Migrating from io-net#

The previous generation of the QNX Neutrino networking stack (io-net) was designed based on a modular approach. It served its purpose in the past by allowing users to separate protocols and drivers, but this came at the expense of incurring a significant amount of overhead when converting to a particular protocol's domain from io-net and vice versa. The new networking stack (io-pkt) is designed to follow as closely as possible the NetBSD networking stack code base and architecture. This design provides the following benefits:

1. Better performance
2. Easier porting of the NetBSD stack code to QNX
3. Updated feature set allowing better socket layer application portabilty
4. Increased leverage of the NetBSD code base including drivers
5. Far richer stack feature set drawing on the latest in improvements from the NetBSD code base
6. 802.11 WiFi client and access point capability

Significant changes that have been implemented in io-pkt

The io-pkt implementation made significant changes to the QNX Neutrino stack architecture.

- mount/unmount capabilities
  - Only io-net drivers may be both mounted *and* unmounted. Other drivers may allow you to detach the driver from the stack using “ifconfig destroy <iface>” if the driver supports it.
  - The IP stack is an integral part of io-pkt. io-pkt can not be started without it. This means that it isn't necessary to specify the "-ptcpip" option to the stack unless there are additional parameters (e.g. prefix=...) that need to be passed in. If the "-ptcpip" option is passed, it will be accepted with no effect.
  - Protocols and enhanced stack functionality can be mounted (e.g. TDP, NAT / IP Filtering), but not unmounted.

- Filter modules
  - The concepts of filters, producers and consumers no longer exist within io-pkt
  - Other hooks into the stack are provided which can be used to provide a similar level of functionality are available. These include:
    - Berkeley Packet Filter interface (allows read and write access to both IP and Ethernet packets from a standard socket interface, but no packet modification or discard)
    - pfil hooks, enabled when the PF filter module is loaded (allows fast read / write / modify of IP and Ethernet packets).

- IP Filtering and NAT
  - The *pf* interface is now used for firewall and NAT configuration. This replaces the io-net *ipf* interface which is no longer supported.

- Driver changes
  - The driver model has changed to provide better integration with the protocol stack (e.g. in io-net, npkts had to be converted into mbufs for use with the stack. In io-pkt, mbufs are used throughout)
  - The driver API and behaviour have been changed to closely match that of the NetBSD stack, allowing NetBSD drivers to be ported to io-pkt.
  - A shim layer is provided which allows an io-net driver to be used as is.
  - Drivers are no longer sequentially numbered with en? designations. They are named according to driver type (e.g. fxp0 would be the interface for an Intel 10/100 driver).
  - Drivers no longer present entries in the name space to be directly opened and accessed with a devctl (e.g. open(“/dev/io-net/en0”)). Instead, a socket FD is opened and queried for interface information. The ioctl is then sent to the stack using this device information (see the source to nicinfo for an example of how this works).

- Protocols
  - SCTP is not yet ported. This will be a future roadmap item.

- Loopback interface checksumming
  - In io-net loopback checksumming was via ifconfig. In io-pkt this is controlled via sysctl.

```
# sysctl -a | grep do_loopback_cksum
net.inet.ip.do_loopback_cksum = 0
net.inet.tcp.do_loopback_cksum = 0
net.inet.udp.do_loopback_cksum = 0
```

Benefits of new architecture

The new architecture provides several benefits that you should be aware of:

- Performance improvements - since io-pkt removes npkt to mbuf translation and mandatory queuing and also reduces context switching on the packet receive path, the IP receive performance is greatly improved.
- Simplified locking of shared resources –results in simpler SMP support
- Closely follows NetBSD code base and architecture
  - Easier maintenance / upgrade capability of IP stack source
  - Existing applications which use BSD standard APIs will port more easily (e.g. tcpdump)
  - Enhanced features included with NetBSD stack are also included with io-pkt
  - NetBSD drivers will port in a straight forward manner

High-level components#

Three major software components constitute io-pkt implementation. These are core components, network drivers and applications / daemons. Each can be individually categorized as follows:

- Core components
  - io-pkt is the main component of the implementation and includes fast forwarding, link layer and stack functionality. io-pkt itself is a link to one of three stack variants, IPv4, IPv4 with full encryption capability and IPv6. Some of the more advanced features of the new stack include the following:
    - Jumbo packets
    - TCP offload (checksum and segmentation)
    - 802.11 Wi-Fi infrastructure
    - Layer 2 support (bridging and spanning tree protocol)
    - Berkeley Packet Filter interface
    - PF interface for packet filtering
    - Network traffic shaping
    - Hardware crypto acceleration
    - TCP SACK
    - Unix Domain Sockets
    - Fast forwarding
    - Routing and raw socket support
    - PF_KEY support (for encryption enabled stacks)
    - Transparent ioctl support

- Configuration utilities : Used to configure, display and manage the stack operation

- Applications/Servers: These are daemons and utilities that have been updated for the new stack (see the next section for a full listing)

- Drivers (For a complete listing of supported drivers, please see the supported hardware link on the Networking Project home page)
  - There are three variants of drivers that are supported:
    - Native (optimized for use with io-pkt)
    - BSD (Ported from NetBSD source using the driver porting library)
    - io-net drivers (inter-operate with io-pkt using a driver shim layer, devn-shim.so)

What components you get with io-pkt#

Core components

Core components are software entities which are only compatible with io-pkt. These can include applications which interface directly with the stack using APIs which are different from those used by io-net.

- *io-pkt-v4* : Basic stack which includes IPv4 support.
- *io-pkt-v4-hc* : Stack with IPSec, Wi-Fi and hardware assisted encryption (FAST_IPSEC) capability to the IPv4 Stack
- *io-pkt-v6-hc*: Stack with both IPv4 / IPv6 support including Wi-Fi and hardware assisted encryption capability
- *pfctl, lsm-pf-v6.so, lsm-pf-v4.so*: IP Filtering and NAT configuration and support
- *tbrconfig*: Traffic shaping utility for configuring a token bucket regulator on the output of an interface
- *ifconfig, netstat, sockstat, ,sysctl* : Stack configuration and parameter / information display
- *pfctl*: Priority packet queuing on tx (QOS)
- *lsm-autoip.so*: autoip interface configuration protocol
- *brconfig*: Bridging and STP configuration along with other layer 2 capabilities
- *pppd, pppoed, pppoectl* : PPP support for io-pkt. This includes PPP, PPPOE (client), and Multilink PPP
- *devnp-shim.so*: io-net binary compatibility shim layer
- *nicinfo*: Driver information display tool (for native / io-net drivers)
- *libsocket.so*: BSD socket application API into the network stack
- *libpcap.so, tcpdump*: Low level packet capture capability that provides an abstraction layer into the Berkely Packet Filter interface
- *lsm-qnet.so*: Transparent Distributed Processing protocol for io-pkt
- *hostapd,wpa_supplicant, wpa_cli*: Authentication daemons and configuration utilities for wireless access point and clients

Applications, services and libraries#

The applications category contains items that interface to the stack through the socket library and are therefore not directly dependent on the Core components. This means that they use the standard BSD socket interfaces (BSD socket API, Routing Socket, PF_KEY, raw socket).

- *libssl.so, libssl.a* : SSL suite ported from the source at www.openssl.org
- *libnbdrvr.so* : BSD Porting library. An abstraction layer provided to allow the porting of NetBSD drivers.
- *libipsec(S).a, setkey* : NetBSD IPSec tools
- *inetd*: Updated internet daemon
- *route*: Updated route configuration utility
- *ping, ping6*: Update ping utilities
- *ftp,ftpd* : Enhanced FTP

Network Drivers#

There are three forms of drivers available with io-pkt. These are

- io-pkt drivers (fully optimized implementations)
- existing io-net drivers (interoperate with io-pkt through the shim layer driver devnp-shim.so
- NetBSD ported drivers ported using the porting library

Native io-pkt drivers are distinguished from io-net drivers by prefixing the drivers by a naming convention (io-net drivers are named devn-xxxxxx.so and io-pkt native drivers are named devnp-xxxxxx.so). Native drivers are written specifically for the io-pkt stack and as such are fully featured and high performance, multi-threaded capability. NetBSD drivers are not as tightly integrated into the overall stack. In the BSD operating system, these drivers are operating with interrupts disabled and, as such, generally have fewer mutexing issues to deal with on the transmit and receive path. With a “straight port” of a BSD driver, the stack will default to a single threaded model in order to prevent possible transmit and receive synchronization issues with simultaneous execution. If the driver has been carefully analyzed and proper synchronization techniques applied, then a flag can be flipped during the driver attach saying that the multi-threaded operation is allowed. Note that one driver operating in single threaded mode means that all drivers operate in single threaded mode. The shim layer provides binary compatibility with existing io-net drivers. As such, these drivers will also not be as tightly integrated into the stack. Features such as dynamically setting media options or jumbo packets for example aren’t supported for these drivers and, given that the driver operates within the io-net design context, the drivers will not be as performant as a native one. In addition to the packet receive / transmit device drivers, device drivers are also available that integrate hardware crypto acceleration functionality directly into the stack.

For a list of supported hardware please see the supported hardware list on the Networking project wiki page.

io-net and io-pkt interoperability and compatibility#

This section describes the compatibility between io-net and io-pkt. Note that io-pkt and the new utilities and daemons are not backwards compatible with io-net.

Installation compatibility#

Both io-net and io-pkt can co-exist on the same system. The updated socket library provided with io-pkt is compatible with io-net. This lets you run both io-net and io-pkt simultaneously. Note that the reverse is not true. If you use the io-net version of the socket library with io-pkt, unresolved symbols will occur when you attempt to use the io-pkt configuration utilities (e.g. ifconfig). The following binaries are duplicated when io-pkt is installed:

- netmanager
- ifconfig
- route
- sysctl
- arp
- netstat
- ping
- ping6
- sockstat
- ftp
- ftpd
- inetd
- nicinfo
- pppd
- pppoed

Binary Compatibility#

The following replaced binaries are known to have compatibility issues with io-net. Essentially, new utilities are likely to contain enhanced features not supported by the old stack.

- pppoed
- inetd
- ftp
- sysctl
- ifconfig

The following io-net binaries are known to have compatibility issues with io-pkt

- snmpd

Socket compatibility #

- The socket library is fully backward compatible with all BSD socket API applications. This also extends to the routing socket.

Protocol compatibility#

- bind() on an AF_INET socket now requires that the second argument has its (struct sockaddr *)->af_family member be initialized to AF_INET. Previously a value of 0 was accepted and assumed to be this value.

Behavioural differences#

- io-pkt will no longer print device loading failure messages to the console by default (they are automatically sent to slog). The "-v" option to io-pkt can be used to force the output to the console for debugging purposes.

Simultaneous support#

Both io-net and io-pkt may be run simultaneously on the same target if the relevant utilities / daemons for each stack are Here are the specific issues you should be aware of:

- The socket library is backward compatible with all BSD socket API applications. This also extends to the routing socket.
- Applications with a tight coupling to the TCP/IP stack aren’t backward compatible. Stack specific versions need to be maintained and executed.
  - ifconfig, netstat, arp, route, sysctl, inetd, pppd, pppoed
- Socket library and headers are not saved. The new versions are backwards compatible.
- Non-network group controlled components may be compatible with io-pkt
  - Phdialer is compatible
  - The network usage widget in pwm will not work with non io-net style drivers since they are not compatible with the devctl interface used by the widget.
- nicinfo has been updated to work with native io-pkt drivers (e.g. nicinfo wm0). NetBSD drivers will not operate with nicinfo.
- Both io-net and io-pkt can execute simultaneously when you provide different instance numbers and prefixes to the stack as in, for example,:

```
io-pkt -d pcnet pci=0
io-net -i1 -dpcnet pci=1 -ptcpip prefix=/boo
```

Note that, in this case, that io-net versions of the utilities have to be present on the target (assumed, for this example, to have been placed in a separate directory) and run with the io-net stack. e.g.

```
SOCK=/boo /io-net/ifconfig en0 192.168.1.2
SOCK=/boo /io-net/inetd
```

Functionality no longer supported#

- Producers
- Filters
- Converters

Discontinued Features#

The current version of the networking stack has deprecated the support for the following features:

- qnet-compat (Neutrino 6.2.1 QNET compatibility protocol)
- The ttcpip (Tiny TCP/IP) stack