# New Networking Features in FreeBSD 6.0

André Oppermann
*andre@FreeBSD.org*

The FreeBSD Project

**Abstract**

FreeBSD 6 has evolved drastically in the development branch since FreeBSD 5.3 [1] and especially so in the network area. The paper gives an in-depth overview of all network stack related enhancements, changes and new code with a narrative on their rationale.

## 1 Internal changes – Stuff under the hood

### Mbuf UMA

UMA (Universal Memory Allocator) is the FreeBSD kernels primary memory allocator for fixed sized data structures. It is a SLAB type allocator, fully SMP aware and maintains per-CPU caches of frequently used objects. All network data is stored in Mbufs of 256 bytes and Mbuf clusters of 2048 bytes which can be attached to Mbufs and replace their internal data storage. When a cluster is attached the Mbuf serves as descriptor for the cluster containing all associated Mbuf and packet information for the kernel and protocols. To use UMA for efficient Mbuf allocation some enhancements have been made to it. Most important is the packet secondary zone holding pre-combined Mbuf+cluster pairs. This allows protocols to save one memory allocation by directly obtaining a large data structure instead of allocating an Mbuf and then attaching a separately allocated Mbuf cluster. The secondary zone is special as it is only a cache zone and does not have its own backing store. All mbuf+cluster combinations in it come from their own original Mbuf and cluster zones. Mbuf UMA provides good SMP scalability and an accelerated allocation path for frequently used Mbuf+cluster pairs. *For more information see [2], mbuf(9) and uma(9).*

### SMP Locking

SMP locking of network related data structures is the main theme of FreeBSD 6. Locking is necessary to prevent two CPUs accessing or manipulating the same data structure at the same time. Locking gives exclusive access to only one CPU at a time and makes them aware of each others work – it prevents CPUs from stomping on each others feet. Generally it is desirable to break down locking into fine-grained portions to avoid lock contention when multiple CPUs want to access related but independent data structures. On the other hand too fine-grained locking is introducing overhead as each locking and unlocking operation has to be reliably propagated to all other CPUs in the system. The first go on fine-grained network locking in FreeBSD 5 has been greatly enhanced and refined for excellent SMP scalability. For single processor machines all performance regressions due to locking overhead have been eleminated and FreeBSD 6 reaches the same speed in heavy duty network streaming as the FreeBSD 4 series. *For more information see [3], mutex(9) and witness(4).*

### Socket Buffer Locking

Every active or listening network connection is represented as a socket structure in the kernel. The socket structure contains general bookkeeping on the socket and two socket buffer for transmitted and received packets. Protocols (TCP, UDP, IPX, SPX, etc.) extend the socket structure with their own bookkeeping to track connections state and other vital information. Many of these structures are linked forth and back and among each other. This makes proper locking complicated. Additionally the socket data structure may be accessed and manipulated at any time either from an application writing, reading or closing the socket or from the kernel itself when it has received data, retransmits or error messages for that socket. FreeBSD 6 implements a multi-level locking strategy to efficiently cope with these constrains. Each socket structure has a general lock and two separate send and receive socket buffer locks. Thus sending and receiving may happen concurrently. Any operation that changes the state of the entire socket (ie. connection tear down) has to obtain the general lock. On the protocol side (using TCP as example) two more locks are embedded. One protects the IN and TCP control blocks which contain IP protocol and TCP specific information, such as the addresses of the end points and the state of the TCP connection. The other lock protects all IN control blocks as a whole. Locks with such a global scope are normally frowned upon but here it is necessary to prevent changes in the control blocks while searches and lookup's are performed on it. A search and lookup happens every time a packet is received from the network. While this is not optimal it has shown to express only modest contention.

### Protocol Locking

Since early 2005 the entire network stack is running without any global and exclusive lock. All Internet protocols and IPX/SPX have been individually locked and thus made fully SMP aware and scalable.

### Network Interface Structure Locking

An area of particular concern for proper locking was the ifnet structure. The ifnet structure contains all information the kernel knows about network interfaces. In FreeBSD network interfaces drivers may be loaded and unloaded any time as KLDs (Kernel Loadable Modules) or may arrive or depart as hot-plug interfaces like PCCARDs in laptops. Allowing these actions to occur in a SMP-safe way has required significant work and re-work of the ifnet structure and its modes of access. For example some fields in the structure were holding flags manipulated by the network stack and the driver. Each of them had to obtain the lock for the full structure to change its own fields and flags. This lead to contention and limitations on parallelism for access to the physical network. Any such unnecessary contention point has been identified and each party has got their own field which they can manipulate independently without stalling the other. *For more information see ifnet(9), netintro(9), [4] and [5].*

## 2 Netgraph

Netgraph is a concept where a number of small, single-job modules are stringed together to process packets through stages. Many modules may be combined in almost arbitrary ways. Netgraph may be best explained as an assembly line with many little functions along a conveyor belt versus one big opaque machine doing all work in one step. As part of the network stack netgraph has received fine grained locking too. Depending on the function and task of the module it was either locked as whole or every instance of it separately. *For more information see netgraph(4), netgraph(3) and ngctl(8).*

### Module ng_netflow

Ng_netflow is a new module for accounting of TCP and UDP flows in ISP (Internet Service Provider) backbones. It accumulates statistics on all TCP and UDP session going through the machine and once one has finished (FIN or RST

for TCP) sends a UDP packet in the Netflow 5 format to a statistics collector for further processing. The node can either run in parallel to the normal IP forwarding and packet processing, in this case it gets a copy of every packet, or all packets are passed through it unmodified. *For more information see ng_netflow(4)*.

Module ng_ipfw

Ng_ipfw is a new module providing a way for injecting arbitrary matched packets into netgraph using ipfw. It works very much like an ipfw divert rule diverting the packet to netgraph instead of a divert socket. This allows, for example, to send by ipfw filtered or rejected packets to netgraph for further analysis or to capture certain types of IP packets for further netgraph manipulations. The packet matching capabilities of ipfw are very powerful in this context. *For more information see ng_ipfw(4), ipfw(8) and ipfw(4)*.

Module ng_nat

Ng_nat is a new module providing netgraph access to the kernel-level libalias for network address translation. Libalias used to be a userland-only application library but was written with in-kernel use in mind. For ng_nat is got imported into the kernel. *For more information see ng_nat(4) and libalias(3)*.

Module ng_tcpmss

Ng_tcpmss is a new module changing the MSS (Maximum Segment Size) optoin of TCP SYN packets. Many broadband users are behind DSL lines with a reduced MTU (Maximum Transmission Unit) of 1492 bytes. The normal size for ethernet is 1500 bytes. If a packet does not fit the MTU of link is has to be fragmented – it gets split into two packets. This is a CPU intensive process and to be avoided if possible. Normally the TCP path MTU discovery mechanism is supposed to automatically detect smaller MTUs along the way but over-zealous firewall administrators often block the ICMP MTU adjustment messages. As workaround a router along the path of the packet scans for TCP SYN packets and manipulates it to reduce the MSS to fit the lower MTU. *For more information see ng_tcpmss(4)*.

## 3 IPv4

DHCP Client

The most visible change is the new DHCP client. It is a port of the OpenBSD dhclient and adapted to FreeBSD specific needs. It has many security features like privilege separation to prevent spoofed DHCP packets from exploiting the machine. Additionally it is network interface link state aware and will re-probe for a new IP address when the link comes back up. This is very convenient for laptop users who may connect to many different networks, be it wired or wireless LANs many times a day. *For more information see dhclient(8), dhclient.conf(5) and dhclient.leases(5)*.

IPFW Firewall

IPFW has received many visible and invisible modifications. The most prominent visible changes are IPv6 rule support and ALTQ tagging of packets. The IPv6 support is further discussed in the IPv6 section. ALTQ is an alternative queuing implementation for network interfaces. Whenever an output interface doesn't have enough bandwidth to forward all waiting packets immediately queuing happens. Excess packets have to wait until earlier packets are drained and capacity is available again. Standard queuing strategy is a tail queue – all new packets get appended to the tail of the queue until the queue is full and any further packets get dropped. In many situations this is undesirable and for QoS (Quality of Service) it should treat various types of packets and traffic differently and with different priorities. ALTQ allows to define different queuing strategies on network interfaces to prioritize, for example, TCP ACKs on slow ADSL uplinks or delay and jitter sensitive VoIP (Voice over IP) packets. IPFW

can be used as packet classifier for ALTQ treatment. IPFW has another packet queue manager called DUMMYNET which can perform many of ALTQ function too. However it is more geared towards network simulations in research setting than to general network interface queuing. Under the hood of IPFW the stateful inspection of packet flows has been converted to use UMA zones for flow-state structure allocation. *For more information see ipfw(8), ipfw(4), altq(8), altq(9) and dummynet(4)*.

IPDIVERT

The IPDIVERT module is used for NAT (Network address Translation) with IPFW. It is now a loadable module that can loaded into the kernel at runtime. Before it always required a kernel re-compile to make it available. *For more information see divert(4)*.

IP Options

IP Options are a sore spot in the entire IPv4 specification. IP Options extend the IP header by a variable size of up to 40 bytes to request and record certain information from routers along the packets path. IP Options are seldom used these days and have essential zero legitimate use other than Record Route perhaps. IP Options handling in the kernel is complicated and was handled through a couple of global variables in the IP code path. Access to these variables had to be locked and it prevented multiple CPUs from working on IP packets in parallel. The global variables have been moved into mtags attached to mbufs containing IP packets with IP Options. This way all CPUs can work on IP packets in parallel without risk of overwriting information and the IP Options information always stays with the packet it belongs to. Even when one CPU hands off the packet to another CPU.

IPFILTER Firewall

IPFILTER 4.1.8 was imported and provides proper locking of its data structures to work in SMP environments. *For more information see ipf(8), ipf(5) and ipf(4)*.

NFS Network File System

NFS has been extensively tested and received numerous bug fixes for many edge cases involving file access as well as some network buffer improvements.

ICMP

ICMP Source Quench support has been removed as it is deprecated for a long time now. Source Quench was intended to signal overloaded links along a packet path but it would send one Source Quench message per dropped payload packet and thus increased the network load rather to reduce it. It is not and was never used in the Internet. *For more information see [6]*.

ICMP replies can now be sent from the IP address of the interface the packet came into the system. Previously it would always respond with the IP address of the interface on the return path. When the machine is used as a router this could give very misleading error messages and traceroute output. *For more information see icmp(4)*.

ARP Address Resolutions Protocol

Many ARP entry manipulation races got fixed. ARP maps an IPv4 address to a hardware (MAC) address used on the ethernet wire. It stores the IP address of each machine on all directly connected subnets as a host route and attaches their MAC address. ARP lookup's and timeouts can happen at any time and may be triggered at any time from other machines on the network. In SMP environments this has led to priority inversions and a couple of race conditions where one CPU was changing parts of an ARP entry when a second CPU tried to do the same. They clashed and stomped on each others work leading to incorrect ARP entries and even crashes sometimes. An extensive rework and locking has been done to make ARP SMP-safe.

IP Multicast

IP Multicast had many races too. Most of them related to changes of IP addresses on network interfaces and disappearing interfaces due to unload or unplug events. Proper locking and ordering of locks has been instituted to make IP Multicast SMP-safe.

IP Sockets

An IP_MINTTL socket option got added. The argument to this socket option is a value between 1 and 255 which specifies the minimum TTL (Time To Live) a packet must have to be accepted on this socket. It can be applied to UDP, TCP and RAW IP sockets. This option is only really useful when set to 255 preventing packets from outside the directly connected networks reaching local listeners on sockets. It allows userland implementation of 'The Generalized TTL Security Mechanism (GTSM)' according to RFC3682. Examples of such use include the Cisco IOS BGP implementation command "neighbor ttl-security". *For more information see ip(4) and RFC3682*.

The IP_DONTFRAG socket option got added. When enabled this socket option sets the Don't Fragment bit in the IP header. It also prevents sending of packets larger than the egress interface MTU with an EMSGSIZE error return value. Previously packets larger than the interface MTU got fragmented on the IP layer and applications didn't have a direct way of ensuring that they send packets fitting into the MTU. It is only implemented for UDP and RAW IP sockets. On TCP sockets the Don't Fragment bit is controlled through the path MTU discovery option. *For more information see ip(4)*.

## 4 TCP Transmission Control Protocol

SACK Selective ACKnowledgements

SACK has received many optimizations and interoperability bug fixes. *For more information see tcp(4)*.

T/TCP Transactional TCP

T/TCP support according to RFC1644 has been removed. The associated socket level changes however remain intact and functional. FreeBSD was the only mainstream operating system that ever implemented T/TCP and its intrusive changes to the TCP processing made code maintenance hard. It primary feature was the shortening of the three-way TCP handshake for hosts that knew each other. Unfortunately it did this in a very insecure way that is very prone to session spoofing and packet injection attacks. Use of it was only possible in well secured Intranets. It never enjoyed any widespread use other than on round trip time sensitive satellite links. A replacement is planned for FreeBSD 7.

TCP Sockets

The TCP_INFO socket option allows the retrieval of vital metrics of an active TCP session such as estimated RTT, negotiated MSS and current window sizes. It is supposed to be compatible with a similar Linux socket option but still experimental.

Security Improvements

The tcpdrop utility allows the administrator to drop or disconnect any active TCP connection on the machine. This tool was ported from OpenBSD. *For more information see tcpdrop(8)*.

The logic processing of TCP timestamps (RFC1323) has been improved to prevent spoofing attempts.

TCP Path MTU Discovery has been improved to prevent spoofing attacks. It now checks the entire TCP header that is quoted in the ICMP Fragmentation Needed message to ensure it matches to a valid and active TCP connection. *For more information see [5]*.

Port Randomization led to some problems when applications with very high connection rates

came close to exhaust the port number range. The randomization function was calculating random ports numbers which were most likely already in use and fell into an almost endless loop as the odds of finding a free port at random dropped constantly. If exhaustion is near it now switches to normal allocation for 45 seconds to make the remaining ports available with little overhead. *For more information see [6].*

UDP

All global variables have been removed to prevent locking contention and allow for parallel processing of packets.

## 5 IPv6

IPFW Firewall

IPFW now supports IPv6 rules and allows all available actions for IPv6 packets too. The previously separate ipfw6 packet filter is to be retired. The primary advantage of this merge is a single code base and packet flow for IPv4 and IPv6 without duplication or feature differences. *For more information see ipfw(8) and ipfw(4).*

KAME netinet6 Code

Many bugfixes and small improvements have been ported from the KAME codebase.

## 6 IPX

IPX/SPX is still in use at a non-negligible number of sites and some significant effort has been made to lock SPX data structures and to make them SMP-safe.

## 7 Interfaces

CARP Common Address Redundancy Protocol

CARP is a special network interface and protocol that allows two or more routers to share the same IP address. Thus for all hosts using that router any fail-over from one to another one is transparent and no service interruption occurs. Routers in a CARP system may do hot-standby with priorities or load-sharing among them. CARP has been ported from OpenBSD and is similar in functionality to VRRP from Cisco. *For more information see carp(4).*

Ethernet Bridge if_bridge

If_bridge is a fully fledged ethernet bridge supporting spanning tree and layer 2 or layer 3 packet filters on bridged packets. If_bridge has been ported from NetBSD and replaces the previous bridge implementation of FreeBSD. Spanning tree is very important in bridged networks because it prevents loops in the topology. Ethernet packets do not have a TTL that is decremented on each hop and all packets in a looped bridge topology would cycle for an infinite amount of time in the network bringing it to a total standstill. *For more information see if_bridge(4).*

IEEE 802.11 Wireless LAN

The Wireless LAN subsystem has been enhanced to support WPA authentication and encryption in addition to WEP. It may be operated in client (Station) mode or AP (Access Point) mode. In both modes it supports the full WPA authentication and encryption set. The availability of the AP mode depends on the wireless LAN chip vendor, obtainable documentation (w/o NDA) and driver implementation. All cited features are implemented in the ath driver for Atheros-based wireless cards which have the best documentation available. *For more information see ieee80211(4), wlan(4), wlan_ccmp(4), wlan_tkip(4), wlan_wep(4), wlan_xauth(4), wpa_supplicant(8), wpa_supplicant(1), hostapd(8), hostapd(1) and ath(4).*

Interface Polling

The network interface polling implementation has been re-implemented to work correctly in SMP environments. Polling is no longer a global configuration variable but enabled or disabled individually per interface if the driver supports it. Most commonly found network drivers support polling. *For more information see polling(4)*.

NDIS Compatibility – Project Evil

Binary compatibility with Windows NDIS miniport drivers. The NDIS compatibility layer emulates the Windows XP/Vista kernel network driver interface and allows Windows network card drivers to be run on FreeBSD. It supports wired and wireless LAN cards. Many parts have been rewritten and updated as more Windows drivers could be tested, better documentation became available and a more throughout understanding of the NDIS nits developed. It has been updated to work in SMP systems. While NDIS emulation works well it is only a last resort when all attempts of obtaining network chip documentation have failed. A FreeBSD native drivers is always preferred to using Windows drivers through the NDIS emulation layer. *For more information see ndis(4), ndis_events(8), ndiscvt(8), ndisgen(8)*.

Network Driver Locking

Network drivers have to set up and maintain a couple of internal structures. Examples of the structures include send and receive DMA rings and MII information from the PHY. Whenever packets are sent or received the CPU must have exclusive access to the these structures to avoid clashes and confusion. Many drivers had to be re-worked to make them SMP-safe as originally multi-access wasn't a concern. Depending on the network card the driver got a single lock covering all aspects of its operation. Sometimes an even more fine grained approach was taken to have a lock for each send and receive direction plus global state manipulations. Separate send and receive locks provide the best efficiency in SMP systems as two CPU may simultaneously receive and transmit packets.

References:

[1] FreeBSD 5 Network Enhancements, André Oppermann, September 2004, http://people.freebsd.org/~andre/FreeBSD-5.3-Networking.pdf

[2] Network Buffer Allocation in the FreeBSD Operating System, Bosko Milekic, May 2004, http://bmilekic.unixdaemons.com/netbuf_bmilekic.pdf

[3] Introduction to Multithreading and Multiprocessing in the FreeBSD SMPng Network Stack, Robert N. M. Watson, November 2005

[4] TCP/IP Illustrated, Vol. 2, Gary R. Wright and W. Richard Stevens, Addison-Wesley, ISBN 0-201-63354-X

[5] The Design and Implementation of the FreeBSD Operating System, Marshall Kirk McKusick and George V. Neville-Neil 2004, Addison-Wesley, ISBN 0-201-70245-2

[6] ICMP attacks against TCP, Fernando Gont, October 2005, IETF Internet Draft draft-gont-tcpm-icmp-attacks-05.txt

[6] Improving TCP/IP Security Through Randomization Without Sacrificing Interoperability, Michael J. Silbersack, November 2005