

FILTERING APPLTALK USING EBTABLES

Why filter AppleTalk?

There are many situations where it is appropriate to filter AppleTalk. Here's one of them. We [tunnel/route](#) AppleTalk between five networks using [netatalk](#). There are very similarly named Tektronix Phaser printers in two of these networks, and often print jobs intended for one are unintentionally sent to the other. We would prefer for each of these Phasers to be visible only in the network in which it is located, and not in all five networks. Unlike [CAP](#), netatalk does not support filtering. Therefore, on this page I describe one method to add external filters to netatalk, on the basis of the MAC address associated with an AppleTalk object or node.

There are pros and cons to filtering on the basis of MAC addresses. They have the advantage of being more robust because AppleTalk node numbers can change with every reboot, while the MAC address will not. They have the disadvantage of not being fine-grained; MAC-based filtering will block all the services associated with the filtered AppleTalk node. In general, AppleTalk nodes in our networks are associated with a single service.

Iptables versus Ebtables

The Linux [netfilter](#) code supports filtering of IPV4, IPV6 and DECnet packets on the basis of MAC addresses. However such filters do not apply to any other type of ethernet frame. Thus, an *iptables* rule such as:

```
iptables -I INPUT -m mac --mac-source TE:KP:HA:SE:R8:60 -j DROP
```

results in only IPV4, IPV6 and DECnet packets from that source address being dropped. More to the point, DDP and AARP packets from the same source address are not dropped. [Ebtables](#) appeared to be perfectly suited to filter ethernet frames on the basis of MAC address as well as ethernet protocol type. However, it only supports bridge interfaces, and not regular ethernet interfaces. [Bart De Schuymer](#), the author of ebtables brought to my attention that a Linux bridge interface can have just a single ethernet interface. Thanks to Bart's generous advice, a working ethernet filtering setup is described below. My instructions relies on the reader being familiar with applying Linux kernel patches, and compiling a custom Linux kernel. The Linux [Kernel HOWTO](#) provides detailed instructions on the ins and outs of compiling kernels.

Setting up Ebtables

Download the source to the latest kernel supported by the the latest bridge-nf-bds patch, and ebtables patch available from Bart de Schuymer's [site](#). Compile the kernel with support for AppleTalk, netfilter (iptables), and 802.1d Ethernet Bridging. Under 802.1d Ethernet bridging, at least the options "Bridge: ebtables", "ebt: filter table support", and "ebt: LOG support" should be selected. Once compiled, install and boot using this new kernel that supports ethernet bridging and ebtables.

Second, the *ebtables* administration tool should be compiled and installed. *Ebtables* can also be downloaded from Bart De Schuymer's [site](#).

Finally, the Linux bridge-utils should be compiled and installed. These are available from the Linux [ethernet bridging](#) site. Do not install the bridge-nf kernel patches available from this site, as they are incompatible with Bart's ebtables kernel patches.

To setup a bridge with a single interface, first create the bridge interface (br0). Then add the relevant ethernet interface to the bridge. Finally, assign to the bridge the IP address previously assigned to the ethernet interface. The commands to do this are detailed below:

```
brctl addbr br0                # create bridge interface
brctl stp br0 off              # disable spanning tree protocol on br0
brctl addif br0 eth0           # add eth0 to br0
ifconfig br0 aaa.bbb.ccc.ddd netmask 255.255.255.0 broadcast aaa.bbb.
ccc.255
ifconfig eth0 0.0.0.0
route add -net aaa.bbb.ccc.0 netmask 255.255.255.0 br0
route add default gw aaa.bbb.ccc.1 netmask 0.0.0.0 metric 1 br0
```

Now network traffic will be routed through the br0 interface rather than the underlying eth0. *Atalkd* can be started to route AppleTalk between br0 and any other desired interfaces. Note that *atalkd.conf* has to be modified so that the reference to eth0 is replaced with br0. For example, the *atalkd.conf* for PC1 shown on my [AppleTalk tunneling page](#) is modified to:

```
br0 -seed -phase 2 -net 2253 -addr 2253.102 -zone "Microbio-Immun"
tap0 -seed -phase 2 -net 60000 -addr 60000.253 -zone "Microbio-Immun"
tap1 -seed -phase 2 -net 60001 -addr 60001.253 -zone "Microbio-Immun"
tap2 -seed -phase 2 -net 60002 -addr 60002.253 -zone "Microbio-Immun"
tap3 -seed -phase 2 -net 60003 -addr 60003.253 -zone "Microbio-Immun"
```

Verify that AppleTalk routing is working, and then proceed to set up ethernet filters using *ebtables*. For this the MAC addresses of the AppleTalk nodes that are not to be routed must be known. One simple method of discovering the MAC address is to send the AppleTalk object a few *aecho* packets, and then read the MAC address from `/proc/net/aarp`. A sample *ebtables* filter is shown below:

```
ebtables -P INPUT ACCEPT
ebtables -P FORWARD ACCEPT
ebtables -P OUTPUT ACCEPT
ebtables -A INPUT -p LENGTH -s TE:KP:HA:SE:R8:60 -j DROP
```

Currently, *ebtables* doesn't support filtering of 802.2 and 802.3 packets such as the DDP and AARP packets used by AppleTalk. However all such packets can be dropped on the basis of the length field -- if

I understand Bart de Schuymer's explanation correctly. Therefore in the example above, all ethernet 802.2, 802.3 packets from the node with the MAC address "TE:KP:HA:SE:R8:60" are dropped. This includes AppleTalk packets, but not IPV4, and ARP packets. This node is left visible in the network in which it is located, but not in any networks to which AppleTalk is routed.

Acknowledgements, Final Comments and Useful Links:

Bart de Schuymer's advice and patient explanations are greatly appreciated. In my experience *atalkd* bound to the br0 interface is as stable as *atalkd* bound to the eth0 interface. In addition the MAC address based filters described here work well for their intended purpose. While this works, there is a performance penalty associated with receiving all IP traffic through br0 and not eth0. This is because traffic destined for the bridge is queued twice (once more than normal) -- that's a lot of overhead. The ebtables *broute* table can be used to circumvent this and directly route the traffic entering the bridge port. This way it will be used only once, eliminating the performance penalty. In the example above:

```
brctl addbr br0
brctl stp br0 off
brctl addif br0 eth0
ifconfig br0 0.0.0.0
ifconfig eth0 a.b.c.d netmask 255.255.255.0 broadcast a.b.c.255
```

The following two ebtables BROUTE table rules should be used:

```
ebtables -t broute -A BROUTING -p IPv4 -i eth0 --ip-dst a.b.c.d -j
DROP
ebtables -t broute -A BROUTING -p ARP -i eth0 -d MAC_of_eth0 -j DROP
```

Atalkd should still be bound to br0, thus allowing AppleTalk to be filtered by ebtables. As best as I can tell this configuration eliminates the performance penalty on IP traffic throughput. Because we [tunnel AppleTalk](#) through IP, this configuration removes any throughput penalties in using a bridge interface and ebtables to route AppleTalk.

External links:

- My instructions for [AppleTalk tunneling](#).
- Bart de Schuymer's [Ebtables site](#).
- Linux [Ethernet bridging site](#).

[Ashok Aiyar](#)

Last modified: Sun Aug 4 12:54:12 CDT 2002

