NETWORK STACK AS A MODULE

9 Jun 2016 – BSDCan 2016

Steve Kiernan
Network Stack as a Module

- Overview
- Issues moving to a module
- Existing layering violations
- What is available
- What needs to be done
- Wish list
OVERVIEW
Overview

Juniper has its own network stack that has grown up from the FreeBSD 2.x days and diverged onto its own path.

With the project to move Junos from FreeBSD 6.x as its base to FreeBSD 10-STABLE and beyond, Juniper wanted to utilize a minimally-modified FreeBSD kernel.

This required proper separation of Junos and FreeBSD pieces.
Overview

Challenges to doing that (prior to network stack module):

1. Not easy to build a kernel without network support
2. No existing method to choose between the FreeBSD network stack or the Junos network stack
3. Many layering violations in between the common FreeBSD kernel code and the network stack code
4. Required NIC driver changes to work with the Junos network stack
Overview

What was done
1. DrvAPI for NIC drivers
2. Create a network stack module (netstack.ko)
3. Eliminate layering violations
4. Add interface between common kernel and network stack
Overview

Juniper can boot a FreeBSD image or Junos image on our hardware

• FreeBSD image contains the FreeBSD network stack as a separate package
• Junos image is a superset of the FreeBSD image
• Junos image contains the Junos network stack as a separate package
• Network stack is dynamically loaded by loader at boot based on the network stack package that is active
Overview

Why is this important to FreeBSD?

• Allows for a kernel without network stack
• Makes it easier for experimenting with new network stack implementations
• Potential for “interesting” combinations of network stacks
ISSUES MOVING TO A MODULE
Issues Moving To a Module

- Protocol domains can only be registered early
- Number of layering violations between the common kernel code and the network stack
  - Jails have intimate knowledge of network stack details
  - Mount exports have assumptions about the network stack
- VNET virtualized global variables and modules (still a problem?)
EXISTING LAYERING VIOLATIONS
Existing Layering Violations

- sys/kern has knowledge of network stack functions
  - socket ioctl groups
    - route
    - Interface
  - Jails
    - IPv4/IPv6 specific data and calls
  - FIBs
  - Mount exports (NFS)
  - Prot
  - Accept filters
Previous Layering Violations

- sys/kern has knowledge of network stack functions
  - SCTP system calls moved to netinnet and committed
- zlib code in sys/net
  - Moved to libkern (and sys/sys) and committed
WHAT IS AVAILABLE
What Is Available

Work being done in my GitHub repo
• https://github.com/hackagadget/freebsd.git
• hackagadget/netstack branch

Caveat: not all original changes pushed up to GitHub repo (yet)

Original patch from 2013 can be found here:
https://people.freebsd.org/~marcel/Juniper/netstack-v2.diff
What Is Available

Additions

• Network stack option (NETSTACK)
• Netstack interface (netstack_if.m)
  • Currently used to handle exports list interaction
  • Also able to inform the network stack about socket creations
    • Sets so_fibnum based on FIB set on process
    • Expected to be the interface between the network stack and the core kernel code
• GENERIC kernel without network stack (NONETSTACK)
What Is Available

Additions

• Kernel module built
  • Must be loaded early (by loader or equivalent)
What Is Available

Some of the layering violations have been fixed

• socket ioctl groups
  • Network stack can register groups
  • Interface group moved to sys/net/if.c
  • Route group moved to sys/net/route.c

• Jails
  • Moved qcmp_{v4,v6} and prison_*_{v4,v6} functions to sys/netinet/in_jail.c and sys/netinet6/in6_jail.c and adjusted names
What Is Available

Some of the layering violations have been fixed

- **FIBs**
  - SO_SETFIB socket calls handled by protocol pr_ctloutput
  - RT_SOSETFIB macro added to make it easy for pr_ctloutput routines to handle setting so_fibnum on the socket structure

- **Mount exports**
  - Using the netstack_if.m interface
    - vfs_export, vfs_setpublicfs, vfs_stdcheckexp
What Is Available

Some of the layering violations have been fixed

• Prot
  • Moved cr_canseeinpcb to sys/netinet/in_prot.c
• Accept filters
  • Rename net.inet.accf sysctl$s to net.accf
  • Retain net.inet.accf node for compatibility
WHAT NEEDS TO BE DONE
What Needs To Be Done

• Get current code reviewed and committed
• Improve export list handling
• Jail code still has knowledge of IPv4/IPv6
  • Would be good to make jails protocol agnostic
• Domain registration should be allowed at any time
  • Also de-registration would be needed in order to dynamically unload modules
What Needs To Be Done

- VNET support
- Expanding the netstack_if.m interface
- Any other changes required to dynamically load the netstack module after the system is booted
- Allow for unloading the netstack module
- Take care of compat code (Linuxulator, SVSR4, etc.)
WISH LIST
Wish List

• Protocol modules
  • Allow for dynamically loading protocol modules
  • Requires changes to ifconfig, netstat, etc. to be aware of the network stack they are talking to
    • Also means having libraries to handle stack-specific details that utilities can use

• On-demand loading of protocol modules
  • Similar to module loading done by mount for file systems
Wish List

• Migrate connections to new netstack
  • Load new netstack module
  • Active connections owned by new module
  • Unload old netstack module

• Per-jail netstack instance
  • Maybe even per-process? (Any need?)
Thank you