# High-Performance Multi-Core Networking Software Design Options

From Intel® Data Plane Development Kit to Wind River Network Acceleration Platform

## Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executive Summary</td>
<td>2</td>
</tr>
<tr>
<td>Multi-core Networking Software Design Options</td>
<td>2</td>
</tr>
<tr>
<td>Traditional Linux SMP</td>
<td>3</td>
</tr>
<tr>
<td>Intel® Data Plane Development Kit</td>
<td>3</td>
</tr>
<tr>
<td>Intel DPDK User Space Libraries with Linux</td>
<td>4</td>
</tr>
<tr>
<td>Intel DPDK with Linux SMP</td>
<td>7</td>
</tr>
<tr>
<td>Network Acceleration Platform with Intel DPDK</td>
<td>8</td>
</tr>
<tr>
<td>Receive Side Scaling and Flow Pinning</td>
<td>9</td>
</tr>
<tr>
<td>Cache Considerations</td>
<td>9</td>
</tr>
<tr>
<td>Lockless Implementation</td>
<td>10</td>
</tr>
<tr>
<td>Huge Pages</td>
<td>10</td>
</tr>
<tr>
<td>Management Plane</td>
<td>10</td>
</tr>
<tr>
<td>Proxy Interfaces</td>
<td>11</td>
</tr>
<tr>
<td>Control NAE</td>
<td>11</td>
</tr>
<tr>
<td>Network Acceleration Daemon</td>
<td>11</td>
</tr>
<tr>
<td>Wind River Workbench</td>
<td>11</td>
</tr>
<tr>
<td>Benefits</td>
<td>12</td>
</tr>
<tr>
<td>Comparison of Design Options</td>
<td>12</td>
</tr>
<tr>
<td>Conclusion</td>
<td>13</td>
</tr>
</tbody>
</table>
EXECUTIVE SUMMARY

Traditional symmetric multiprocessing (SMP) systems scale poorly as the number of cores goes up. Bottlenecks due to shared memory contention and multi-core scheduling become increasingly prominent. The result is a leveling off in packet processing performance, where the addition of new cores leads only to marginal improvements in performance, or in some cases, even performance degradation.

Equipment providers who are looking to base their designs on traditional SMP for reasons of simplicity and convenience should understand the possible drawbacks before proceeding. Increases in performance when moving from a single-core to a dual or quad-core system do not necessarily translate into similar performance advances when moving from a quad-core system to systems with eight or more cores.

This paper looks at a couple of different multi-core networking software design options that scale better than traditional SMP, allowing equipment providers to future-proof their designs by having a single common architecture that can meet today’s performance requirements and scale for the future.

Starting with an overview of the typical multi-core design options, this paper then provides a brief summary of traditional Linux SMP, then goes on to examine the use of the Intel® Data Plane Development Kit (DPDK) and Wind River’s multi-core asymmetric multiprocessing (AMP) product, Wind River Network Acceleration Platform. Both of these options provide better performance and better scaling than Linux SMP alone.

MULTI-CORE NETWORKING SOFTWARE DESIGN OPTIONS

When designing a multi-core networking platform, system architects have typically had to choose between adopting an off-the-shelf SMP operating system or creating a design from the ground up using AMP. SMP is a shared architecture, where all cores run a single instance of the operating system and share all data. AMP is a partitioned architecture, where different cores run different operating systems or independent instances of the same operating systems.

When properly designed, today’s applications can be ported onto general purpose SMP-aware operating systems with very little effort. The underlying operating system figures out where and when to schedule application threads and how to migrate a thread seamlessly from one core to another. These applications can often achieve good performance with some fine-tuning. The problem, however, is one of scale. The shared memory model breaks down as the number of cores and threads grows, limited by the increasing frequency of locks on shared data structures and other performance bottlenecks.

AMP systems, on the other hand, tend to be purpose-built, where the architecture depends a lot on the needs of the application itself. In networking, AMP often refers to architectures containing a single management plane instance on one or a small subset of cores, together with many independent instances of a purpose-built data plane on the remaining cores. Moving an application from a general purpose operating system to a purpose-built AMP
platform not only requires the separation of the application’s data and management planes but may even require a “roll-your-own” implementation of low-level libraries and kernel-like services for the data plane. The result of this additional effort is higher performance, even more so if the software is optimized for the underlying hardware architecture.

TRADITIONAL LINUX SMP

Those in the computing industry are well aware that to make the most of multi-core capabilities, the software must be multi-threaded; that is, there must be more than one sequence of execution so the cores can run different threads concurrently. In other words, the ability for software to parallelize its work is crucial to multi-core performance. Many present-day applications take advantage of this concept with impressive results.

Likewise, many advances have been made in Linux to take advantage of multi-core processors. Improvements to scheduler efficiency and thread affinity, and the move away from coarse-grained kernel locks, have all contributed to increases in performance. While improvements will undoubtedly continue, contending for shared memory and scheduling across cores can become significant impediments as the number of cores scales up. The single network stack instance, represented by multiple threads but common data, is also a constant bottleneck in a traditional SMP system. Other performance constraints include increased translation lookaside buffer (TLB) misses and limited page sizes.

Furthermore, in many networking applications, the performance bottleneck is getting data into and out of the system. The data path needs to be lean, moving packets from the network interface controller (NIC) to the user space application and back to the NIC as quickly as possible. In newer processor architectures, increased bus efficiencies, larger caches, faster memories, and symmetric multi-threading all contribute to more effective data movement and fewer CPU stalls. Linux, too, has improved its handling of network input/output (I/O), but the singular focus that a lean data path demands runs contrary to the mandate of a general purpose operating system, which must provide fairness to all.

So notwithstanding the recent advances in Linux and its applications, what Linux SMP offers most of all is the convenience of getting an application up and running by enabling the application to leverage the operating system for multi-core support, with a commensurate and modest level of performance increase in smaller SMP systems.

INTEL DATA PLANE DEVELOPMENT KIT

The Intel Data Plane Development Kit (DPDK) is a set of data plane libraries designed for high-speed networking. Compatible across all processors in the Intel architecture (IA) product family, the Intel DPDK offers a single software programming model that can be used from the Atom to the latest Intel Xeon processors, able to accommodate vastly different networking applications and system configurations.

The Intel DPDK provides a set of APIs that can be used from Linux user space, offering low-overhead alternatives to traditional Linux system calls, enabling the creation of purpose-built user space applications that can scale in performance beyond what native Linux can offer.

Optimized for Intel processors, the libraries include the environment abstraction layer (EAL) for hardware-specific initialization and configuration, buffer management for real-time allocation and de-allocation of buffers, ring management for the use of lockless rings to increase elasticity and hide latency between communicating tasks, flow classification for lookup and classification of packets and flows, the poll mode driver (PMD) for a polled, interrupt-free mechanism to receive frames from the NIC, and a host of other utilities such as timers, logs, and debugging capabilities (Figure 1).
**Intel DPDK User Space Libraries with Linux**

Linux with Intel DPDK applications sit in user space and make use of the data plane libraries to receive and transmit packets, bypassing the Linux kernel for regular data plane processing. The Linux kernel treats the Intel DPDK application like any other user space application; it is compiled, linked, and loaded in the usual manner. The application starts up as a single Linux process, using Pthreads for parallelism, and Pthread affinity to attach each thread to a specified core.

To understand the benefit of the Intel DPDK approach, it is helpful to recap how traditional Linux handles network traffic and how user applications normally interact with the Linux kernel. A general purpose operating system such as Linux is designed to handle a variety of applications without bias. It must balance competing interests and include sufficient safeguards to prevent applications from trampling on each other or the kernel. Those safeguards include a clear delineation between applications and the kernel, and a set of rules that govern their interactions. Despite a number of advances improving the kernel and how it handles network traffic, Linux remains general purpose, along with the advantages and disadvantages inherent to this label.

The Intel DPDK, on the other hand, is intended to be used for purpose-built, data I/O–intensive networking applications. The libraries are designed to help the application receive and transmit data as efficiently as possible and to provide building blocks for the development of more complex networking software. This approach allows the software developer to focus on the application itself rather than try to determine how to configure and fine-tune the underlying operating system for performance. At the same time, although the implementation of the libraries has been optimized for the Intel architecture, the Intel DPDK does not introduce or impose any particular hardware paradigm on the application. Rather, the Intel DPDK can be viewed as a toolkit upon which application developers can draw, offering optimized implementations of performance-enhancing techniques familiar to developers of real-time software, such as polling and use of lockless rings and zero-copy buffers, all available from user space, while bypassing the generic issues of the kernel.

In a typical networking scenario, packets are received on a NIC, classified to produce a prescribed action, and then acted upon. Regardless of whether the application is a router, a firewall, or a content-aware application such as an application proxy, the need to process traffic as quickly as it is received is paramount. In an era where multi-Gbps rates are common, it is important to realize that the time budget to process a packet on even a single Gigabit Ethernet link can be as low as 672ns (or 1,613 CPU cycles on a 2.4GHz processor), a challenge for any general purpose operating system. While performance may vary based on a number of factors, recent measurements of a native, modern Linux stack on an Intel platform show a forwarding rate of roughly 1Mpps per core, or 667Mbps with small packets. To achieve higher aggregate link rates or to support more advanced applications at line rate, a different approach is required.

It may be surprising that a significant proportion of the time used to process a packet in the traditional Linux model relates to receiving a packet into the system and transmitting a packet out of the system. In other words, even if the user space application does nothing other than relay packets from ingress port to egress port, a significant amount of processing still takes place. To illustrate the baseline costs associated with running a Linux application, consider what happens when a frame is received and transmitted by a user space application (Figure 2).

When a NIC receives an incoming frame from the link, the NIC uses direct memory access (DMA) to transfer the frame into kernel buffers pre-allocated for this purpose, updates the appropriate receive descriptor ring, and raises an interrupt signifying the arrival of the frame. The operating system services the interrupt, updates the ring, and hands the frame to the network stack. The network stack processes the frame, and if the frame is destined for a local socket, copies the data into the socket where it is received by the user space application owning the socket.

On transmit, the user application writes data to a socket through a system call, causing the Linux kernel to copy the data from user buffers to kernel buffers. The network stack then processes the data, encapsulating it as needed, and invokes the NIC driver. The NIC driver updates the appropriate transmit descriptor ring and signals the NIC of a pending transmit. The NIC transfers the frame from kernel buffers into its own internal first-in-first-out (FIFO) and transmits the frame out the link. The NIC then raises an interrupt to signal the successful transmission of the frame, allowing the kernel to release the buffers associated with that frame.
While greatly simplified, this description highlights a few areas of processing that are not part of application processing and can be considered as overhead (at least from the perspective of the application):

- **Interrupt handling**: This includes the suspension of the executing task at the time the interrupt was received, servicing the interrupt, and scheduling the softIRQ handler that performs the actual work for which the interrupt was raised. As the network traffic load increases, the system spends more and more time servicing interrupts such that performance is severely affected when traffic rates approach line rate for a 10GbE NIC. When moving to a system with multiple 10GbE NICs, the system can be overwhelmed, adversely affecting all services.

- **Context switching**: Context switching is the saving of registers and state information from the currently executing thread, followed by the restoration of registers and state information from a pre-empted thread so it can restart execution from where it was left off. Context switching can occur due to scheduling or when interrupts are raised.

- **System calls**: A system call causes a switch from user mode to kernel mode and back to user mode. This causes a pipeline flush and pollutes the caches.

- **Data copying**: Frames are copied from kernel buffers to user sockets and from user sockets to kernel buffers. The time it takes to perform this operation varies with the amount of data being copied.

- **Scheduling**: A scheduler gives the illusion of concurrency in multitasking kernels by allowing each thread to run for a short period of time. The Linux scheduler is run when the scheduling timer interrupt occurs and at various other times when a check is made to see if the current thread has run its course. When the scheduler determines another thread is to be run, a context switch occurs.

Although some of this can be amortized over multiple frame arrivals, the costs, by and large, can easily run into many hundreds of CPU cycles per frame, the difference between processing at line rate and processing at a fraction of line rate.

The Intel DPDK tackles these issues by providing a framework for applications and a network stack to be developed that work...
directly with the hardware for processing of network traffic, minimizing interrupts, context switching, and data copying. The applications and the network stack sit in user space and call on the Intel DPDK libraries for data plane functions (including relaying the incoming packet to Linux for exception processing) instead of making system calls to the Linux kernel. Intel DPDK APIs are available to allow the application to set aside memory at initialization time, including buffer pools, descriptor rings, and lockless queues.

By allocating buffers upfront, the application avoids the overhead of requesting buffers from the Linux kernel during regular data plane processing. From the perspective of the Linux kernel, the Intel DPDK application owns that memory, and the kernel is not involved in how that memory is subsequently used. During normal operation, the application calls the respective Intel DPDK APIs to allocate and de-allocate buffers from this memory pool as needed. Coupled with an Intel DPDK–enabled NIC driver, frames can be placed directly into application buffers on receive and taken directly from application buffers on transmit, with no kernel intervention, offering a zero-copy data plane environment. In effect, the Intel DPDK allows the Linux kernel to be bypassed for data plane processing (Figure 3).

Conceptually, an Intel DPDK application has its own run-time environment (RTE) that operates on top of Linux on each Intel DPDK core. For each Intel DPDK core, the RTE provides a low-overhead, compact environment in which applications can execute and includes a run-to-completion dispatch loop that cycles through all the work that the application is required to perform on that core.

One of the functions of the loop is to continually check the NIC for received frames, eliminating the overhead of servicing interrupts arising from network activity. The composition of the dispatch loop, including the frequency of servicing each task, is entirely up to the application (Figure 4).

In this simple example, a number of tasks are created to handle network traffic. In this context, a task simply refers to a piece of work that performs a particular function and is not intended to denote an operating system task.

The incoming frames are detected by the poll mode driver and passed to the flow classification task where the frame is classified. Based on the lookup results, the frame can be forwarded, discarded, passed to IPsec, relayed to Linux, or sent for endpoint processing. Input to each task is performed by enqueuing to a

---

**Figure 3: Intel DPDK model**

**Figure 4: Dispatch loop example**
multi-producer, single-consumer lockless queue that provides an asynchronous coupling between tasks.

Now when a NIC receives an incoming frame from the link, the NIC transfers the frame into user space buffers pre-allocated for this purpose and updates the appropriate receive descriptor ring. The application detects the arrival of the new frame by polling and proceeds to process the frame. No copying is performed.

On transmit, the user application constructs the frame in user space buffers, updates the transmit descriptor ring, and notifies the NIC of a pending transmit. The NIC transfers the frame directly from user space into its own internal FIFO and transmits the frame out the link. The application polls the NIC to detect the completion of the transmit and releases the buffers associated with that frame.

The set of Intel DPDK components and their descriptions are shown in Table 1.

Table 1: Intel DPDK Components

<table>
<thead>
<tr>
<th>Intel DPDK Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment Adaptation Layer</td>
<td>The EAL provides a set of APIs primarily involved in the initialization of the system, obtaining configuration information such as the number of cores and threads, discovering peripheral component interconnect (PCI) devices, setting up huge pages, setting aside cache-aligned memory, buffers, and descriptor rings, initializing the PMD, and spawning threads on other cores.</td>
</tr>
<tr>
<td>Buffer Management</td>
<td>Buffer management APIs are used to acquire memory and create pools from which buffers can be dispensed during data plane processing, speeding up run-time allocation and de-allocation of buffers. Buffers are cache-aligned and assigned per core so that locks are not needed most of the time.</td>
</tr>
<tr>
<td>Ring Management</td>
<td>Ring management APIs provide a lockless implementation for single or multi-producer, single-consumer enqueue and dequeue operations, supporting bulk operations to reduce overhead. High and low water-mark thresholds for back pressure are provided.</td>
</tr>
<tr>
<td>Flow Classification</td>
<td>The flow classification library provides IA-optimized lookups for n-tuple exact match as well as longest prefix match (LPM) and can be called synchronously or asynchronously. Table lookups are inherently difficult to perform in software at high speed given the size of the tables and the likelihood of a cold cache. To overcome this, the algorithms take advantage of pre-fetching to warm up the caches. Only a single memory access is required for an IPv4 LPM lookup in most cases.</td>
</tr>
<tr>
<td>Poll Mode Driver</td>
<td>The PMD provides an interrupt-free mechanism to receive and transmit frames at high rates and includes zero-copy data movement to and from application buffers.</td>
</tr>
</tbody>
</table>

Intel DPDK with Linux SMP

In this model, the Intel DPDK application on the master core reserves and initializes cache-aligned memory, detects devices, sets up huge pages, and spawns application threads on the other cores. Once the threads have been created successfully and initialized, the master core sets up the mappings between devices and the cores, including using lockless queues where appropriate. To avoid contention for buffers, each thread has its own buffer pool cache, which gets replenished as necessary.

Linux handles all aspects of scheduling the application threads on each core, but for best performance, each application thread should be locked onto a specific hardware thread. The Intel DPDK performs this through Pthread affinity (Figure 5).

When running the Intel DPDK under this model on a six-core 2.4GHz Intel Xeon Processor E5645, IP L3 forwarding performance using flow classification was measured at 35.2Mpps for 64-byte packets using four of the six cores, single thread per core, with four 10GbE ports. When using LPM (longest prefix match) on a six-core 2.0GHz Intel Xeon Processor L5638, IP L3 forwarding performance was measured at 41.1Mpps for 64-byte packets using four of the six cores, single thread per core, with four 10GbE ports. This is approximately a tenfold increase over native Linux, which obtained 12.2Mpps in a dual processor six-core 2.4GHz model.

This approach is tailored to customers who have developed their own network stack and want to quickly demonstrate primary path...
performance on a multi-core platform. If the stack is not already single threaded, then some effort would be required to adapt the stack to using the Intel DPDK single-threaded environment. A deeper understanding of Intel architecture and optimizations will help customers port their software to the latest Intel processors and hardware targets, allowing them to benefit from the tick-tock processor roadmap.

While the presence of Linux solves many infrastructure issues such as booting, using shared memory, and scheduling in SMP, there are fundamental design issues to consider. The separation of the data plane from Linux leads to better data plane performance, but the management of the data plane needs to be defined and resolved. For example, where are route updates handled and how are forwarding tables changed? What are the interactions between Linux and the data plane, and how is exception path processing handled? Does Linux see the data plane interfaces or does Linux manage its own set of interfaces?

By leaving the management plane open to design, this approach allows customers to implement their own proprietary management scheme.

NETWORK ACCELERATION PLATFORM WITH INTEL DPDK

Wind River Network Acceleration Platform is a ready-made, scalable, multi-core AMP solution that integrates an accelerated data plane with a Wind River Linux management plane. Running directly on bare metal, the accelerated data plane consists of one or more cores, with each data plane core called a network acceleration engine (NAE). The NAE provides a high performance networking data path, supplanting the Linux kernel and stack for data plane processing. When deployed on an Intel platform, the Network Acceleration Platform solution leverages the Intel DPDK for IA-specific optimizations in the data plane while cleverly integrating Linux for management control. Whereas the Intel DPDK provides functions to work with the hardware for low-level processing and packet I/O, Network Acceleration Platform provides an optimized network stack implementation on top of the Intel DPDK. The Network Acceleration Platform network stack is complete and is not limited to primary path use cases or static routes and addressing. In effect, Network Acceleration Platform picks up where the Intel DPDK leaves off. Compared to a Linux SMP solution, Network Acceleration Platform scales better because of reduced shared memory contention and reduced dependency on the Linux kernel. Compared to the Intel DPDK with Linux SMP solution, Network Acceleration Platform provides a complete network stack rather than libraries from which customers can develop their own stack (Figure 6).

The Network Acceleration Platform with Intel DPDK data plane includes the Wind River Executive, a streamlined run-time environment for networking applications. On top of the Wind River Executive is a proven, full-featured, highly modular network stack optimized for a single-threaded environment, accelerating network-layer services and functions such as packet forwarding, IPsec, IP fragmentation and reassembly, and virtual local area networks (VLANs) as well as applications requiring full TCP/UDP/SCTP protocol termination. The NAE stack can be configured and built with only the needed components, resulting in a stack that is as full-featured or as lean as desired. Wind River tools are available for the development, run-time analysis, and debugging of software on both the management and the data planes. Management of the stack is performed from the Linux cores, using a combination of a supplied kernel module and a management application.

While some specialized applications may require stack modifications, most software developers should be able to create applications directly on top of the stack using the familiar sockets API. The NAE supports stream and datagram sockets for protocol termination as well as raw sockets that allow the application access to raw packets arriving on the wire. All sockets are zero-copy, meaning that no copying is incurred when passing data from the network stack to the application and from the application to the stack.
No system is complete without management plane control. Just as Linux is used to manage the NAE network stack, software developers can write applications on Linux that manage their data plane applications. To facilitate communication between the management and data planes, Network Acceleration Platform supports Wind River’s multi-core inter-process communication (MIPC) for message passing between cores. The NAE passes packets that cannot be processed by the acceleration plane to the management plane via MIPC. Similarly, any application developed on top of the NAE stack can use MIPC sockets to communicate with corresponding applications on the management plane (Figure 7).

At the heart of the NAE is the Wind River Executive, a run-time environment that consists of a run-to-completion poll-based dispatcher that uses the Intel DPDK PMD to work with the NIC. The Wind River Executive operates in a tight loop that checks for work to be performed, making use of the Intel DPDK libraries for low-overhead run-time buffer allocation and de-allocation and for the implementation of lockless queues to accommodate elasticity of work, unencumbered by interrupt processing and context switching.

Receive Side Scaling and Flow Pinning

When working with NICs that have multi-queue and receive side scaling (RSS) capability, the NAE can receive and transmit packets from and to the NICs without requiring locks. The NIC receives packets from the link and places them into aggregate flow queues based on a configurable (usually an n-tuple) hash. Each NAE processes packets from its set of aggregate flow queues, enabling multiple cores to handle packets from the same link without having to contend with locks and other shared memory concerns. Similarly, when transmitting, each NAE enqueues processed packets onto its own per-NIC transmit queue, again without having to contend with other cores (Figure 8).

Pinning flows to cores in this manner means that each NAE normally handles a subset of flows, increasing the likelihood that flow context and connection state information will already reside in the local cache when the packet arrives, reducing the frequency of cache thrashing where some contexts are flushed out to make room for others.

Cache Considerations

In the ideal scenario, packets would arrive into a system where all required information needed to process that packet is found in the core’s local cache. Imagine if the lookup table entry, the flow context, and the connection control block are already in cache when the packet arrives. There would be no need to “pend,” that is,
wait for external sequential memory accesses to complete before taking action on the packet. The reality, of course, is quite different. In multi-core systems handling tens of thousands of flows, the likelihood that the needed information is already in a particular core’s cache is slim if left to chance, often depending on the recent packet processing history on that core.

While there is no solution to guarantee that the needed information is placed into cache in advance, the Network Acceleration Platform with Intel DPDK uses a number of techniques to improve the odds:

- The NAE is single-threaded, meaning that the cache is not polluted from context switches between interrupt, kernel, and user space.
- Network Acceleration Platform makes use of flow pinning, where specific flows are pinned to specific cores.
- Where possible, the NAE makes use of pre-fetching to warm up the cache with the needed data prior to access.

**Lockless Implementation**

To minimize the need to lock data structures during regular data path processing, the NAEs make liberal use of local data, such as local flow caches and local forwarding information bases. That is not to say that the NAEs do not work on shared data. Rather, data access hierarchies are put in place such that local data is consulted first, followed by accesses to data with increasing scope. Even buffers are allocated from local stores first, before requesting from globally managed pools. Local pools are then replenished from global pools as needed.

When setting up descriptor rings, the NAEs use Intel DPDK’s IA-optimized lockless rings where possible. For interactions with the NIC, the NAEs take advantage of the NIC’s multi-queuing capability. Furthermore, all data structures are cache aligned, maximizing cache-line efficiency and minimizing cache-line contention. In systems with non-uniform memory access (NUMA), memory allocation is NUMA-aware such that memory is allocated from the local node where possible.

**Huge Pages**

HugeTLB is supported using 2MB and 1GB pages. Huge pages reduce the number of translation lookaside buffer (TLB) misses as well as minimize the processing required to handle TLB misses.

**Management Plane**

The challenges of building a multi-core solution do not end with data plane performance. While performance is often the motivating factor in moving to multi-core, solving data plane performance is only solving part of the problem. A complete solution includes a viable and integrated management plane that can manage the system and its interfaces seamlessly, presenting a look and feel that belie the disparate and asymmetric aspects of the underlying architecture.

The complexities of managing the communication between the management plane and data plane are hardly trivial. Dynamic updates of data plane tables, resetting and re-establishing the state of individual cores, and seamless integration with Linux are some of the challenges.

![Figure 9: Management plane architecture](image-url)
The Network Acceleration Platform management plane (Figure 9) consists of a Wind River Linux or other standard Linux installation, working together with a user space application called the Network Acceleration Daemon (NAD), a special NAE called the control NAE (cNAE), and special proxy interface drivers. The NAD, cNAE, and drivers all work in concert to give Linux networking the illusion of a single-OS system, transparently working to allow packets to be sent and received by Linux, as well as to accommodate commands such as ifconfig for controlling Ethernet interfaces and assigning addresses.

**Proxy Interfaces**

Fundamental to the Network Acceleration Platform management architecture is the concept of proxy interfaces. Since Linux and the NAE cannot both own the same network interface, Linux is assigned a proxy interface for each actual network interface used by the NAE. Acting as a conduit between Linux and the actual interface, the proxy interface resembles a regular device driver to Linux, but rather than send packets directly onto a physical link, the proxy interface driver sends packets over MIPC to the cNAE, which in turn sends the packets out onto the link.

Similarly, when a packet arrives on a physical port that requires handling by the management plane, the NAE passes that packet over MIPC to the proxy interface driver on the Linux core. The proxy interface driver then delivers the packet to the Linux stack via the proxy interface, as if it had been received directly from the actual physical port on which the packet arrived.

Packaged as a kernel module, the proxy interface driver provides standard Ethernet functionality such as supporting ifconfig, changing MTU size, and entering promiscuous mode, allowing Linux control over the NAE interfaces.

**Control NAE**

The control NAE is like a master NAE that, in addition to regular NAE responsibilities, performs much of the NAE initialization, including obtaining configuration information from the management plane. The cNAE brings up hardware ports and reports on port capabilities. The management plane then provides information on what NAEs poll which ports and other various information that allows the cNAE to allocate and initialize shared data structures such as forwarding tables.

At run-time, the cNAE performs two key functions:

- It provides support for the proxy interface model, handling Linux packet transmit requests and miscellaneous commands from the management plane as well as propagating interface state changes back to Linux.
- It manages shared data such as forwarding tables, invalidating entries as needed to inform the NAEs to update their local copies.

Because cNAE events are infrequent, and the cNAE only runs when needed, the impact to the data plane should be minor. If warranted, however, the cNAE can be configured strictly for control processing only.

**Network Acceleration Daemon**

The role of the NAD is to snoop and transparently pass events of interest to the cNAE. The NAD is a user-space process running on the Linux core that registers for and monitors netlink events at the link and network layers, relaying those events to the cNAE over a MIPC socket. This includes events such as routing table updates and neighbor state changes.

This mirroring of the Linux control plane enables the integration of standard Linux control plane mechanisms with the NAEs, removing the need for a parallel control plane structure.

**Wind River Workbench**

Debugging a multi-core system is a major undertaking. Often an afterthought, tools are now recognized as crucial to multi-core software development, where the parallelism offered by the architecture brings the advantages of performance but with the increased likelihood of deadlocks, memory corruption, and synchronization problems. Wind River Workbench is a complete, integrated tool suite for debugging, code analysis, advanced visualization, root-cause analysis, and test. Designed for both single and multi-core systems, Wind River Workbench includes a debugger, memory analyzer, performance profiler, data monitor, on-chip debugging, and a host of other features that provide deep analysis of multi-core software, identifying software synchronization and other issues.
Benefits
Unlike those customers who have chosen to leverage the Intel DPDK libraries directly for the creation of their own proprietary data plane and network stack, Network Acceleration Platform customers want a complete, packaged, scalable solution on which to develop their applications. Their expertise lies with their applications, and they do not have the time or perhaps the resources to develop knowledge of the hardware, the drivers, the network stack, IPC, and myriad infrastructure items. The included and highly optimized Network Acceleration Platform network stack removes the need for these customers to create their own networking solution. The integration of the Linux management plane allows them to manage interfaces using familiar methods. The familiar sockets API allows them to quickly develop applications that take advantage of multi-core acceleration. Porting to future hardware architectures should also be easier as Network Acceleration Platform abstracts away the hardware specifics.

Wind River Network Acceleration Platform consists of Wind River Linux and the proxy interface drivers, the NAD application, the cNAE, and the NAE data plane, all packaged and tested together with Wind River Workbench. This complete one-stop solution accelerates not only system performance, but with an advanced set of multi-core tools, time-to-productivity and time-to-market as well.

COMPARISON OF DESIGN OPTIONS
Each design has inherent advantages and disadvantages. What may be appropriate for one situation may prove to be unsuitable for another. Traditional Linux SMP trades off performance and scalability for convenience, while both Intel DPDK and Wind River Network Acceleration Platform provide better performance and scalability but require some amount of additional development effort. Table 2 highlights some of the main considerations when trying to choose the right option.

Table 2: Multi-core Software Design Options

<table>
<thead>
<tr>
<th>Design Options</th>
<th>Target Customers</th>
<th>Performance</th>
<th>Scalability</th>
<th>Data Plane</th>
<th>Management Plane</th>
<th>Network Stack</th>
<th>Sockets API</th>
<th>IPC</th>
<th>Tools and Debugging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional Linux SMP</td>
<td>Intended for customers who are satisfied with the performance and scalability capabilities of SMP and who simply do not foresee a need for higher performance</td>
<td>Modest</td>
<td>Limited</td>
<td>Included</td>
<td>Included</td>
<td>Multi-core</td>
<td>Yes</td>
<td>N/A</td>
<td>Many</td>
</tr>
<tr>
<td>Intel DPDK with Linux SMP</td>
<td>Intended for customers who have developed a data plane they want to integrate with a Linux infrastructure, or for “roll-your-own” customers who have developed proprietary networking solutions and merely want to leverage the DPDK for IA optimizations</td>
<td>High for low-level packet movement</td>
<td>High</td>
<td>Customer supplied</td>
<td>Customer supplied</td>
<td>No</td>
<td>N/A</td>
<td>Many (same as for Linux)</td>
<td></td>
</tr>
<tr>
<td>Wind River Network Acceleration Platform with Intel DPDK</td>
<td>A complete, ready-made product that includes a high-performance network stack pre-integrated with a Linux management plane, intended for customers who want to focus their efforts on their applications rather than on the underlying infrastructure.</td>
<td>High for protocol processing at layers 2, 3, 4, and above</td>
<td>High</td>
<td>Included</td>
<td>Included</td>
<td>Optimized for multi-core and parallel execution, fastpath protocol processing</td>
<td>Non-blocking</td>
<td>Wind River MIPC</td>
<td>Wind River Workbench</td>
</tr>
</tbody>
</table>
CONCLUSION

Network equipment providers looking to take advantage of multi-core platforms are no longer limited to traditional SMP designs that scale poorly as the number of cores goes up. This paper describes two new packet processing approaches that offer better scaling than traditional Linux SMP, enabling the creation of a common architecture that can scale up and down the product line.

Intel DPDK is a set of Linux user space libraries that provides low-level building blocks for customers to create their own high-performance data plane applications and network stack. This solution requires additional development effort and is intended for customers who want to develop their own proprietary data plane.

Wind River Network Acceleration Platform leverages the performance of the Intel DPDK libraries but adds acceleration to networking protocols beyond what is possible with Intel DPDK alone, achieving performance for IP networking, TCP/UDP/SCTP protocol termination, as well as IPsec and VLANs. By offering a ready-made, complete Linux-managed solution, customers can focus on building the applications themselves rather than on building the infrastructure that supports their applications.

For more information on Wind River Network Acceleration Platform, visit www.windriver.com or call 800-545-9463. For more information on the Intel Data Plane Development Kit, visit www.intel.com/go/dpdk.