WHITE PAPER

The Emergence of the OpenDataPlane™ Standard

The networking silicon industry's new open source, cross-platform API for data plane applications on multi-core networking systems on a chip (SoCs).

Abstract

With the great expansion of networks to encompass exabytes of traffic, billions of mobile devices, and billions more endpoints in the Internet of Things, a new era of creative network usage and applications is upon us. The exciting advances in network services — such as 4G and 5G — are leading to the acknowledgement by silicon vendors and application developers of the important role of workload-optimized systems on a chip (SoCs) for hardware offload. Beyond the features in network software alone, SoCs can play an important role in helping operators overcome challenges such as constrained power budgets and user experience demands, contributing to more efficient throughput and lower latency. Yet until recently there hasn't been an open software development kit (SDK) available for silicon designers and independent software vendors (ISVs) that provides APIs for the data plane of diverse SoC architectures. Now there is. Evolving quickly through an industry consortium, OpenDataPlane™ (ODP) is a standardized data plane API that can be used to support Linux-based network applications across the array of silicon architectures and hardware/software configurations. This white paper explores ODP and its features, its relation to the Intel Data Plane Development Kit (DPDK), and several key ODP use cases.

Why an Additional Data Plane Standard is Needed

In areas as diverse as architecture, engineering, and manufacturing, industry standards have proven to be a very good thing. According to the Institute of Electrical and Electronics Engineers (IEEE), standards have enabled faster innovation as diverse products and technologies can interoperate. Ecosystems of innovators large and small can more easily and rapidly create new offerings based on industry standards that benefit billions of consumers and numerous organizations in markets around the world.

In networking there are thousands of standards that encompass everything from small processes to larger categories such as management and security. Networking silicon vendors, however, have until recently had no common data plane standards developed by the consensus of a critical mass of those vendors, and therefore original equipment manufacturers (OEMs) and independent software vendors (ISVs) have had to deal with a wide range of disparate, incompatible interfaces to hardware (e.g., HyperExec, NetOS, Simple Executive, DPDK). This has proven inefficient and it greatly limits developer innovation and customer choice.

With the coming of next-generation application requirements — including the ability to scale to packet rates of 40G to 100G and beyond and the use of network functions virtualization (NFV) to deploy virtual network functions (VNFs) — open silicon data plane standards are more vital than ever.

- **Silicon vendors** want any application to be able to run on their platform without the application developers needing to master the intricacies of the platform. With open standards, silicon vendors can provide optimized implementations of open APIs that can be leveraged across all applications running on the platform. Thus the platform can compete for any socket without having to overcome customer application migration hurdles. Instead, an application can be simply recompiled to execute straight away. With a standard set of data plane APIs, silicon developers can address a much wider market and deliver customers more varied SoCs.
- **ISVs** want software interoperability so they don’t have to write different versions of code for different platforms to do hardware acceleration, service chaining between different chips, and other tasks. Application developers want to be able to move applications easily between platforms while effortlessly taking advantage of the hardware offload capabilities of each. Open data plane standards offer application developers the widest range of target platforms at various price/performance points to optimize the market reach of the application.
- **Operators** need a mix of different kinds of processors in their infrastructure, both workload-optimized that are very good at specific tasks and commodity platforms for other tasks. An open data plane standard makes it easier to provide and maintain such a heterogeneous environment.

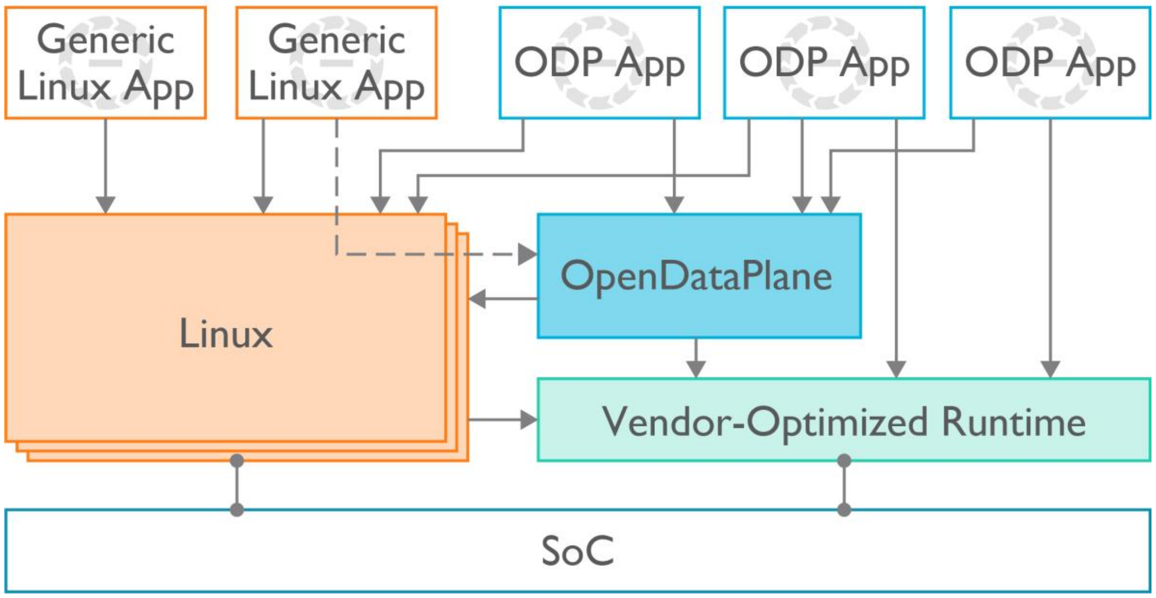
In the computer graphics industry, software developers once had to write custom interfaces and drivers for each hardware platform. There was no common standard for writing software for graphics hardware until the development of the Open Graphics Library (OpenGL) standard. Originally developed by Silicon Graphics in 1991 and released in 1992, OpenGL is a multi-platform API. It interfaces with a graphics processing unit (GPU) to enable hardware-accelerated rendering. The OpenGL API is now managed by a non-profit technology consortium.

The time has come for ODP, a common, open data plane programming interface across diverse silicon architectures (Figure 1).

Figure 1. OpenDataPlane Standards for SoCs

OpenDataPlane™

An emerging standard for common application interfaces to the networking data plane



OpenDataPlane Overview

In 2013 a consortium of networking vendors was organized through the nonprofit Linaro Networking Group open source software engineering organization. Its charter was an open source project that provides an application programming environment for data plane applications that is easy to use, high performance and portable across networking SoCs of various instruction sets and SoC architectures. ODP is the only project focused on networking silicon abstraction with a critical mass of representative stakeholders from networking SoC vendors, OEMs and ISVs, collaborating in the design and evolution of the API, to arrive at the optimal solution built in open collaboration. This is very important to maintain ongoing applicability and market reach.

ODP applications are organized into threads that divide work into separate execution units. Each thread has local storage and threads may share storage with each other via shared memory and may communicate with each other via queue-based message passing.

The ODP environment consists of common APIs, configuration files, services, and utilities on top of an implementation optimized for the underlying hardware. ODP cleanly separates the API from the underlying implementation and is designed to support implementations ranging from NIC and software acceleration to those that deeply exploit underlying hardware co-processing and acceleration features present in most modern networking SoCs.

ODP features and benefits are shown in Table 1.

Table 1. ODP Features and Benefits

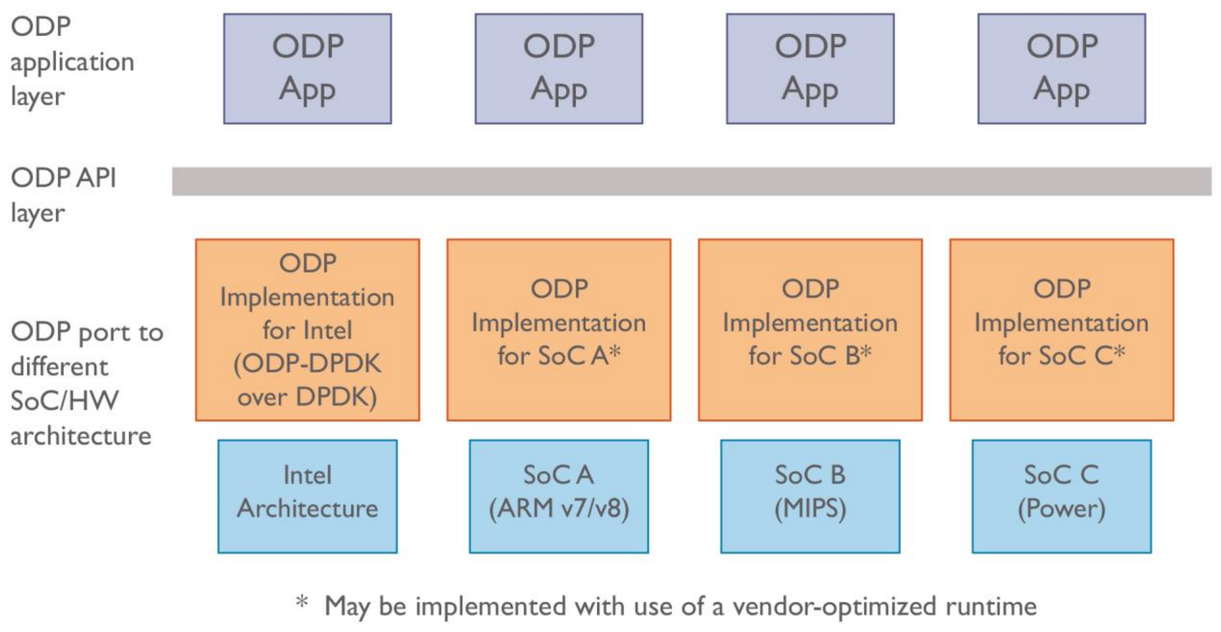
Feature	**Benefits**
Hardware Offload	Greater innovation is possible in the use of hardware offload for various uses, including hardware acceleration, software acceleration, encryption and decryption, buffer management, hardware scheduling, and multicore scalability. You can use any available (as supported by the ODP API), underlying hardware acceleration capabilities, decoupling software innovation from hardware resources while maximizing throughput efficiency.
Acceleration	Special purpose hardware enables very high throughput, performance and power efficiency when properly used. ODP provides an abstraction of common SoC hardware acceleration features, which can be used on multiple SoCs at native performance levels. ODP aims not to abstract all hardware features of all SoCs, but rather a set of the most commonly used and provided features.
Packet Processing	Flexibly schedule packets across many-core architectures. Packets arrive on one or more ingress interfaces and are separated into flows via a classifier function that assigns them to queues. Work is processed from the queues via a scheduler to one or more application threads and/or offload function accelerators and then routed via queues and another scheduler/shaper instance to one or more egress interfaces. (Not every ODP application must follow this model but it is expected to be used for most.)
SoC Flexibility	Enables SoCs to be customized to the requirements of particular applications and deployments (e.g., content delivery caching, transcoding, data analysis).
Application Portability	Allows applications to be easily ported across diverse silicon platform architectures, including those based on ARM®, MIPS, x86, and Power architectures.
Multicore Scalability	Allows for multicore scalability of application processing based on the sophisticated hardware scheduling that is available with many SoCs. An application can be written for a small number of cores, and then run on an SoC with a larger number of cores as needed.

Guiding Principles

The ODP effort has been based on a set of guiding principles. Foremost among them is the decision to tailor the interface based on application use cases rather than implementation details. This was done to avoid prescribing or precluding any underlying hardware or software implementation in the APIs so existing vendor-optimized Runtime environments (RTEs) and SDKs already developed can be used to take full advantage of the underlying hardware (Figure

2). It also enables developers using ODP to avoid having to master all of the proprietary details of each different silicon architecture.

Figure 2. Cross-platform Implementation View of ODP


ODP application layer
ODP App
ODP App
ODP App
ODP App
ODP API layer
ODP port to different SoC/HW architecture
ODP Implementation for Intel (ODP-DPDK over DPDK)
ODP Implementation for SoC A*
ODP Implementation for SoC B*
ODP Implementation for SoC C*
Intel Architecture
SoC A (ARM v7/v8)
SoC B (MIPS)
SoC C (Power)
* May be implemented with use of a vendor-optimized runtime

ODP's guiding design principles include:

- Hardware abstraction without diminishing innovation so existing accelerators and optimized SDKs can be leveraged by ODP implementations
- Event-based programming model abstraction for handling I/O and offload capabilities
- Ability to map APIs efficiently to various hardware offload and acceleration capabilities
- Abstract APIs that reflect application use models, not implementation details
- Support for different I/O and event scheduling approaches
- Demonstrated cross-platform support for multiple architectures, approaches, and device implementations
- A rich set of primitives useful for the data plane

Scalability Support for Multi-Core and Many-Core Processors

Most networking applications are examples of what computer scientists term "embarrassingly parallel" problems — that is, applications that can be divided into an essentially unlimited number of parts. This makes networking an ideal application area for multi-core and many-core processors. One of the main design goals of ODP is to provide a scheduling framework that naturally allows applications to scale out to whatever amount of processing cores are available, without having to redesign the application to run on increasing numbers of cores. This permits applications to target a wide range of price/performance points to best address various market segments and to optimize for various modern design goals such as throughput per watt or per square millimeter of die area.

The way ODP achieves this is by defining abstract APIs that strongly separate the application view from the implementation model used by any given embodiment of ODP. This means that ODP implementations are free to map ODP APIs directly to underlying hardware offload and acceleration capabilities so that applications enjoy vendor-optimized performance on each platform without sacrificing portability.

Support for Accelerators

Silicon OEMs today are providing tremendous value to customers based on the differentiated intellectual property contained within their SOC platforms. This includes features such as traffic managers, workload schedulers, and specialized hardware offload capabilities to accelerate things like baseband processing and cryptography. These features can be tailored to different market niches but until now it has been a challenge to enable applications to take advantage of them due to the extensive redesign and rewrite required. With ODP, API abstractions address this challenge by defining the functional level of an API without specifying the implementation model behind it. This gives vendors the freedom to innovate while still allowing applications to move between ODP implementations as business needs dictate.

ODP Version 1.0 APIs

The ODP APIs are shown in Table 2. The different functional groupings represent the types of processing most commonly needed by data plane applications and those most often found in accelerated form on SoC platforms. For example, cryptography is typically very slow to perform in software but various SoCs have their own unique interface requirements for accessing their hardware cryptography capabilities. So by providing a common abstract API for crypto, ODP allows applications to access vendor-specific hardware without requiring redesign or recoding.

Another key abstraction that ODP provides is a queue-based event model. This enables ODP applications to run on platforms that support both traditional shared-memory and Non-Uniform Memory Access (NUMA)-style processing. The combination of events, queues, and the scheduler are what permit scale-out across a wide range of processing architectures and configurations.

Table 2. ODP API Descriptions

API Group	**Description**
Buffers	Fixed-sized storage blocks for application use
Classification	Parses and assigns incoming packets to Classes-of-Service (CoS) according to Pattern Matching Rules (PMRs)

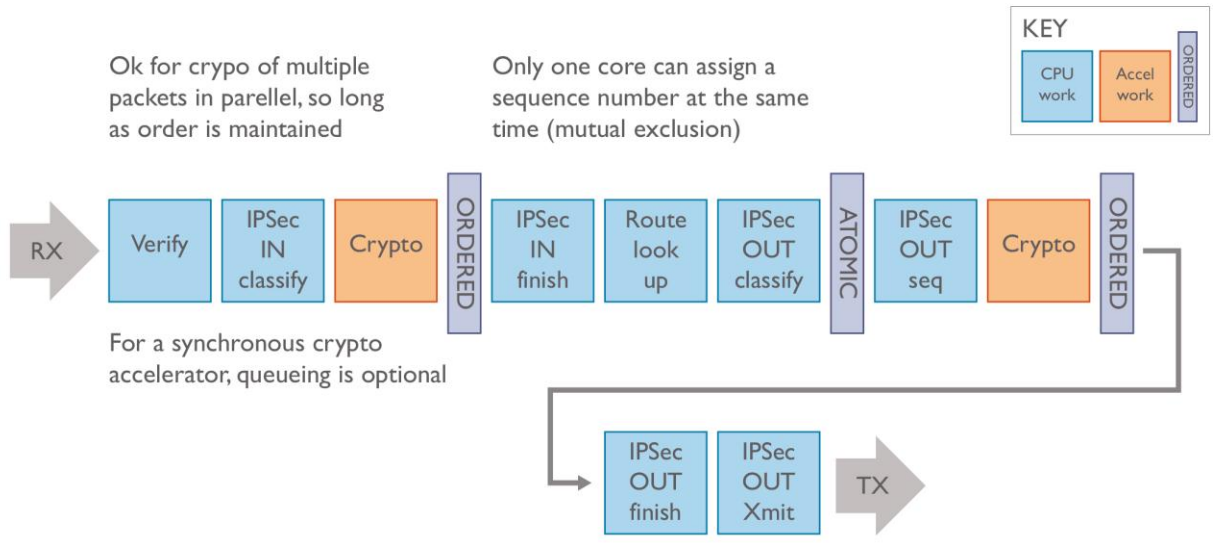
Crypto	Algorithmic offload and acceleration for encryption/decryption of packet and buffer data
Events	Provide notification of packet arrival, buffered communication, timeouts, and completions
Packets	Manipulation, processing of Ethernet frames and their contents
PktIO	Abstracts logical Input/Output ports (network interfaces)
Pools	Storage management for buffers, packets, timeouts, and completion events
Queues	Channels that store and deliver events
Scheduler	Handles sequencing and scale-out of event processing to available application threads
Shared Memory	Abstracts bulk memory allocation for ODP internal and application use
Synchronizers	Atomic and synchronizing operations needed for multiprocessing
Threads	Units of parallel execution
Timers	Notifications of passage of time

A unique feature of data plane applications is their need for very large numbers of concurrent timers — often numbering in the millions. The ODP timer APIs are designed to provide very lightweight access to timing capabilities important to the data plane while permitting individual platforms to map these directly to their underlying timer hardware blocks.

ODP in Action

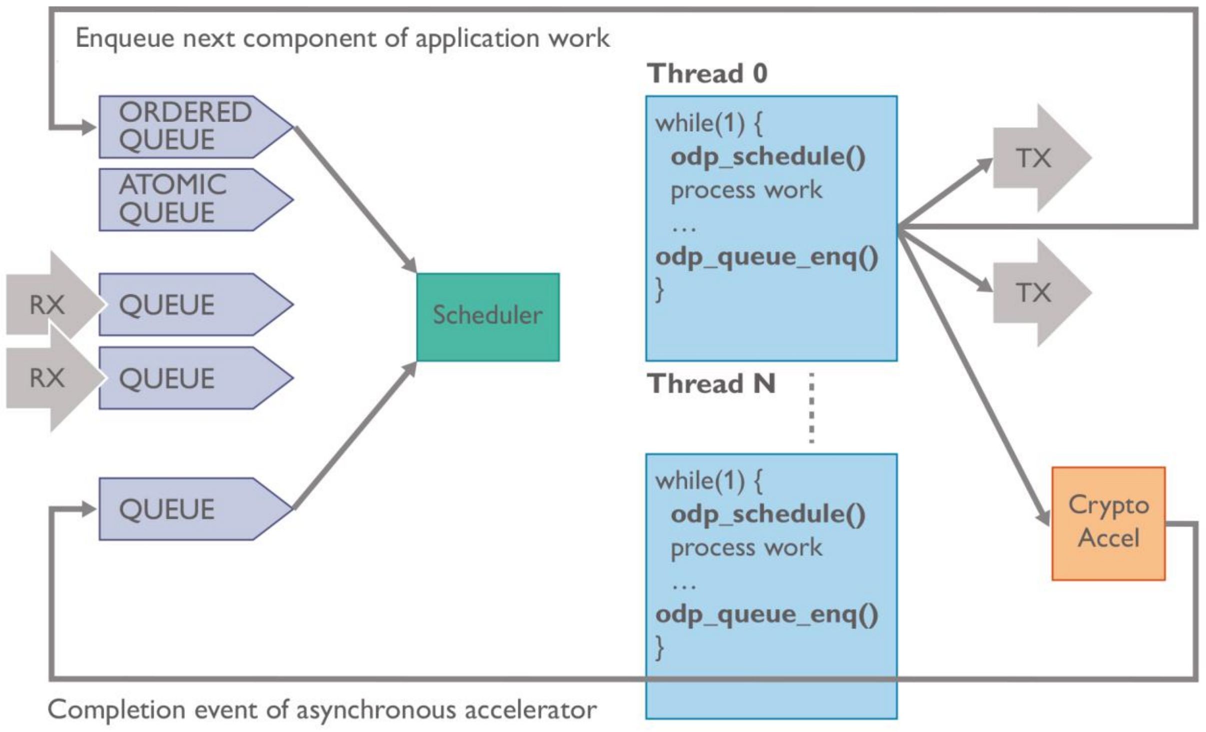
Figure 3 shows an example of an IPsec gateway. The path from Ingress (Rx) to Egress (Tx) involves several logical processing stages. A packet is received and classified and recognized as an IPsec packet needing decryption. The various packets in a flow can be decrypted in parallel, using an ordered queue to preserve their arrival order for subsequent processing. Once decrypted they can be routed and sent for output processing, which again may involve re-encryption using ordered queues before being scheduled for transmission out the target interface.

Figure 3. IPsec Gateway Example


KEY
CPU work
Accel work
ORDERED
Ok for crypo of multiple packets in parellel, so long as order is maintained
Only one core can assign a sequence number at the same time (mutual exclusion)
RX
Verify
IPSec IN classify
Crypto
ORDERED
IPSec IN finish
Route look up
IPSec OUT classify
ATOMIC
IPSec OUT seq
Crypto
ORDERED
For a synchronous crypto accelerator, queueing is optional
IPSec OUT finish
IPSec OUT Xmit
TX

To scale the IPsec function, ODP is designed around a simple run-to-completion/work-scheduling thread model where each thread simply calls the scheduler to obtain work, does whatever application processing is required, and then queues the work for subsequent processing (Figure 4). The scheduler can distribute work across many such threads running in parallel to take advantage of the processing cores that are available to the application. In this manner the application can scale from a handful of cores to many cores without redesign.

Figure 4. IPsec Gateway Implementation with ODP



One of the other important features of ODP is a validation suite based on the open-source CUnit framework. This enables ODP implementations to verify that they conform to the ODP API specification. ODP follows a self-certifying model and simply provides the tools that enable vendors and application writers to verify that various ODP implementations provide the needed level of compatibility of the ODP API specification.

ODP and DPDK

ODP is an open initiative designed to abstract networking functions. It is based on APIs that allow a myriad of vendor-optimized implementations underneath. Applications using ODP can run on a diverse set of silicon architectures and implementations. These include homogeneous server platforms, the multicore SoCs that run carrier networks today, highly complex SoCs such as those used for wireless edge processing, and diverse workload-optimized many-core servers.

DPDK is an Intel® implementation of packet processing best practices in software, originally designed for Intel processors, and was optimized as non-public software for those platforms for several years before the software was published in April 2013 on the www.dpdk.org portal. Packet processing is implemented in software, interfacing to a NIC, with thin support for hardware acceleration underneath. DPDK has been positioned by its creators as a framework for packet processing, preferring to let developers decide how legacy APIs interface to this framework. In addition, there are a number of technical assumptions made that tend to narrow the flexibility of SoC vendors wishing to leverage different hardware capabilities.

ODP takes a different view and does indeed want to drive software interoperability by defining specific APIs for common SoC features and acceleration capability. And therefore ODP has a broader scope that has taken the approach to enable implementers to extend and embrace DPDK, just like any other vendor-optimized runtime environment, when DPDK's software-centric implementation is desired- most typically with Intel x86 based systems for which it was designed.

ODP maintains an optimized, low-overhead integration on top of DPDK (using DPDK for packet I/O and as a software accelerator). Legacy applications and interfaces developed using DPDK can be enhanced with the addition of ODP APIs to migrate *ad hoc*, legacy hardware interfaces to use ODP to broaden abstraction of hardware acceleration and enable instant access to a growing list of optimized ODP implementations. ODP provides portable versions of equivalent DPDK functionality (e.g., get packet, get buffer) to facilitate easy porting.

OpenDataPlane Use Cases

Cavium

At the March 2015 Mobile World Congress, ARM silicon partner Cavium demonstrated its OCTEON® III 48 Core 64bit processor SoC that can run a full IPsec security application at 100Gbps throughput—a requirement for next generation LTE and 5G networks—using ODP APIs. Cavium has been a Linaro member since 2012 when it joined the Linaro Enterprise Group (LEG). Cavium expanded its Linaro membership in 2013 when it joined the Linaro Networking

Group (LNG). Cavium's engineers are active contributors to multiple Linaro projects and Cavium has representation on the Technical Steering Committee for both LEG and LNG.

"OpenDataPlane will allow Cavium customers to run their data plane applications on a wide range of our processors", said Raghib Hussain, Corporate VP/GM and CTO. "The ability to write applications once and then use them on many different processors and architectures is very appealing and since it is backed by a true open standards body, customers will not be locked into legacy architectures."

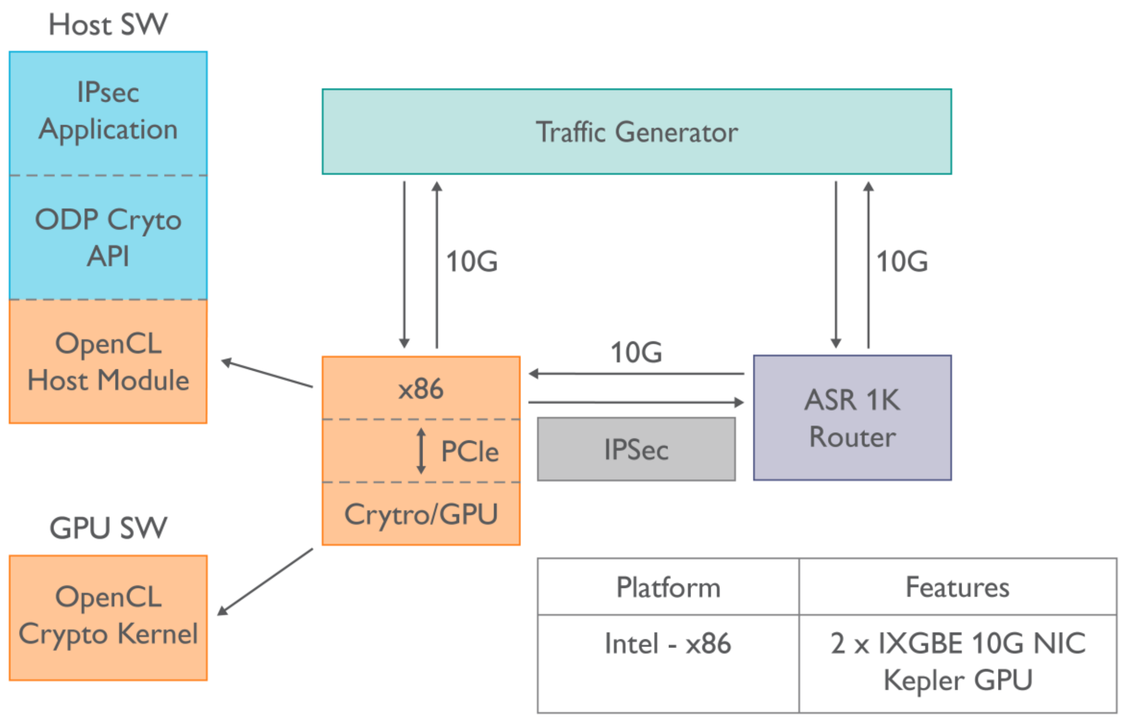
Cavium has also been working on a server-class SoC (ThunderX) based on ODP APIs with ARMv8-A-based cores in a completely virtualized environment to help accelerate the use of NFV for the deployment of VNFs.

Cisco

Using ODP APIs, Cisco recently defined a common implementation of IPsec cryptography that can be used across multiple SoC architectures. The company wrote an application that performs IPv4 forwarding and IPsec encapsulation and decapsulation, with traffic encrypted and decrypted out of two interfaces. Then the application was deployed using general purpose graphic processing unit (GPU) devices based on an x86 platform to provide packet processing offload. The solution is based on a robust implementation of the ODP APIs with an integration layer that leverages many features of DPDK underneath.

As shown in Figure 5, a traffic generator was deployed that generates bi-directional traffic across 10G links to a Cisco ASR 1000 Series router, which encrypts the traffic over an IPsec tunnel between an x86-based, Cisco Unified Computing System server. Inside of the server is a peripheral component interconnect express (PCIe) crypto card and a crypto GPU with an Nvidia k40 GPU node tailored for data processing. The GPU software kernel was developed under the OpenCL open software standard and does encryption and decryption on the GPU. At the upper left, the software running on the CPU on the x86 server is shown. The IPsec application was written by Cisco to demonstrate the versatility of the ODP crypto API. The application uses the ODP crypto API to interface with the OpenCL host module.

Figure 5. Cisco IPsec Crypto Application using ODP API



Platform	Features
Intel - x86	2 x IXGBE 10G NIC Kepler GPU

With this implementation, diverse SoC architectures that handle crypto in different ways can utilize the same ODP IPsec application and run it on different hardware and processors. The red boxes represent components that may be substituted based on different SoC architectures. For Cisco and other application developers, this is an example of one application that can be written once to target the array of different silicon on the market without the need for custom coding for each SoC architecture.

Summary

ODP opens up a world of innovation and flexibility for silicon OEMs and ISVs and new choice and value for customers. For silicon developers, ODP opens up a much wider addressable market as application developers will find it much easier to write code for varied types of SoCs. For software developers, it provides true software interoperability across diverse silicon architectures without the need to write different versions of code. ISVs can use ODP to take advantage of the unique features of each SoC design.

One of the most important takeaways for you today is the fact that ODP is the only project focused on this problem of networking silicon abstraction with a critical mass of representative stakeholders from the networking SoC vendors, including networking OEMs and ISVs. Competitors and partners alike share in the collaborative design and debate. Each step in the design and implementation of ODP is ratified by a majority of consortium members.

For more Information

For more information on OpenDataPlane, visit http://www.opendataplane.org

For more information on the Linaro organization, visit http://www.linaro.org/

For more information on ARM processor design, visit http://www.arm.com/