber configurado el AP. No he jugado con esta opción, pero sospecho que habilitarla y/o interrumpirla antes de iniciar hostapd puede resultar en que los clientes cableados tengan la obligación de usar WPA también. Ahora es el momento de lidiar con la frase de contraseña para asociarse: hay más de una forma de tratar las contraseñas, pero me gusta usar wpa_psk_file ya que permite las contraseñas de NIC, e incluso tener contraseñas diferentes para sus diferentes estaciones wifi. Si quieres mantener las cosas simples solo usa el MAC especial “00:00:00:00:00:00” que coincidirá con cada MAC, por lo tanto, tendrá la misma frase de contraseña para todos los clientes. Aquí hay un ejemplo:

```
00:00:00:00:00:00 your_psk_for_all_stations
```

Ahora deberías estar listo para encender hostapd

```
/usr/local/bin/hostapd -B /etc/hostapd/wlan0.conf
iwconfig  wlan0
wlan0     IEEE 802.11bg  Mode:Master  Tx-Power=20 dBm
        Retry  long limit:7   RTS thr:off   Fragment thr:off
        Power Management:on
```

e iwconfig debe informar que la interfaz está en modo maestro. Si esto falla, o bien tiene una versión anterior de hostapd que no es compatible con el nl80211 de su kernel le faltan las opciones necesarias para admitir nl80211.

Si prefiere usar iw, este es el tipo de salida que debe tener:

```
iw dev wlan0 info
Interface wlan0
      ifindex 4
      type AP
      wiphy 0
      channel 6 (2437 MHz) NO HT
```

4.2 El puente

Ahora es el momento de unir las NIC inalámbricas y por cable. Es posible usar la configuración nativa del puente desde rc.inet1.conf, pero es posible que deba reiniciar el puente después de iniciar hostapd, por lo que aquí están los comandos necesarios para configurarlo manualmente.

```
/usr/sbin/brctl addbr br0
/usr/sbin/brctl stp br0 off #turn off spanning tree unless you know you're
going to make loops
/usr/sbin/brctl addif br0 wlan0 #add the wireless nic in the bridge
/usr/sbin/brctl addif br0 eth0  #add the wired nic in the bridge
/sbin/ifconfig wlan0 0.0.0.0 up #bring up the interfaces
/sbin/ifconfig eth0 0.0.0.0 up
/sbin/ifconfig br0 192.168.0.1 up #give the bridge an ip address or dnsmasq
will not work right
```

4.3 Servidores DNS y DHCP

Ahora es el momento de comenzar dnsmasq. Si lo desea, puede dejarlo ejecutándose desde el arranque o incluso ejecutar servidores específicos a su elección.Ahora ejecuto instancias dnsmasq separadas para cada AP, así que abandoné rc.dnsmasq de Slackware y lo inicio desde mis scripts de red personalizados que requieren archivos de configuración separados para cada interfaz (como /etc/dnsmasq/br0.conf y /etc/dnsmasw/wlan1.conf). La configuración es algo que debe considerar para adaptarse mejor a sus necesidades de red… Echemos un vistazo a algunos de los tings más comunes. Suponiendo que desea asignar direcciones IP que pertenecen a 192.168.0.0/24 y estas son las opciones que necesitará:

- interface=br0
- except-interface=lo
- bind-interfaces
- listen-address=192.168.0.1
- dhcp-range=192.168.0.2,192.168.0.254,24h
- dhcp-leasefile=/run/dnsmasq/dnsmasq.leases
- conf-dir=/etc/dnsmasq.d

Las opciones except-interface bind-interface y listen-address son particularmente útiles si desea ejecutar más de una instancia de dnsmasq. Para que el puente funcione correctamente, no olvide permitir el reenvío utilizando rc.ip_forward o ejecutando

```
/bin/echo 1 > /proc/sys/net/ipv4/ip_forward
```

Mirar la sección 3.1.1 para los otros parámetros útiles del kernel en tiempo de ejecución.

4.4 Cortafuegos

Ahora es un buen momento para configurar su protección de firewall. Suponiendo que la caja estará enrutando paquetes, creo que le mostraré algunas reglas que le pueden resultar útiles. Esta es la salida de iptables-save, puede editarla para realizar los cambios que necesita y luego canalizar su archivo editado a iptables-restore. Iptables-save / iptables-restore es una forma práctica de mantener la configuración para una fácil activación y edición del cortafuegos.

```
*mangle
:PREROUTING ACCEPT [16570:1916193]
:INPUT ACCEPT [16553:1912438]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
-A POSTROUTING -p tcp --tcp-flags SYN,RST SYN -j TCPMSS --clamp-mss-to-pmtu
-m comment --comment "fix packet size for stuff that's being routed through
this box (SEE NOTE *)"
COMMIT
*nat
:PREROUTING ACCEPT [1906:207422]
:POSTROUTING ACCEPT [0:0]
```

```
:OUTPUT ACCEPT [0:0]
-A POSTROUTING -s 192.168.0.0/24 -j MASQUERADE
COMMIT
*filter
:INPUT DROP [1906:207422]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -p all -m conntrack --ctstate ESTABLISHED,RELATED  -j ACCEPT
-A INPUT -p icmp -s 192.168.1.0/24 -j ACCEPT
-A INPUT -p tcp --dport 53 -s 192.168.0.0/24 -j ACCEPT -m comment --comment
"allow dns from LAN"
-A INPUT -p udp --dport 53 -s 192.168.0.0/24 -j ACCEPT -m comment --comment
"allow dns from LAN"
-A INPUT -p tcp --dport 67 -i br0 -j ACCEPT -m comment --comment "allow dhcp
from LAN"
-A INPUT -p udp --dport 67 -i br0 -j ACCEPT -m comment --comment "allow dhcp
from LAN"
-A INPUT -m mac --mac-source 0a:0b:0c:0d:0e:0f -j ACCEPT -m comment --
comment "this one can acces the AP "
-A FORWARD -p all -m conntrack --ctstate ESTABLISHED,RELATED  -j ACCEPT
-A FORWARD -m mac --mac-source 00:01:02:03:04:05 -j ACCEPT -m comment --
comment "this one can route trough the AP"
COMMIT
```

NOTA *: La fijación de MSS a PMTU puede hacer que la navegación por Internet funcione desde su LAN, pero puede frenar los paquetes VPN. La solución propuesta se ha hecho necesaria por la creciente tendencia a no encontrar PMTU. No siempre es necesario que active la solución alternativa, pero tenga en cuenta que también puede dejar de funcionar de manera inadvertida, lo que le deja un problema intermitente que es difícil de depurar. Puede ser intermitente porque la ruta que toman sus paquetes para llegar a cualquier destino en Internet no siempre es la misma necesidad de tolerancia a fallas. Esta nota podría convertirse en un artículo completo por sí mismo, por lo que here es una lectura interesante sobre el tema.

Generalmente pongo el contenido arriba en /ect/firewall.cf y agrego “/usr/sbin/iptables-restore < /etc/firewall.cf” a rc.local.

Si desea administrar su AP a través de ssh ahora es un buen momento para reiniciar sshd que no estaba de acuerdo en eliminar las interfaces con las que estaba enlazado (en la sección 4.2).

```
/etc/rc.d/rc.sshd restart
```

Si va a compartir la conexión a Internet, es posible que desee detener el acceso a ssh desde Internet agregando una regla de firewall para regularlo o haciendo que sshd se enlace solo a la dirección asignada a br0. La configuración que se muestra arriba no permitirá el tráfico ssh entrante desde el enlace de Internet (según la regla de seguimiento de la conexión), pero es posible que desee realizar una copia de seguridad con mayor seguridad. Las cadenas de iptabls más largas generalmente tienen un impacto negativo en el rendimiento del firewall, por lo que es posible que desee agregar algo como esto a /etc/ssh/sshd_config:

```
ListenAddress 192.168.0.1
```

En este punto, debe poder asociar clientes al AP. Con las reglas de iptables anteriores, el cliente con MAC 0a:0b:0c:0d:0e:0f puede asociarse y acceder al AP en sí, pero no puede enrutar a través del AP (con la excepción de las consultas dns que son enviadas por dnsmasq),el cliente con MAC 00:01:02:03:04:05 puede enrutar el AP pero no acceder al AP en sí. El ejemplo fue simplemente para mostrar claramente la diferencia de tener paquetes que van a la caja y enrutar paquetes a través de la caja ya que este comportamiento fue radicalmente diferente en las ip_chains. Otra forma de implementar MAC ACL aún más compleja para decidir quién administra AP, quién solo puede enrutar y quién puede hacer ambas cosas se puede hacer mediante ebtables (es necesario agregar eso a la lista de software que no está presente en Slackware) y jugar con el cadena de broute En todos los casos, puede encontrar interesante el diagrama de flujo de filtros de red.

Si no desea la ACL de la dirección MAC o solo implementa la ACL de MAC para el AP, puede reemplazar

1. A INPUT -p icmp -s 192.168.1.0/24 -j ACCEPT
2. A INPUT -p tcp –dport 53 -s 192.168.0.0/24 -j ACCEPT -m comment –comment “allow dns from LAN”
3. A INPUT -p udp –dport 53 -s 192.168.0.0/24 -j ACCEPT -m comment –comment “allow dns from LAN”
4. A INPUT -p tcp –dport 67 -i br0 -j ACCEPT -m comment –comment “allow dhcp from LAN”
5. A INPUT -p udp –dport 67 -i br0 -j ACCEPT -m comment –comment “allow dhcp from LAN”
6. A INPUT -m mac –mac-source 00:01:02:03:04:05 -j ACCEPT -m comment –comment “some comment”
7. A FORWARD -m mac –mac-source 00:01:02:03:04:05 -j ACCEPT -m comment –comment “some other comment”

Con algo como esto

```
-A INPUT -p all -i br0 -j ACCEPT
-A FORWARD -p all -i br0 -j ACCEPT
```

La mayoría de los AP disponibles también le permiten hacer una cantidad de reenvío de puertos, esto también es un trabajo de iptables. Recuerde que estamos enmascarando todo el tráfico saliente para que parezca que viene del propio AP, por lo que solo necesitamos crear reglas del tráfico entrante. Suponiendo que desea ejecutar un servidor web en 192.168.0.2 es posible que desee agregar una regla como esta en la cadena FORWARD de la tabla de filtro:

1. A FORWARD -p tcp -d 192.168.0.2 -m multiport –dports 80,443 -j ACCEPT -m comment –comment “allow http traffic to be routed thought the box only to the correct server”

y una regla como esta en la cadena PREROUTING de la tabla nat:

```
-A PREROUTING -p tcp -m multiport --dports 80,443 -j DNAT --to-destination
192.168.0.2 -m comment --comment "nat incomming http requests to local
destination before routing"
```

Para entender cómo funciona esto, debe consultar el diagrama de flujo de filtro de red: Cuando una solicitud http llega al AP desde el enlace de Internet, primero se cambiará el DNAT en la etapa de pre-enrutamiento nat, pero la política de filtro la eliminará si no encontramos una manera de déjalo

entrar, ahí es donde entra en juego la regla de filtro de avance.

Si comienza a volverse loco con la transferencia de archivos grandes a través de redes rápidas para un problema que parece estar relacionado con mtu pero no es posible que desee considerar la posibilidad de desactivar net.ipv4.tcp_sack.

Si su ISP le da algún tipo de cuota de tráfico, es posible que desee agregar algunas cuotas a la configuración de su firewall. Es posible que entienda completamente las consecuencias de la transmisión en su cuota de ISP, pero tal vez el resto de la familia no lo haga: darles una cuota podría ahorrarle un ajuste cuando necesite hacer un trabajo urgente que requiera conexión a Internet. Hay varias maneras en las que podría establecer cuotas en clientes específicos en su LAN, solo tenga en cuenta algunas cosas:

- las reglas con cuotas dejan de coincidir una vez que se excede la cuota
- el lavado de sus tablas reiniciará todos los contadores de cuotas
- los contadores de cuota no se reinician cuando su ISP restablece su cuota de Internet

Este es un ejemplo de cómo podría poner una cuota en la cadena FORWARD para detener a un cliente que usa más de 300Mb diarios:

```
-A FORWARD -p all -m conntrack --ctstate ESTABLISHED,RELATED ! -d
192.168.0.200 -j ACCEPT -m comment --comment "allow related traffic but not
for 192.168.0.200 that has a quota"
-A FORWARD -p all -m conntrack --ctstate ESTABLISHED,RELATED -d
192.168.0.200 -j ACCEPT -m quota --quota 314572800 -m comment --comment
"allow quota related traffic for 192.168.0.200"
-A FORWARD -s 192.168.0.200 -j ACCEPT -m quota --quota 314572800 -m comment
--comment "outgoing quota for 192.168.0.200"
-A FORWARD -d 192.168.0.200 -j ACCEPT -m quota --quota 314572800 -m comment
--comment "incoming quotafor 192.168.0.200"
-A FORWARD -s 192.168.0.200 -d 192.168.0.0/24 -j ACCEPT -m quota --quota
314572800 -m comment --comment "allow LAN traffic anyway for 192.168.0.200"
```

O puede usar una cadena definida por el usuario para agrupar todo su cuota de tráfico en una sola cuota como esta:

```
-A FORWARD -d 192.168.1.0/24 -m conntrack --ctstate RELATED,ESTABLISHED -m
comment --comment "allow quota related traff. to quoted LAN" -j QUOTA
-A FORWARD -s 192.168.1.0/24 -m comment --comment "allow quoted traff. from
quoted LAN" -j QUOTA
-A QUOTA -m quota --quota 314572800 -m comment --comment "accept traffic
within quota in this userdefined chain" -j ACCEPT
-A QUOTA -m comment --comment "when quota is exceeded start rejecting" -j
REJECT --reject-with icmp-port-unreachable
```

Junto con esto, debe vaciar los contadores de iptables todos los días (o cada vez que quiera reiniciar los contadores de cuotas) con algo como esto

```
iptables -Z
```

Generalmente hago esto con un trabajo de AT porque AT tiene menos efectos adversos que CRON en sistemas de solo lectura. Aquí hay una manera posible de hacer que un trabajo de AT se programe todos los días a las 00:30:

```
# cat /usr/local/bin/flush_iptables_counters
/usr/sbin/iptables -Z
/usr/bin/at -f /usr/local/bin/flush_iptables_counters "0030 tomorrow"
#
```

Simplemente ejecútelo una vez y luego debería volver a programarse. En un sistema de solo lectura, deberá tener el directorio atjobs en tmpfs y ejecutarlo la primera vez desde rc.local.

4.5 PPP

Tectónicamente, no va a necesitar esto en un punto de acceso puro (AP), pero es común que el AP también actúe como enrutador para su acceso a Internet, en este caso necesitará configurar su enlace PPP desde su AP. Dependiendo de la forma en que su ISP le brinde acceso a Internet, puede tener o no un módem externo que puede o no entender el protocolo pppoe. Hoy en día parece que se ha vuelto menos común tener módem *DSL interno mientras que los módems externos se han vuelto más populares. La mayoría de ellos utilizan el protocolo pppoe. Técnicamente, no hay mucha diferencia entre configurar un módem interno con solo el uso de pppd en un módem externo que usa pppoe, el primero requerirá un paquete adicional (rp-pppoe) y solo algunas opciones adicionales en el archivo de configuración. Mostraré cómo configurar con el uso de pppoe, si ese no es su caso, solo elimine algunas opciones.

Si ha instalado tanto ppp como rp-pppoe, encontrará que la mayoría de la configuración ya se ha realizado con las opciones más comunes en /etc/ppp, aún hay una cantidad limitada de trabajo que debe hacerse y también se aconseja desviarse del uso de pppoe-start para iniciar la conexión para mantener algo que permanece igual, independientemente de si está utilizando pppoe o no. Para hacer esto, configuro un igual para mi ISP y agrego algunas opciones que le indican a pppd que use el complemento pppoe.

Para hacer esto, necesita crear un archivo en /etc/ppp/peers, llámelo como algo que le haga evidente a qué se conectará el peer: por ejemplo, si su ISM se llama “Telco”, es posible que desee crear un archivo llamado algo así como /etc/ppp/peers/telco y esto es lo que debe incluir normalmente:

```
plugin rp-pppoe.so
eth1 #or whatever nic your modem is connected to
unit 0
user "username your ISM gave you"
noipdefault
noauth
default-asyncmap
defaultroute
hide-password
nodetach
usepeerdns
mtu 1492
mru 1492
```

```
noaccomp
nodeflate
nopcomp
novj
novjccomp
lcp-echo-interval 20
lcp-echo-failure 3
persist
maxfail 0           #these 3 options are particularly important if your
connection is prone to braking as the default is just 10
lcp-max-failure 0   #and after that the peer will not be able to persists
any more
lcp-max-configure 0 #so your internet connection will fail with no apparent
reason
holdoff 20
#idle 900
#demand
updetach
```

Si su ISM le cobra por tiempo en lugar del uso de la banda, es posible que desee descomentar las opciones de inactividad y demanda para que la conexión PPP no permanezca activa cuando no esté en uso. Haga lo mejor que pueda aquí en el par y tan poco como sea posible en el archivo /etc/ppp/options, de modo que posiblemente pueda tener una o más tareas de copia de seguridad que funcionen sin editar ninguna configuración. Si estas opciones no funcionan para usted, le sugiero que lea la documentación /usr/doc/rp-pppoe-3.11/HOW-TO-CONNECT y use pppoe-setup para una configuración guiada, inicie la conexión con pppoe-start y tome No es una de las opciones utilizadas, luego edite en consecuencia a su par y vuelva a usar pppd call <peer>.

Aún no hemos terminado. Necesitamos editar pap-secrets o chap-secrets para que pppd pueda completar la autenticación. El archivo que deba abordarse puede depender de su ISP, pero si coloca el mismo contenido en ambos, debería estar bien, independientemente de lo que requiera su ISP.

```
# Secrets for authentication using PAP
# client        server  secret                  IP addresses
"username your ISM gave you"  *       "password your ISM gave you"
```

Ahora puede activar su conexión a Internet con

```
pppd call telco
```

4.6 DNS Dinámico

Si va a hacer el reenvío de puertos y si su ISP no le proporciona una IP estática, es posible que desee utilizar algún tipo de servicio de DNS dinámico para que las personas puedan acceder al servicio que está reenviando.

Hay muchos proveedores de servicios DNS dinámicos gratuitos… google es tu amigo si no te gustan ninguno de los que te sugiero:

- http://www.gnutomorrow.com/r/namecheap
- http://www.dnsexit.com/
- http://www.dynu.com/
- http://freedns.afraid.org/

La mayoría le brinda un cliente para administrar la actualización… todo lo que necesita hacer es organizar que el cliente se ejecute desde el enrutador o desde cualquier otra PC de su LAN con acceso a Internet.

4.7 Configuring Clients

For the client side configurations there are many ways too, to keep things simple let's just go about it by using wpa_supplicant. Remember that regardless of whether it's a wired or wireless client: if you're using managing MAC ACL from iptables your client's MAC address needs to be added to the firewall configuration while ff you're doing MAC ACL just with hostapd then you only need to make sure your wireless clients are in the ACL.

4.7.1 Wired Clients

In most cases the modules for your wireless NIC get automatically loaded so that should not be an issue. Keep an eye out on kernel ring buffer and messages to see if your device is requesting firmware that is not available. If all is fine and the configuration is OK it's really simple: just uplink the client to the AP and start your favorite dhcp client.

4.7.2 Slackware Wireless Clients

Configuring a client to access the newly crated AP is best done with a tool like wpa_gui, as it can create the correct settings in wpa_supplicant.conf based on what's detected. So fire up wpa_gui and activate scanning until you find the newly created AP then double click it. On the window that pops up just add the password without changing any of the detected settings and click add.

If, for any reason, you don't want to use a tool like wpa_gui, and if you followed all the AP side setup to the letter you could just simply put this in /etc/wpa_supplicant.conf

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=root

network={
      ssid="your_ssid"
      psk="your_psk_for_all_AP"
      key_mgmt=WPA-PSK
      pairwise=CCMP
      auth_alg=OPEN
}
```

Make sure rc.inet1.conf is configured for using wpa_supplicant

```
IFNAME[4]="wlan0"
USE_DHCP[4]="yes"
WLAN_MODE[4]=Managed
WLAN_WPA[4]="wpa_supplicant"
WLAN_WPADRIVER[4]="wext"
```

It should then be possible to restart rc.inet1 (or just rc.inet1 wlan0_down and rc.inet1 wlan0_up) and the client should associate.

There's also the command line alternative with wpa_cli in this example we'll assume that your client is totally unconfigured and wpa_supplicant is not running. We're going to do everything on the command line:

```
# wpa_supplicant -B -W -Dwext -i wlan0 -c /etc/wpa_supplicant.conf
# wpa_cli
wpa_cli v2.4
Copyright (c) 2004-2015, Jouni Malinen <j@w1.fi> and contributors

This software may be distributed under the terms of the BSD license.
See README for more details.

Selected interface 'wlan0'

Interactive mode

> scan
OK
<3>WPS-AP-AVAILABLE
> scan_results
bssid / frequency / signal level / flags / ssid
02:0c:42:f9:73:23       2412    -58     [WPA-PSK-CCMP][WPA2-PSK-CCMP][ESS]
a4:51:6f:95:37:b6       2462    -58     [WPA2-PSK-CCMP][WPS][ESS]
Windows Phone0377
00:0c:42:f9:73:23       2412    -62     [ESS]   Insecure-WiFi
>  add_network
0
> set_network 0 ssid "Windows Phone0377"
OK
> set_network 0 psk "passwordforcrappywindowsphone"
OK
> enable_network 0OK
OK
<2>Trying to authenticate with a4:51:6f:95:37:b6 (SSID='Windows Phone0377'
freq=2437 MHz)
<2>Trying to associate with a4:51:6f:95:37:b6 (SSID='Windows Phone0377'
freq=2437 MHz)
<2>Associated with a4:51:6f:95:37:b6
<2>WPA: Key negotiation completed with a4:51:6f:95:37:b6 [PTK=CCMP GTK=CCMP]
<2>CTRL-EVENT-CONNECTED - Connection to a4:51:6f:95:37:b6 completed (reauth)
```

```
[id=0 id_str=]
> save_config
OK
> quit
#
```

If all went right and your wpa_supplicant.conf file had

```
update_config=1
```

in it the above snippet would have saved the new network to wpa_supplicant.conf and associated you with it.

Remember that if you're associating with an non secured AP you need to use this:

```
> set_network 0 ssid "Insecure-WiFi"
OK
> set_network 0 key_mgmt NONE
OK
>
```

4.7.3 Other Linux Distributions Wireless Clients

I've tried various other flavor distributions ... most don't use wpa_gui for associating to AP but some sort of other tool that generally pops up when you click on the icon that notifies the presence of an Access Point. After a few headaches I found that best association success is achieved by forcing setup for hidden AP even if the AP I'm configuring has not the hidden essid. You can always use wpa_cli on the command line if it's shipped with whatever distro you prefer.

5 The AP's Components Status

It might be interesting to query the AP status at any time to see how it's doing. There are several ways to go about this, I'll try to show the simplest way that possibly works on any Linux system.

5.1 Associated Stations

The associated stations can be shown by using either hostapd_cli or iw:

```
root@router:~# hostapd_cli -p /run/hostapd all_sta
Selected interface 'wlan0'
00:01:02:03:04:05
flags=[AUTH][ASSOC][AUTHORIZED][SHORT_PREAMBLE][WMM]
aid=2
capability=0x431
listen_interval=5
```

```
supported_rates=02 04 0b 16 0c 12 18 24 30 48 60 6c
timeout_next=NULLFUNC POLL
dot11RSNAStatsSTAAddress=00:01:02:03:04:05
dot11RSNAStatsVersion=1
dot11RSNAStatsSelectedPairwiseCipher=00-0f-ac-4
dot11RSNAStatsTKIPLocalMICFailures=0
dot11RSNAStatsTKIPRemoteMICFailures=0
hostapdWPAPTKState=11
hostapdWPAPTKGroupState=0
rx_packets=16176
tx_packets=20464
rx_bytes=1974499
tx_bytes=23121726
connected_time=439
root@router:~#
```

```
root@router:~# iw wlan0 station dump
Station 00:01:02:03:04:05 (on wlan0)
      inactive time:  0 ms
      rx bytes:       1956931
      rx packets:     16009
      tx bytes:       23105532
      tx packets:     20352
      tx retries:     1484
      tx failed:      1
      signal:         -63 dBm
      signal avg:     -62 dBm
      tx bitrate:     48.0 MBit/s
      authorized:     yes
      authenticated:  yes
      preamble:       short
      WMM/WME:        yes
      MFP:            no
      TDLS peer:              no
root@router:~#
```

5.2 DHCP Leases

You can dump dnsmasq's lease file to see the dhcp leases

```
root@router:~# cat /run/dnsmasq/br0.leases
1411875361 00:01:02:03:04:05 192.168.0.3 b3bo *
1411874427 0a:0b:0c:0d:0e:0f 192.168.0.4 printsrv *
root@router:~#
```

5.3 ARP Table

The system arp cache cache can be inspected by using the arp command

```
root@router:~# arp -an
? (192.168.0.3) at 00:01:02:03:03:05 [ether] on br0
? (192.168.0.4) at 0a:0b:0c:0d:0e:0f [ether] on br0
root@router:~#
```

5.4 Socket Status

To get a list of the open sockets on your system you can use socklist if you have it installed. If you don't have it installed you can use a somewhat simpler script like the one shown blow (a little less information rich and careless formatting but still useful and much faster then doing it manually):

```
#!/bin/bash

ahex2i ()
{ echo "0x$1" |awk '{printf("%i",strtonum($1))}'
}

hex2ip ()
{ ahex2i ${1:6:2}
  echo -n "."
  ahex2i ${1:4:2}
  echo -n "."
  ahex2i ${1:2:2}
  echo -n "."
  ahex2i ${1:0:2}
}

getpidofsocket ()
{ find /proc/*/fd -type l -printf "%h %l\n" 2>/dev/null| grep
"socket:\[$1\]" |awk -F/ '{print $3}'
}

echo "          local                   forein  pid/cmd"
grep -v "^ *sl " /proc/net/tcp |awk '{printf("%s %s %s\n",$2,$3,$10)}' |sed
-e "s/:/ /g" | while read LINE
do
  LA=(${LINE})
  PID=$(getpidofsocket ${LA[4]})
  echo "tcp     $(hex2ip ${LA[0]}):$(ahex2i ${LA[1]})   $(hex2ip
${LA[2]}):$(ahex2i ${LA[3]})  ${PID}/$(cat /proc/$PID/comm)"
done

grep -v "^ *sl " /proc/net/udp |awk '{printf("%s %s %s\n",$2,$3,$10)}' |sed
-e "s/:/ /g" | while read LINE
```

```
do
  LA=(${LINE})
  PID=$(getpidofsocket ${LA[4]})
  echo "udp     $(hex2ip ${LA[0]}):$(ahex2i ${LA[1]})   $(hex2ip
${LA[2]}):$(ahex2i ${LA[3]})  ${PID}/$(cat /proc/$PID/comm)"
done
```

Please note the above script only works on 2.6 kernels or above. This probably also applies to socklist.

6 Remote Administration

I'm not advocating that allowing remote administration from your WAN connection is a good thing but there are times where it may be necessary so here are some tips for minimizing the risk of having your router suffering brute force attacks or other bad things happen to it.

I'm an old fashioned system administrator so for me remote administration is done via ssh, if you've added a nice web administration tool to your AP/Router keep in mind that running apache just for the sake of having remote web administration will expose you to a whole lot of security issues that need to be addressed and maintained over time.

1. use non standard ports
2. disallow password authentication
3. minimize your attack surfaces

Let me give you a little reasoning for the list.

6.1 Use Non Standard Ports

Whatever is your remote administration tool of choice it's a good idea not to leave access to it from WAN on it's well known port, making it less obvious that you run such a service. If you do this there's a good chance that your AP/Router will never get unwanted attention.

6.2 Disallow Password Authentication

Allowing password authentication is a welcome for brute force attacks so avoid it wherever possible (ie for ssh administrations only allow authentication with keys). If you're doing web based remote administration you could send in a key via get and then set a cookie or something like that along with password protected htaccess.

6.3 Minimize Your Attack Surfaces

Your AP/Router should expose to the WAN connection nothing more then what is really needed. Scanning your own AP/Router and closing or disabling unnecessary services to WAN is something you should allays do so that you minimize the attack surfaces should you ever get unwanted attention.

6.3.1 Avoid Running Remote Administration 24x7

If you can have remote administration active only when you need it you're not leaving the attack surface available all the time but then you need an easy way to turn it on when you're away from home. I've two means of doing so:

- If any family member is home it can be temporarily activated by pressing a specific button on the router itself (it's the second button under the blue led in the images below).
- If nobody is home I've modified a 200 line minimal web server (nweb) to listen to requests on a non standard port and temporarily allow remote administration if a specific url is requested.

Whichever way the temporary remote admin is enabled it also get's automatically turned off after some time (should you ever forget to turn it off once you're done).

Nweb is a really basic webserver that only servers static html images and a few archive formats, it does not even allow directory listing. Besides that I have it parse and enable before serving the page … so if you don't physically have the page that enables the remote administration a 403 is returned anyway leaving no clue as to waht was done in response to that request.

If you're interested in nweb you can get it by googling “nweb tiny web server”. You should hit github with something like nweb23.c with some 204 lines of C code. It should be easy for you to modify the source to match your needs.

7 Wrapping It Up

Now that you've done the configuration maybe next time you want to start the AP you want to do it surely more efficiently.

7.1 Simple Starter Script

Just put a few commands in a script to start it up really quickly:

```
/bin/echo 1 > /proc/sys/net/ipv4/ip_forward #you don't need this if you use
rc.ip_forward
#. /etc/rc.d/rc.dnsmasq restart #only if it's not started at boot time
/usr/local/bin/hostapd -B /etc/hostapd/wlan0.conf
/usr/sbin/brctl addbr br0
/usr/sbin/brctl stp br0 off
/usr/sbin/brctl addif br0 wlan0
/usr/sbin/brctl addif br0 eth0
/sbin/ifconfig wlan0 0.0.0.0 up
/sbin/ifconfig eth0 0.0.0.0 up
```

```
/sbin/ifconfig br0 192.168.0.1 up
/usr/sbin/iptables-restore < /etc/firewall.cf
/etc/rc.d/rc.sshd start
```

If you want it to come up at boot time you could run the script from rc.local or even just put those commands directly in there. If you want a neater solution totally integrated in the init scripts read on.

7.2 Modifying Slackware Init Scripts

I've not yet done this but I can suggest a possible way of doing it.

Edit rc.inet1.conf and put in the following things: (that will later be managed by rc.wireless)

```
IPADDR[0]=""
NETMASK[0]=""
USE_DHCP[0]=""
DHCP_HOSTNAME[0]=""

IFNAME[0]="br0"
BRNICS[0]="wlan0 eth0"
IPADDR[0]="192.168.0.1"
NETMASK[0]="255.255.255.0"

IFNAME[4]="wlan0"
WLAN_ESSID[4]=your_ssid
WLAN_MODE[4]=Master
WLAN_RATE[4]="g"
WLAN_CHANNEL[4]="6"
WLAN_IWPRIV[4]="set AuthMode=WPA-PSK | set EncrypType=TKIP | set
WPAPSK=your_psk_for_all_AP"
```

Some of this will be picked up by rc.inet1 (the bridge stuff) and some by rc.wireless but the stock one knows nothing about Master mode. We need to modify rc.wireless to do some extra stuff: pick up values from WLAN_ESSID[4], WLAN_MODE[4], WLAN_CHANNEL[4], WLAN_IWPRIV[4] and sed inline the values in /etc/hostapd/wlan0.conf and /etc/hostapd/hostapd.wpa_psk. Also WLAN_RATE[4]=“g” will be indigestible for the stock rc.wireless if you like you could leave that as 54 and only convert it for Master mode. A special attention needs to go to WLAN_MODE[4]=Master as the stock rc.wireless knows nothing about master mode. Alternatively one could just have IFNAME[4]=“wlan0” and WLAN_MODE[4]=Master and just manually put all the other stuff in etc/hostapd/wlan0.conf.

I had a quick look at rc.inet1 and apparently it starts bridge after wireless so it should be ok without having to restart br0 after AP is initiated.

The sshd server is started after networking so no need to restart that now.

You can write your own rc.firewall or just leave

```
usr/sbin/iptables-restore < /etc/firewall.cf
```

in rc.local.

7.3 Automating AP Startup For USB WiFi Dongles

Udev is very powerful and can do a variety of actions upon detecting certain events, like the appearance of a NIC. In fact it already does that and renames interfaces according to MAC address (have a look at /etc/udev/rules.d/70-persistent-net.rules and see how your interfaces get the same name even if you remove the modules and reinsert them in the wrong order). Apart from renaming NICs and creating device files it can also execute commands or external helper scripts … this is particularly handy if, for example, you wish that upon plugging a USB Ethernet dongle it automatically assigns an address via DHCP. A few years ago I wrote a nethelper.sh script for my ClashNG that upon detecting LAN devices would run rc.inet1 to start the NIC according to what was configured in rc.inet1.conf. Ok ClashNG is a minimalistic thing that uses busybox to replace most of the binaries but still it might be a useful example. Ignore the part that writes stuff to debug what was going on … I'm not udev expert.

```
#!/bin/sh
DEVNAME="$1"
COMMAND="$2"

case $DEVNAME in
  eth*|ath*|wlan*|ra*|sta*|ctc*|lcs*|hsi*)
    case $COMMAND in
      'start')
        if [ -x /etc/rc.d/rc.inet1 ]; then
          if ! /sbin/ifconfig | /bin/grep -q "^${DEVNAME} "; then
            /etc/rc.d/rc.inet1 ${DEVNAME}_start
          fi
        fi
      ;;
      'stop')
         if [ -x /etc/rc.d/rc.inet1 ]; then
           if /sbin/ifconfig | /bin/grep -q "^${DEVNAME} "; then
             /etc/rc.d/rc.inet1 ${DEVNAME}_stop
           fi
         fi
         # Does dhcpcd appear to still be running on the
         # interface?  If so, try to stop it.
         if [ -r /etc/dhcpc/dhcpcd-$DEVNAME.pid -o -r /var/run/dhcpcd-
$DEVNAME.pid ]; then
           /sbin/dhcpcd -k $DEVNAME
           # Force garbage removal, if needed:
           if [ -r /etc/dhcpc/dhcpcd-$DEVNAME.pid ]; then
             /bin/rm -f /etc/dhcpc/dhcpcd-$DEVNAME.pid
           elif [ -r /var/run/dhcpcd-$DEVNAME.pid ]; then
             /bin/rm -f /var/run/dhcpcd-$DEVNAME.pid
           fi
         fi
         # If the interface is now down, exit with a status of 0:
```

```
        if /sbin/ifconfig | /bin/grep -q "^${DEVNAME} " ; then
          exit 0
        fi
      ;;
      *)
        logger "usage $0 interface start|stop"
        exit 1
      ;;
    esac
  ;;
  *)
    logger "Interface $DEVNAME not supported."
    exit 1
  ;;
esac
exit 0
```

Although this was written for ClashNG in 2011 it still works right with Slackware 14.1, just modify udev rules to correctly use the helper script like shown below and it's done.

```
root@r2d2:/tmp/clashng/lib/udev/rules.d# grep  "nethelper\.sh" *
90-network.rules:SUBSYSTEM=="net", NAME=="?*", ACTION=="add",
RUN+="nethelper.sh $env{INTERFACE} start"
90-network.rules:SUBSYSTEM=="net", NAME=="?*", ACTION=="remove",
RUN+="nethelper.sh $env{INTERFACE} stop"
root@r2d2:/tmp/clashng/lib/udev/rules.d#
```

These udev rules were in 90-network.rules which no longer exists so I suggest you put these rules in 90-local.rules in current Slackware versions. Without further modification (as long as the rc.inet1.conf has USE_DHCP[?]=“yes” for the interface that will be assigned to your dongle) this will allow for USB Ethernat dongles to be configured via DHCP automatically when you plug them in to your box. Remember to do a “/etc/rc.d.rc.udev reload” after changing udev rules.

At the time I also had a modified rc.wireless that would also start up AP … but I was using a different configuration scheme dew to busybox ash limitations (arrays were a problem) and other requirements I had in mind at the time … but never the less here's the section relevant to AP and it was heavily based on Slackware's rc.wireless anyway. Have a look at the code script fragments below to get ideas non how you might want to go about it:

```
# If we stop a wireless interface using wpa_supplicant,
# we'll kill its wpa_supplicant daemon too and exit this script:
if [ "$2" = "stop" ]
then
  if [ "${WLAN_MODE}" = "Master" ]
  then
    echo "$0: $2 $1 AP mode" |$LOGGER
    /bin/kill $(/usr/bin/fuser /var/run/hostapd/$1 2>&1 |/usr/bin/awk
'{print $NF}')
  fi
```

```
  WPAPID=`echo \`ps axww|grep wpa_supplicant |grep i${INTERFACE}\` |cut -f1
-d' '`
  [ ${WPAPID} ] &&  kill ${WPAPID}
  return 0
fi
...
...
...
  if [ "$MODE" = "Master" ]
  then
    echo "$0: $2 $1 AP mode" |$LOGGER
    if [ -r /etc/hostapd/${1}.conf -a "$2" = "start" ]
    then
      [ $DEBUG -eq 1 ] && \
        /usr/bin/hostapd -B -d /etc/hostapd/${1}.conf >/tmp/hostapd.log 2>&1
|| \
        /usr/bin/hostapd -B /etc/hostapd/${1}.conf
    else
      /bin/kill $(/usr/bin/fuser /var/run/hostapd/$1 2>&1 |/usr/bin/tr -d
[a-z])
    fi
  else
  echo "$0:  $IWCOMMAND mode $MODE" | $LOGGER
# if $IWCOMMAND fails, try taking the interface down to run it.
# Some drivers require this.
    if ! $IWCOMMAND mode $MODE 2> /dev/null
    then
      $IFCOMMAND down
      $IWCOMMAND mode $MODE
      $IFCOMMAND up
      sleep 3
    fi
  fi
fi

# This is a bit hackish, but should do the job right...
[ ! -n "$NICKNAME" ] && NICKNAME=`/bin/hostname`
if [ -n "$ESSID" -a "$MODE" != "Master"]
then
  echo "$0:  $IWCOMMAND nick $NICKNAME" | $LOGGER
  $IWCOMMAND nick $NICKNAME
fi
```

If you do all this stuff right you can get your AP to be automatically initiated when you plug in the USB gongle devoted to doing that.

I used to devote 2 usb dongle for this: one that would be left inserted most of the time and has random generated WAP-PSK (only for family use) and one that would get temporarily plugged in for guests with a much simple WPA-PSK to aid them accessing my home network.

7.3.1 Automation With Custom Scripts

Over the years trying to maintain modified rc scripts functional across updates that involved the rc scripts themselves became cumbersome so I started moving away from modifying the stock scripts and started developing my own stuff. Don't get me wrong I still use and appreciate the stock stuff for my desktop systems. The idea behind my own scripts is based on udev detecting the interfaces (even at boot time). The basic idea is still the same: upon detection udev executes nethelper.sh script that looks for and executes /etc/rc.d/network/<NIC> start. This is not for everyone because it requires manually writing the /etc/rc.d/network/<NIC> script but I think most have the basic knowledge and maybe with a little help most can manage.

Here's is what my latest nethelper.sh looks like:

```
#!/bin/bash
PATH=/usr/local/sbin:/usr/sbin:/sbin:/usr/local/bin:/usr/bin:/bin
DEVNAME="$1"
COMMAND="$2"

case $DEVNAME in
  eth*|ath*|wlan*|ra*|sta*|ctc*|lcs*|hsi*)
    case $COMMAND in
      'start') logger -p local3.info "start: $DEVNAME $COMMAND"
        [ -x /etc/rc.d/network/$DEVNAME ] && /etc/rc.d/network/$DEVNAME
start
      ;;
      'stop') logger -p local3.info "stop: $DEVNAME $COMMAND"
        [ -x /etc/rc.d/network/$DEVNAME ] && /etc/rc.d/network/$DEVNAME stop
      ;;
       *) logger -p local3.info "unsupported command: $DEVNAME $COMMAND" ;;
    esac
  ;;
  *) logger -p local3.info "unsupported interface: $DEVNAME $COMMAND" ;;
esac
exit 0
```

Be warned that the logger facility is not yet functional in the early boot stages so it may not log the interface startup at boot time.

Here's an example of /etc/rc.d/network/wlan0 configured in master mode:

```
#!/bin/bash
set -o pipefail
PATH=/usr/local/sbin:/usr/sbin:/sbin:/usr/local/bin:/usr/bin:/bin

INAME=$(basename $0 |awk -F. '{print $NF}')

do_start ()
{ echo "Start $INAME : $(date "+%F %H:%M:%S") $$"
  ifconfig ${INAME} up
```

```
  sleep 1
  iwconfig ${INAME}
  sleep 1
  ifconfig ${INAME} down
  sleep 1
  hostapd -B /etc/hostapd/${INAME}.conf -e /dev/hwrng
  sleep 1
  ifconfig ${INAME} 0.0.0.0 up
}

do_stop ()
{ echo "Stop $INAME : $(date "+%F %H:%M:%S") $$"
  ifconfig ${INAME} 0.0.0.0 down
  PID=$(ps -eo pid,cmd |grep hostapd |grep  -w "${INAME}\.conf" |awk '{print
$1}')
  [ "$PID" != "" ] && kill -9 $PID
}

case $1 in
  start) [ $(grep -wc "${INAME}:" /proc/net/dev) -gt 0 -a $(ps -ef |grep -v
grep |grep -c "/etc/hostapd/${INAME}.conf") -lt 1 ] && do_start |tee -a
/run/${INAME}.log || exit 1 ;;
  stop) [ $(grep -wc "${INAME}:" /proc/net/dev) -gt 0 ] && do_stop |tee -a
/run/${INAME}.log || exit 1 ;;
  restart)
    if [ $(grep -wc "${INAME}:" /proc/net/dev) -gt 0 ]
    then
      do_stop |tee -a /run/${INAME}.log
      do_start |tee -a /run/${INAME}.log
    else
      exit 1
    fi
  ;;
  status)
    if [ $(grep -wc "${INAME}:" /proc/net/dev) -gt 0 ]
    then
      ip link list $INAME |sed -e "s/^[0-9]*: *//"
      iw dev $INAME info
      echo "Associated stations:"
      iw $INAME station dump | grep -w "^Station"
    else
    else
      echo "$INAME: no sich interface found on system"
      exit 1
    fi
  ;;
esac
```

Logical devices like bridges would need to be initiated either by telling all the physical device starter scripts which is the parent logical device or by adding them to rc.local. In fact wlan0 is bridged with eth0 … for simplicity I chose not to have daughter devices call parent device init scripts so I start br0

from rc.local but care has to be taken not to start daughter devices if they have already been brought up by udev (note how nothing is done in wlan0 if hostapd is already started).

Here's my br0 script:

```
#!/bin/bash
set -o pipefail
PATH=/usr/local/sbin:/usr/sbin:/sbin:/usr/local/bin:/usr/bin:/bin

INAME=$(basename $0 |awk -F. '{print $NF}')
BR_NICS="eth0 wlan0"

do_start ()
{ echo "Start $INAME : $(date "+%F %H:%M:%S") $$"
  brctl addbr $INAME
  brctl stp $INAME off #turn off spanning tree
  for NIC in $BR_NICS
  do
    [ -x /etc/rc.d/network/$NIC ] && /etc/rc.d/network/$NIC start 2>&1
    sleep 1
    brctl addif $INAME $NIC
  done
  ifconfig $INAME 10.1.0.1 netmask 255.255.254.0 up
  dnsmasq -C /etc/dnsmasq/${INAME}.conf
}

do_stop ()
{ echo "Stop $INAME : $(date "+%F %H:%M:%S") $$"
  PID=$(ps -eo pid,cmd |grep dnsmasq | grep  -w "${INAME}\.conf" |awk
'{print $1}')
  [ "$PID" != "" ] && kill -9 $PID
  ifconfig $INAME 10.1.0.1 down
  for NIC in $BR_NICS
  do
    brctl delif $INAME $NIC
    [ -x /etc/rc.d/network/$NIC ] && /etc/rc.d/network/$NIC stop
  done
  brctl delbr $INAME
}

case $1 in
  start) do_start |tee -a /run/${INAME}.log ;;
  stop) do_stop |tee -a /run/${INAME}.log ;;
  restart) do_stop |tee -a /run/${INAME}.log ; do_start |tee -a
/run/${INAME}.log ;;
  status) brctl show
    for NIC in $BR_NICS
    do
      [ $(grep -wc "${NIC}:" /proc/net/dev) -gt 0 ] && ifconfig $NIC |grep -
Ew "^${NIC}|ether" || echo "${NIC}: not found"
```

```
    done
    ifconfig $INAME
    ps -ef | grep -v grep | grep "dnsmasq/${INAME}.conf"
  ;;
esac
```

If you've gone so far you might want a neater way to deal with unplugging dongles as anything udev could possibly do would be too late. One possible approach is to use gpio pins with buttons attached to them and a script that monitors the button status and takes appropriate action when each button is pressed. I've done this on mi RPi2 with 5 buttons and 5 leds indicating whether the button press has been caught by the monitor script: one shuts down the RPI2 while the other 4 are for deactivating whatever is in the respective usb port to prepare it for unplugging. Here's a picture of such a setup while the script for handling it is here.

8 Setting up on an Embedded Device

Supposing you want all this but you don't want to leave a laptop or desktop on all the time you might want to put in on an embedded ARM system that will only use a fraction of the power required to run an X86 laptop/desktop of any sort. (intel Curie might kick in with a 2.2W x86 SOC when the segfault bug is sorted out). Well the official ARM Slackware port userland runs on almost any ARM device out in the market today. There is a number of machines that are officially supported from the ARM port and a god deal of community supported efforts for the platforms that are not officially supported.

In terms of space you can fit all that's required in 151 Mb with a little careful stripping of unnecessary locales and documents. The best way to go about that is to add to the miniroot all the packages you need and then strip unnecessary stuff. Starting from a full install and the stripping unnecessary stuff will take a bigger effort.

Al the configuration so far will be exactly the same even if you're running the Slackware ARM port (actually since I did most of my testing on an ARM device it's more likely that there be an issue on the x86 platform).

On an embedded device there are however some issues that you need to address:

1. preserving the life of the flash mass storage
2. secure the system against unexpected reboots

1) By default the access time of any file gets automatically updated upon access of any file, this is a bad thing to do on flash that only has a limited number of rewrite cycles (typically 10,000 for MLC and 100,000 for SLC devices). Make sure you mount any filesystems on flash devices with the “noatime” option to prevent the access time modification. This is what the root entry in fstab could look like if you mount it with the noatime option (I use labels buy you can also use UUID or just plainly the device file)

```
LABEL=root      /               ext4        defaults,noatime         1   1
```

There are some generic filesystem type that may do better then other at handling flash devices and some that were specifically designed for that purpose but only run on native flash layer (not an option on usb flash sticks) like cffs2 and ubifs. There are also other options I've just shown 2 common options

that allow mounting rw if required.

2) Being on all the time your embedded device will crash every time uncontrollable power failures occur (whether it be a general powe failure, wife using too much power and causing the general switch to trip, your kids play unplug dad's stuff or a hard reset required because your AP crashed). Although journal-ed filesystems are robust with regards to that on the long run bad things may occour. What you really want to eliminate the problem altogether is leave your filesystems on flash read only (ro): Slackware was not conceived to run with root mounted ro but with a little tinkering in the init scripts you can work around that problem. Also having the filesystems ro will be even better then avoiding access time updating. You can remount temporarily with Read Write (rw) any time you need to make permanent changes then remount ro once you've finished. With the system running ro and only temporary files in ram filesystems all that can get lost across reboots are logs and temporary files ... but your system will never suffer catastrophic filesystem failures or prompt for interactive fsck during boot.

Here are some changes I frequently make to /etc/rc.d/rc.S to leave root mounter ro:

- reconfigure all I can to use /run instead of /var
- move /etc/{mtab,resolv.conf,ld.so.cache} in /run/etc/ and make links that point to where I moved them
- move /var/log/{messages,syslog,lastlog,wtmp,btmp,secure,dmesg} in /run/log/ and make links that point to where I moved them
- change the line that mounts root rw “/sbin/mount -w -v -n -o remount /” to something like this “/sbin/mount -v -n -o remount /”
- save the random-seed to unused sector on flash when shutting down (rc.0 ... well it's a link to rc.6) and load it back to /dev/urandom from rc.S when system comes up again

You could optionally have fstab specify that root should be mounted ro like this:

/dev/ubi0_0 / ubifs ro 0 1

You will also haveto make some links in various places so that they end up writing in /run that has been mounted tmpfs. I do that manually just once with / mounted rw and then remount it ro.

If you're interested in actually making such changes to the init scripts I've shared how I go about it on Linux Questions.

9 Unexpected Trouble Sources

When you start doing your own stuff you may run into problems that are hard to debug because the cause is pretty much unexpected.

Here's an example of one that recently gave me a headache over a long weekend after migrating from the seagate dockstar to RPi2. I had previously tested rigorously the functionality of everything I had changed in the setup and even back-ported some stuff to the dockstar to make sure it was stable. All seemed fine but when I finally switched to the Pi I started getting plagued with unstable internet connection. I fiddled endlessly with the peer setup without any improvement. I called a friend that uses the same ISP and he told me he had no issues so whatever I was experiencing I was the cause of

my own problems. During all my tests on both RPi and RPi2 I had never run into issues and nothing had changed that had not been previously tested, what on earth was going on ? It then dawned on me that I had done all my testing on the new hardware running from a USB power bank and that maybe there was some sort of parasitic coupling between the power supply I was now using and the modem itself. Sure enough the problems dissipated as I powered the PRi2 off the power bank again. I then pulled out a better quality power supply for the Pi and everything still worked fine.

It was that lousy wall charger, that I took off a really low end tablet my son broke, that was making enough noise to bother my modem. Had it been a FCC or CE certified power supply that would probably not have happened.

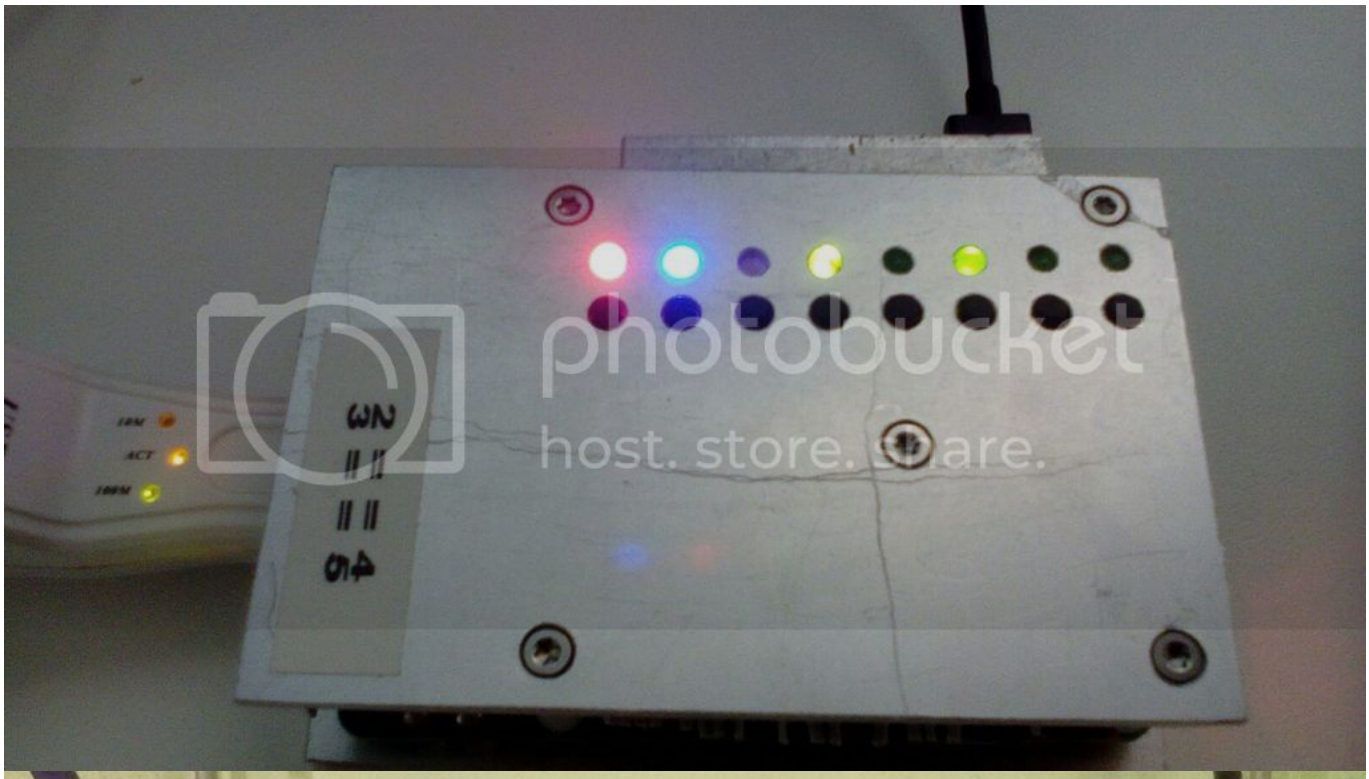
Obviously there may be innumerable other trouble source you may run into (some of which you have no control … like when your ISP makes clumsy NAT for you) but if you're doing your own stuff be prepared to put up and debug them.

10 Conclusions

You have set up an AP with entry level security and the most common options available in the low end appliances but you have gained:

- ability to easily keep your AP software up-to date
- ability to run several AP at the same time (and even mesh into other WiFis … not covered in this howto)
- plug and network capability for the usb dongles
- full control over the way interfaces are bridged and/or the way traffic is routed trough the AP
- full control on the way dhcp works (limitless subnets and address reservation)
- full control over fire-waling (no limits in port forwarding, MAC acl, DMZ ot subnets with specific set of permissions and quotas)
- ability to re-share ANY type of connection the AP has
- ability to choose what and how to backup in the AP
- ssh access for management with a standard Linux environment for easy debugging issues
- scale up/down the hardware that runs the AP without having to change the way you setup things
- update components for which new vulnerabilities have been found without having to wait the the manufacturer does something about it
- whatever the flexibility of a standard Linux environment can give you that a custom crippled embedded environment cannot.

I'm not going to miss my old low end AP, are you ? I replaced it with home-brew stuff from which this article is derived. Over the years I've done several remakes adding some new features. The last upgrade was 8 programmable buttons with indicator leds to carry out arbitrary functions.


photobucket
host. store. share.


photobucket
host. store. share.

If anyone is interested in a nearly ready to go Raspberry Pi 1/2 image of what has been discussed in this article you cab get if from here. Just extract tar and do “cat rpi_wrap_512mb.img > <your SD device>” into an SD at least 512Mb, boot it and longin via the console, read readme.txt and complete the setup manually. The readme.txt is also in the tarball so you can read it before even logging into the Pi.

Sources

- Originally written by louigi600

howtos, louigi600