WHITE PAPER

Avoid Development Hell To Realize Software ROI

The new generation of servers based on increasingly powerful multicore processors and high speed Ethernet technologies (10G, 40G, and 100G soon) enables the development of cost-effective network and telecom equipment using generic bare metal and virtualized servers. Standard OS networking stacks have not been designed to extract the required level of network performance from this new generation of hardware platforms to compete with legacy architectures.

Developing and maintaining a scalable networking stack optimized for multicore hardware architectures is a very complex task. It's very important to understand this complexity before deciding to develop stacks internally or to use off-the-shelf solutions.

Multicore processor vendors provide a software environment to develop network applications. For instance Intel® provides its Data Plane Development Kit (DPDK) that is a set of libraries and drivers for fast packet processing. Software Development Kits (SDKs) such as the DPDK are mandatory to make the best use of hardware resources to receive and transmit packets, but it is important to note that they are not a networking stack.

In the first section, this white paper details the capabilities of the DPDK and why DPDK is not a networking stack.

Then, it reviews the key architecture design challenges to develop a high performance networking stack that can be easily integrated with standard and virtualized software environments and management frameworks. This stack must be designed to benefit from future evolutions of processor and NIC (Network Interface Card) technologies and integrate new protocols.

Finally, the effort for two development options, in-house versus using 6WINDGate, is compared using security gateway equipment as an example.

DPDK

DPDK is provided under an Open Source BSD licensed project called dpdk.org.

The DPDK framework creates a set of libraries for specific hardware/software environments through the creation of an Environment Abstraction Layer (EAL). The EAL hides the environmental specifics and provides a standard programming interface to libraries, available hardware accelerators and other hardware and operating system (Linux, FreeBSD) elements. Once the EAL is created for a specific environment, developers link to the library to create their applications. For instance, the EAL provides the framework to support Linux, FreeBSD, Intel IA (32- or 64-bit) or IBM Power8. Other architectures will be supported soon. The EAL also provides additional services including boot support, PCIe bus access, trace and debug functions and alarm operations.

The DPDK implements a low overhead run-to-completion model for fast data plane performance and accesses devices via polling to eliminate the performance overhead of interrupt processing.

The DPDK also includes software examples that highlight best practices for software architecture, tips for data structure design and storage, application profiling and performance tuning utilities and tips that address common network performance deficits.

The DPDK includes data plane libraries and optimized NIC drivers for the following:

- A queue manager implements lockless queues,
- A buffer manager pre-allocates fixed size buffers,
- A memory manager allocates pools of objects in memory and uses a ring to store free objects; ensures that objects are spread equally on all DRAM channels,
- Poll Mode Drivers (PMD) are designed to work without asynchronous, interrupt-based signaling mechanisms,
- A packet framework - set of libraries that assist with developing packet processing.

The EAL allows loading some plugins without recompiling any applications that use the DPDK libraries. The following plugins are available:

- Ethernet layer PMD supporting Virtio paravirtualized NIC,
- Ethernet layer PMD supporting VMXNET3 paravirtualized NIC,
- Ethernet layer PMD based on shared memory and packet copy for a virtual NIC.

DPDK is mandatory to receive and transmit packets at a very high speed but DPDK does not include any implementation of network protocols. It only provides basic examples to explain how networking software can use the DPDK low-level services. A networking stack of a standard operating system cannot be reused on top of DPDK because its internal architecture has not been designed to scale on a large number of cores. So, network software developers have to design and implement from the ground up a new networking stack on top of DPDK and integrate this stack with the software environment and the management framework.

High Performance Networking Stack Architecture Design

This section reviews the challenges to be solved to develop this kind of software.

Required Software Skills

Developing high performance data plane software requires very specific networking and software skills. Developers must have a skill set combining embedded software background, expertise for a large number of network protocols and hardware/software integration experience. Although having practice in embedded Linux software is mandatory, developing on top of DPDK is very different from developing in the Linux kernel (refer to “Software Design” section). Software developers need to have an in-depth understanding of how the processor works internally (not only how it can be programmed) combined with PCIe, firmware and Ethernet NIC skills.

Software team managers should not underestimate the learning curve for a development team to reach a good level of productivity. At 6WIND, we train teams of embedded software engineers to develop data plane software on top of DPDK and our experience shows an experienced engineer needs between 3 to 6 months to be fully productive.

Software Design

To achieve the highest performance, developers have to use specialized techniques because high performance data plane software is very specific. Reusing available networking stacks is impossible because their architecture has not been designed to scale on multicore processors:

- Specific programming models like run-to-completion leveraging multi-core architectures must be well understood.
- As Layer 3 (IP) performance and latency are directly related to the number of processor cycles used to implement the software, packet flow processes must be designed and optimized to minimize the number of processor cycles.
- Layer 3 fragmentation is an example to illustrate potential design problems. A first design of an IP stack can be done without fragmentation and it will work in 99% of the use cases. However, adding fragmentation to have a fully compliant IP stack will cause many problems such as hardware checksum offloading. IP fragments must be memorized and reordered instead of copying fragments into a linear area where all bytes fit. Multi-segment packets must be correctly implemented. Fragmentation will also have impacts in the higher layers. TCP is now getting horribly complex as you must reassemble IP fragments and TCP segments.
- Modern software stacks have now to be IPv6 compliant. Unfortunately, IPv6 is a completely different software stack compared to IPv4 (fragmentation, automatic address acquisition, end-to-end and hop by hop options…) and requires specific optimizations.
- Buffer copies and locks are restricted to the very minimum and with the most efficient scheme (simple locks, read-write locks or Read-Copy-Update), or implementation method (transactional memory). For instance, table updates must be implemented to avoid any lock that would freeze the system.
- Layer 4 protocol (TCP/UDP) performance and latency are directly related to an efficient memory utilization and locking scheme because for each packet a socket has to be found and updated.
- High performance implies to manage a large number of network objects (interfaces, routing tables, filtering tables, security tunnels and associations…); efficient algorithms to parse large tables must be implemented.
- Hardware / software integration is critical to obtain high performance: location of data, use of caches, use of threads, communication between processors, and interface with PMD network drivers… must be optimized and require an in-depth knowledge of

hardware capabilities.

- Adding protocols may have side effects and need a complete system test and validation.
- Stressing high performance software requires developing specific tools and environments to test software at different steps of the development.
- The validation of a new networking stack in carrier networks is a very long and costly process including proven interoperability.

As a result, specifying, developing and testing high performance embedded data plane protocols is a very low-productivity software task even for skilled engineers.

As there are so many technical challenges to solve, it's almost impossible to define the final software architecture without having proof of concept steps to select the right design options. Skipping this learning phase under the pressure of the development schedule significantly increases risks and may finally lead to major development slippage if software has to be partly or totally redesigned.

Impact on Software Environment
It's very important to design data plane protocols to avoid any impact on the software environment including control plane and management protocols, the Linux operating system, the hypervisor and DPDK. If the right architecture design rules are not well defined, the following major problems have to be solved:

- As new data plane software is developed, it requires APIs to be configured and monitored by both control plane (routing, security…) and management plane including SNMP, RMON, NetFlow, etc. both running in userland. These APIs exist in a standard Linux environment and reusing them for the new data plane software avoids any modification to the control / management planes, otherwise control / management software has to be rewritten / revalidated and support may be broken.
- Data plane interacts with a large number of software components (Linux, KVM, QEMU, OpenStack…) that come from the open source community. If the data plane software requires patches to work with its environment, these patches must be proposed to the community. This process is unpredictable and requires allocating dedicated engineering resources that are familiar with the open source process. Sometimes patches that can be considered as too specific may not be accepted. In this case, these patches have to be ported, sometimes redesigned, and revalidated with each new version of the software and each version of the open source components.
- Unclear boundaries between the data plane software and open source environment may lead to potential issues with open source licenses like GPL. These issues can only be avoided by using a strict company-level process.
- Patching an environment that is supported by a commercial open source distribution provider generally terminates support contracts.

Even if the data plane software is designed to avoid any impact on its software environment, must be validated with new versions of its environment. Commercially supported open source distributions can be used. In that case, data plane software will have to be validated with typical revision cycles of commercial releases.

Direct use of open source distributions leads to specific problems. Open source projects generally add new features to the latest released version. For instance, dpdk.org releases 3 or 4 versions a year. An internal process to use and support releases has to be defined to either always use the latest version or keep previous versions with some backports if features only available in new versions are required. Fixes done internally have to be provided to the community to be integrated in new versions to avoid additional backports.

This multiplies the maintenance effort to support the products in the field and those under development, which will likely use different open source distribution versions. The manpower tasked to this effort increases over time as more versions are deployed.

Technology Evolutions
Multicore and NIC technologies evolve very fast, sometimes faster than their software development.

To benefit from the latest technology improvements that increase data plane software performance, packet processing software must be developed to be easily reused on different hardware architectures. Clean hardware networking abstraction layers to fully leverage processor architectures and hardware accelerators have to be defined using standard packet handling services. Otherwise, software redesign and porting will be required to use different hardware architectures for a complete range of products and leverage technology improvements.

Network protocols also evolve. Existing protocols integrate new options and new protocols emerge. For instance, a large number of encapsulation protocols have recently been defined or are being standardized (VxLAN, NVGRE, STT, Trill...).

Networking and telco markets have fierce competition. Marketing teams want to differentiate products and drive for fast innovation. So, products must be enhanced quickly.

For all these reasons, data plane software must evolve often with new features and protocols and requires a complete development and validation process. Of course, if the existing software has not been well designed, integrating new data plane features may have a significant impact in terms of development costs and time.

Other Costs
Like any other network software, the correct implementation of data plane protocols has to be checked through intensive integration, interoperability and vulnerability tests requiring high performance and very expensive testing equipment because data plane protocols process packets at a very high speed. Validation in real networks can be a long and expensive process for new network software implementations.

IPsec Security Gateway Example
To illustrate all these different challenges, take the example of a security gateway. IPsec, for data plane, and IKE, for control plane, are the core protocols of a security gateway. However, the equipment has to implement a large number of additional Layer 2 and Layer 3 protocols to be integrated in complete network architecture. All these protocols have to be optimized, otherwise the overall performance of the equipment will be very poor.

IPsec itself is a very complex protocol with a large number of options including support for IPv4, IPv6 and many encapsulation mechanisms. IPsec crypto algorithms consume a lot of processing bandwidth that may require hardware accelerators to offload the main processor. Keeping a common interface to use either software crypto libraries or hardware accelerators is a very important requirement to develop a range of products.

High performance security gateways have to manage a very large number of security policies and associations. Standard implementations of IPsec and IKE cannot scale to address this requirement.

Keeping standard interfaces between IKE and IPsec is a key requirement to avoid redesigning the IKE control plane protocol.

Finally, managing security policies is a very important feature and high-level configuration tools are required to automate policy configuration and interfaces with key or certificate management systems.

A complete IPsec security gateway based on Intel / DPDK architecture should at least implement the following protocols:

DPDK
- DPDK with required PMD NIC drivers
- Crypto software libraries and crypto accelerator support (Intel Cave Creek, Cavium Nitrox...)
- Virtio guest DPDK driver if the security gateway runs in a Virtual Machine

Data Plane Protocols
- VLAN
- Link Aggregation
- Ethernet Bridge
- IPv4 Forwarding
- IPv6 Forwarding
- Virtual Routing and Forwarding
- IPv4 Reassembly
- IPv6 Reassembly
- Tunneling (IP in IP, GRE...)
- IPsec v4 and IPsec v6
- QoS
- IPv4 Filtering
- IPv6 Filtering
- Flow Inspection / Packet Capture (for debugging)

Control Plane Protocols
- Synchronization between Linux and data plane protocols to easily reuse standard control plane protocols
- Routing including virtual routing
- IKE
- LACP

Management
- CLI, XML, SNMP, NetFlow...
- Interface with key or certificate management systems

Comparing In-house vs. 6WINDGate-based Development

6WINDGate is a fast path–based data plane networking stack that has been specifically designed to extract the highest performance for packet processing on multicore platforms. Beyond pure performance, 6WINDGate includes all the required features to provide a long-term, ready-to-use solution to minimize development costs and reduce time to market:

- **Optimized software architecture** that linearly scales over a large number of cores located in a single processor or in different processors to deliver unequalled packet processing performance.
- **Complete modular Layer 2 – Layer 4 networking stack** optimizing all IP protocols; customers can purchase the exact list of modules required for their applications and add new modules to provide more services in further steps.
- **Transparent solution for the software environment.** Running Linux and 6WINDGate is identical to running Linux. 6WINDGate's fast path is completely hidden to applications thanks to its continuous synchronization with Linux. So, Linux applications, including management frameworks, work unmodified with 6WINDGate. Using 6WINDGate doesn't require any patching, in the Linux kernel, the hypervisor, or management framework. Customers can keep their standard commercial support agreements in place.
- **Availability on all market-leading multicore platforms**. More than 90% of the 6WINDGate data plane software is written in standard C code and can be reused as is on the networking hardware abstraction layer developed by 6WIND on top of the processor SDKs. The 6WIND DPDK commercial distribution also supports a large number of NICs and crypto accelerators from several providers. Using 6WINDGate guarantees fast porting on new hardware architectures and minimizes support costs in case a customer uses 6WINDGate on different hardware platforms to develop a complete range of products.
- **6WINDGate is a proven solution.** Since its first shipment in 2007, it has been deployed in production in critical carrier network equipment and has been in operation for years showing its quality, interoperability and scalability.

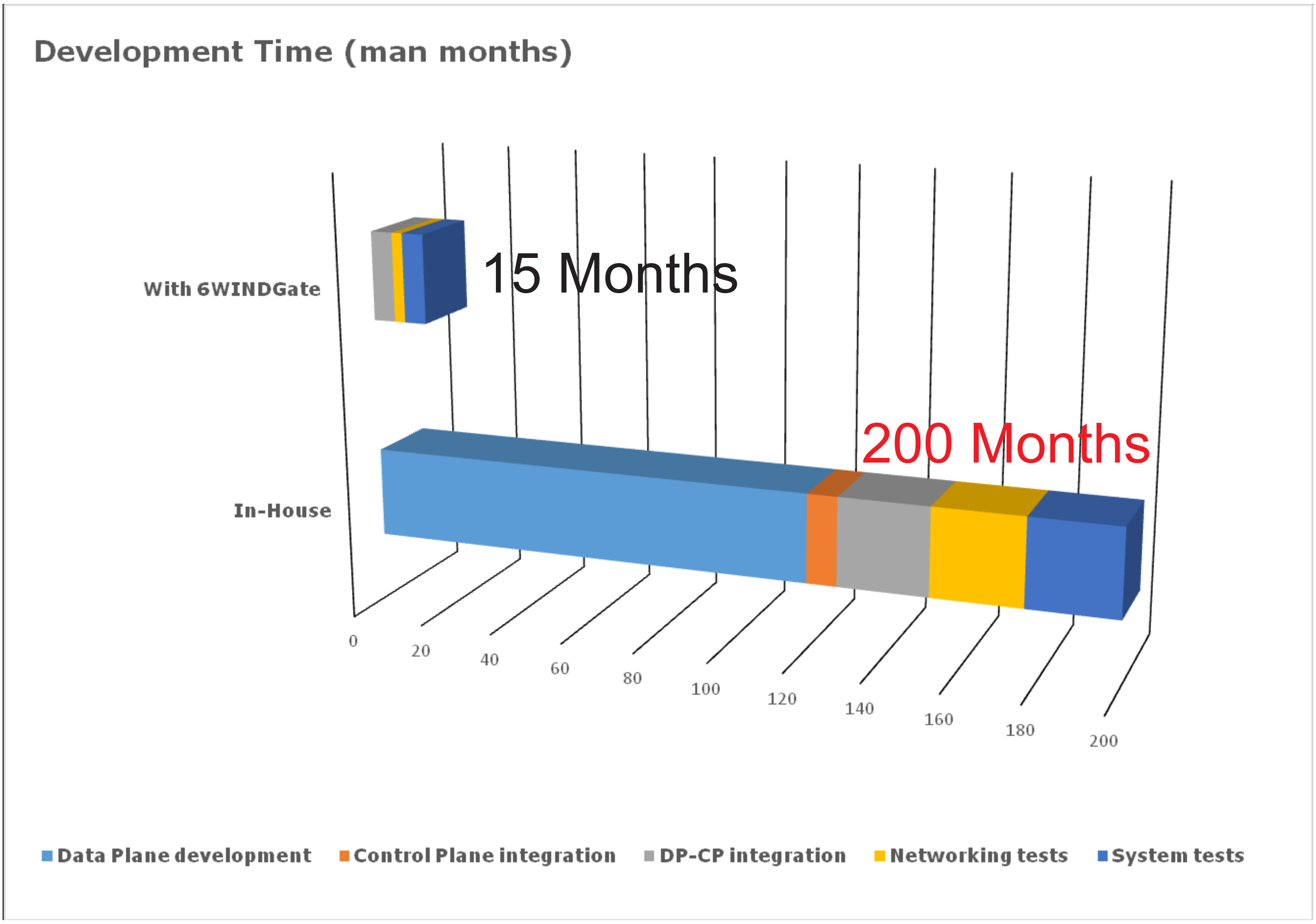
For the complete set of data plane protocols listed in the previous section that are required for a security gateway, the estimated in-house development workload to release a first version of the security gateway equipment based on a generic bare-metal Intel server using DPDK is **200 man months (8 engineers during a period of 25 months)**. This estimation assumes the work is done by a team of skilled networking software developers (refer to "**Development Requirements Task List**" section) and does not include the ramp up period to let the team acquire the required skills. It also does not include any additional workload due to software redesign in the course of the project that may increase development costs by at least 25% and extend the schedule by at least 6 months depending on when the redesign phase occurs.

In comparison the estimated workload to integrate 6WINDGate for a first version of the same equipment is **15 man months (3 engineers during 5 months)**. This team should have more standard software integration skills.

Development Requirements Task List

- Build Team
- Develop optimized data protocols
- Bare metal and virtualization support
- Linux Integration
- Monitoring (SNMP, RMON, NetFlow)
- Create Control Plane API
- Integration with Control Plane And Management Plane
- CLI, XML, key/certificate management
- Performance Tests
- Vulnerability Tests
- Validation in real networks
- Maintenance
- Validation with new versions of the software environment
- Development of additional features to differentiate products
- Portability

The following diagram details the engineering workload for the two options in the different phases of the project. It's assumed the in-house development uses third party or in-house control plane protocols (including management) that are not to be redeveloped but only integrated with the high performance data plane. Networking tests include performance and interoperability tests. Maintenance, validation with evolutions of software environment (DPDK, Linux...) are not included.



6WINDGate also delivers long-term benefits including:

- Availabilty on different hardware platforms to avoid vendor lock-in
- Virtualization ready solution to accelerate the evolution to virtualized appliances
- Simple development of added value features thanks to Linux transparency
- Extensibility with new protocols when required
- Validation with major software distributions (Linux, hypervisor, OpenStack)
- 6WIND roadmap

6WINDGate Avoids Network Software Development Hell

Developing high performance embedded data plane software requires very specific networking and software skills. Specifying, developing and testing this kind of software is a very low-productivity software task.

Considering the extreme complexity of the software to be developed for the current generation of networking and telecom equipment, product line and software team managers should carefully analyze the potential risk as underestimating the complexity may lead to software development hell, unexpected extra development costs and significant delivery delays.

6WINDGate is the result of more than 350 man-years of software R&D and deep expertise in networking. Using 6WINDGate significantly reduces development risk to keep develop costs under control, ship products on time and quickly generate revenues. 6WINDGate's architecture is a valuable long-term investment to easily benefit from the latest improvements of processors, NIC technologies and network protocols.