

Re: resend: multiple routing table roadmap (format fix)

Source: <http://unix.derkeiler.com/Mailing-Lists/FreeBSD/net/2007-12/msg00303.html>

- *From:* Julian Elischer <julian@xxxxxxxxxxxxx>
 - *Date:* Thu, 27 Dec 2007 13:19:08 -0800
-

Ivo Vachkov wrote:

On Dec 27, 2007 2:26 AM, Julian Elischer <julian@xxxxxxxxxxxxx> wrote:

Resending as my mailer made a dog's breakfast of the first one with all sorts of wierd line breaks... hopefully this will be better. (I haven't sent it yet so I'm hoping)..

On thing where FreeBSD has been falling behind, and which by chance I have some time to work on is "policy based routing", which allows different packet streams to be routed by more than just the destination address.

Constraints:

I want to make some form of this available in the 6.x tree (and by extension 7.x) , but FreeBSD in general needs it so I might as well do it in -current and back port the portions I need.

One of the ways that this can be done is to have the ability to instantiate multiple kernel routing tables (which I will now refer to as "Forwarding Information Bases" or "FIBs" for political correctness reasons. Which FIB a particular packet uses to make the next hop decision can be decided by a number of mechanisms. The policies these mechanisms implement are the "Policies" referred to in "Policy based routing".

One of the constraints I have if I try to back port this work to 6.x is that it must be implemented as a EXTENSION to the existing ABIs in 6.x so that third party applications do not need to be recompiled in timespan of the branch.

Re: resend: multiple routing table roadmap (format fix)

Implementation method, (part 1)

For this reason I have implemented a "sufficient subset" of a multiple routing table solution in Perforce, and back-ported it to 6.x. (also in Perforce though not yet caught up with what I have done in -current/P4). The subset allows a number of FIBs to be defined at compile time (sufficient for my purposes in 6.x) and implements the changes needed to allow IPV4 to use them. I have not done the changes for ipv6 simply because I do not need it, and I do not have enough knowledge of ipv6 (e.g. neighbor discovery) needed to do it.

By the way, I might add that in the 6.x compat. version I may end up limiting the feature to 8 tables. This is because I need to store some stuff in an efficient way in the mbuf, and in a compatible manner this is easiest done by stealing the top 4 bits in the mbuf dlags word and defining them as:

```
#define M_HAVEFIB 0x10000000
#define M_FIBMASK 0x07
#define M_FIBNUM 0xe0000000
#define M_FIBSHIFT 29
#define m_getfib(_m, _default) ((m->m_flags & M_HAVE_FIBNUM) ? ((m->m_flags >> M_FIBSHIFT) &
M_FIBMASK) : _default)
#define M_SETFIB(_m, _fib) do { \
    _m->m_flags &= ~M_FIBNUM; \
    _m->m_flags |= (M_HAVEFIB|((_fib & M_FIBMASK) << M_FIBSHIFT)); \
} while (0)
```

This then becomes very easy to change to use a tag or whatever is needed in later versions, and the number can be expanded past 8 predefined FIBs at that time..

Other protocol families are left untouched and should there be users with proprietary protocol families, they should continue to work and be oblivious to the existence of the extra FIBs.

To understand how this is done, one must know that the current FIB code starts everything off with a single dimensional array of pointers to FIB head structures (One per protocol family), each of which in turn points to the trie of routes available to that family.

The basic change in the ABI compatible version of the change is to extent that array to be a 2 dimensional array, so that instead of protocol family X looking at `rt_tables[X]` for the table it needs, it looks at `rt_tables[Y][X]` when for all protocol families except ipv4 Y is always 0. Code that is unaware of the change always just sees the first row

Re: resend: multiple routing table roadmap (format fix)

of the table, which of course looks just like the one dimensional array that existed before.

Pretty much like the OpenBSD approach :)

well, I did look at the code briefly, but I didn't base it on it..

The entry points `rtrequest()`, `rtalloc()`, `rtalloc1()`, `rtalloc_ign()` are all maintained, but refer only to the first row of the array, so that existing callers in proprietary protocols can continue to do the "right thing". Some new entry points are added, for the exclusive use of ipv4 code called `in_rtrequest()`, `in_rtalloc()`, `in_rtalloc1()` and `in_rtalloc_ign()`, which have an extra argument which refers the code to the correct row.

In addition, there are some new entry points (currently called `dom_rtalloc()` and friends) that check the Address family being looked up and call either `rtalloc()` (and friends) if the protocol is not IPv4 forcing the action to row 0 or to the appropriate row if it IS IPv4 (and that info is available). These are for calling from code that is not specific to any particular protocol. The way these are implemented would change in the non ABI preserving code to be added later.

One feature of the first version of the code is that for ipv4, the interface routes show up automatically on all the FIBs, so that no matter what FIB you select you always have the basic direct attached hosts available to you. (`rtinit()` does this automatically).

You CAN delete an interface route from one FIB should you want to but by default it's there. ARP information is also available in each FIB. It's assumed that the same machine would have the same MAC address, regardless of which FIB you are using to get to it.

This brings us as to how the correct FIB is selected for an outgoing IPV4 packet.

Packets fall into one of a number of classes.
1/ locally generated packets, coming from a socket/PCB.
Such packets select a FIB from a number associated with the socket/PCB. This in turn is inherited from the process, but can be changed by a socket option. The process in turn inherits it on fork. I have written a utility call `setfib` that acts a bit like `nice`..

Re: resend: multiple routing table roadmap (format fix)

setfib -n 3 ping target.example.com # will use fib 3 for ping.

2/ packets received on an interface for forwarding.
By default these packets would use table 0,
(or possibly a number settable in a sysctl(not yet)).
but prior to routing the firewall can inspect them (see below).

3/ packets inspected by a packet classifier, which can arbitrarily
associate a fib with it on a packet by packet basis.
A fib assigned to a packet by a packet classifier
(such as ipfw) would over-ride a fib associated by
a more default source. (such as cases 1 or 2).

For the 2/ and 3/ cases I added (in a personal work i've been doing
lately) additional field in struct mbuf which can be set by a packet
filter or other application upon receiving which points the right
table to use for the lookup. This way a simple "marking" can be used
to divide different flows and create policy based routing.

This would be the final way but I want to really minimise problems
in the compat versions, so I'll avoid doing that for now.

Do you have this work available?
And have you looked at mi diffs below?

Routing messages would be associated with their
process, and thus select one FIB or another.

In addition Netstat has been edited to be able to cope with the
fact that the array is now 2 dimensional. (It looks in system
memory using libkvm (!)).

In addition two sysctls are added to give:
a) the number of FIBs compiled in (active)
b) the default FIB of the calling process.

Early testing experience:

Basically our (IronPort's) appliance does this functionality already
using ipfw fwd but that method has some drawbacks.

For example,
It can't fully simulate a routing table because it can't influence the
socket's choice of local address when a connect() is done.

Re: resend: multiple routing table roadmap (format fix)

Testing during the generating of these changes has been remarkably smooth so far. Multiple tables have co-existed with no notable side effects, and packets have been routes accordingly.

I have not yet added the changes to ipfw. pf has some similar changes already but they seem to rely on the various FIBs having symbolic names. Which I do not plan to support in the first version of these changes.

SCTP has interestingly enough built in support for this, called VRFs in Cisco parlance. it will be interesting to see how that handles it when it suddenly actually does something.

I have not redone my testing since my last edits, but will be retesting with the current code asap.

Where to next:

After committing the ABI compatible version and MFCing it, I'd like to proceed in a forward direction in -current. this will result in some roto-tilling in the routing code.

Firstly: the current code's idea of having a separate tree per protocol family, all of the same format, and pointed to by the 1 dimensional array is a bit silly. Especially when one considers that there is code that makes assumptions about every protocol having the same internal structures there. Some protocols don't WANT that sort of structure. (for example the whole idea of a netmask is foreign to appletalk). This needs to be made opaque to the external code.

My suggested first change is to add routing method pointers to the 'domain' structure, along with information pointing the data. instead of having an array of pointers to uniform structures, there would be an array pointing to the 'domain' structures for each protocol address domain (protocol family), and the methods this reached would be called. The methods would have an argument that gives FIB number, but the protocol would be free to ignore it.

Interaction with the ARP layer/ LL layer would need to be revisited as well. Qing Li has been working on this already.

diffs
for those with p4 access:

Re: resend: multiple routing table roadmap (format fix)

```
p4 diff2 -du //depot/vendor/freebsd/src/sys/...@131121
//depot/user/julian/routing/src/sys/...
```

for those with the makediff perl script:
perl ~/makediff.pl //depot/vendor/freebsd/src/sys/...@131121
//depot/user/julian/routing/src/sys/...

for those with neither:

<http://people.freebsd.org/~julian/mrt2.diff>

I just put the userland utility in usr.sbin/setfib/ in p4.
and changes to netstat in usr.bin/netstat/

see:

<http://perforce.freebsd.org/depotTreeBrowser.cgi?FSPC=//depot/user/julian/routing/src&HIDEDEL=N>

I'd like to get comments on this (compat) version, so that I can
commit it,
get general testing under way to start the clock for MFC, and then get
moving on the fuller implementation (that breaks ABIs) and other
routing issues.

Julian

freebsd-arch@xxxxxxxxxxxxx mailing list
<http://lists.freebsd.org/mailman/listinfo/freebsd-arch>
To unsubscribe, send any mail to
"freebsd-arch-unsubscribe@xxxxxxxxxxxxx"

freebsd-net@xxxxxxxxxxxxx mailing list
<http://lists.freebsd.org/mailman/listinfo/freebsd-net>
To unsubscribe, send any mail to "freebsd-net-unsubscribe@xxxxxxxxxxxxx"