CORPORATE PARTICIPANTS

Xiaojun (Shawn) Li
Intel – Sales Director, Next Wave OEM & eODM

Tim O'Driscoll
Intel – Software Product Manager

PRESENTATION

Shawn Li

Welcome, everyone, to the Intel Network Builders Insights Series. I'm Shawn Li, Sales Director, Next Wave OEM and eODM, Network & Communications Sales Organization at Intel Corporation, and I am your host for today's webinar.

Thank you for taking the time to join us today for our webinar titled:

"High Speed Packet Processing with the Data Plane Development Kit (DPDK)".

Before we get started, I want to point out some of the features of the BrightTALK tool that may improve your experience. There is a Questions tab below your viewer. I encourage our live audience to please ask questions at any time. Our presenter will hold answering them until the end of the presentation.

Below your viewer screen, you will also find the Attachments tab with additional documents and reference materials, which pertain to this presentation. Finally, at the end of the presentation, please take the time to provide feedback, using the Rating tab. We value your thoughts and we will use the information to improve our future webinars.

Intel Network Builders Insights Series takes place live every month. So, please check the channel to see what's coming and access our growing library for recorded content. In addition to the resources you see here, we also offer comprehensive NFV and 5G training programs through the Intel Network Builders University. You can find the link to this program in the Attachments tab as well as a link to the Intel Network Builders Newsletter.

Today, we are pleased to welcome Tim O'Driscoll. Tim O'Driscoll is a software product manager at Intel. He's responsible for data plane software, focusing on the Data Plane Development Kit (DPDK), Open vSwitch, Vector Packet Processing, and the Linux kernel projects.

Welcome, Tim, and thank you for taking the time to join us today. I will hand it over to you to start off. Thank you.

Tim O'Driscoll

Okay. Thanks, Shawn. So, as Shawn said, I'm going to talk today about the Data Plane Development Kit, commonly known by its acronym DPDK. So it's a fairly broad topic. DPDK has been around 11 or 12 years. It's been enhanced many times over the years. The project continues to grow, so it's a very broad topic. So what I'm going to do is talk a little bit about what it is, why we need it, what particular problem it's solving. I'm going to talk a little bit about some of the more recent enhancements to it. I have a number of links throughout the slides, and particularly in one slide at the end, to references to where you can get further information. So some of those are in the Attachments section at the bottom of the screen like Shawn mentioned. For the others that aren't there, if you download the PDF of the slides later, you'll be able to access all of the links from that.

So just to start then, first of all, I guess, just the standard legal disclaimers. So this is just referencing the regular disclaimers that apply during the presentation.

So DPDK, at a very high level, there's a one-sentence description of it in the very center of the slide. So it's a set of software libraries and drivers for accelerating packet processing workloads on COTS hardware platforms, and that's essentially the essence of it. So it's looking at how you can run networking workloads as software applications on general-purpose hardware, and it's providing the

infrastructure software, an extensive set of libraries and drivers to allow you to do that. Like I said, it's a very long list of libraries, a very long list of drivers. It's enhanced many times over the last 11 or 12 years, but at its essence, that's what it is. That's what it's trying to achieve. So it's all about running networking workloads as software applications on general-purpose hardware.

Some other things to note about it. It's Intel optimized, so we spend a lot of time optimizing for the full range of Intel hardware. So from a CPU perspective, everything from the Atom series through the Xeon D SOC series, up to the Xeon. It's optimized for Intel Ethernet, it's optimized for Intel accelerators, such as Intel QuickAssist and others. So basically, any piece of hardware that we have that's relevant to the networking world, we enabled through DPDK. So that lets you pick and choose the right hardware platforms depending on your target platform, your particular environment, your particular application needs. But the aim is to provide a framework that scales right throughout the Intel product line, like I said, from very low-end Atom-based processors up to the high-end Xeons.

The second thing to note about it is it's an open-source, multi-vendor project, and that's important because people don't want to be locked into a software solution that restricts them to a single vendor. So over the last-- oh, probably going back to 2015, 2016, we've seen increasing participation in the project by multiple other vendors, and some of them called out on the slide there. That's not by any means an exhaustive list. There are many more. You can go and look at the history of patches and see who's been contributing. You can look at the documentation and see who has drivers supported in DPDK, et cetera, but pretty much all of the vendors that have a play in the networking area participate in DPDK in some way, shape, or form.

The second thing to note about it is it's commercially proven. So using some open-source software always carries an element of risk. Different companies have different tolerance for risk in terms of the open-source and external software that they use. DPDK has been around for many, many years. It's used in a huge range of applications, used by a huge range of different companies. Again, there are some called out on the slide there, but that's far from an exhaustive list. That's a small selection. If you want to find out more, a quick search in Google would throw up many more use cases of DPDK. The point really is that there’s a large number of customers who have built and deployed and run commercial applications based on DPDK. So it's commercially proven, proven to be good quality, proven in deployment.

And the last point to note on DPDK is there's a strong open-source ecosystem built around it. So there’s a number of other open-source projects that leverage it, and build on it, and that add more functionality to it. So some of those from an NFV infrastructure point of view would be things like Open vSwitch, commonly used vSwitch, in the NFVI layer. So there's a stock version of OvS, but there's also a DPDK accelerated version. Similarly, Tungsten Fabric. If you need a networking stack, a TCP/IP stack, there are options in terms of the FD.io VPP project. Also F-Stack and mTCP are fairly widely used. If you need a storage framework on top of DPDK then the SPDK project provides that. So there's a strong open-source ecosystem built on top of DPDK as well.

So just to look at why do we need it in a little bit more detail, what's the overriding reason for it? So if you look back in time before the origin of DPDK, you're really looking at a model that's represented on the left-hand side of this diagram. What you have there is a particular application using proprietary hardware and software. Everything’s siloed, so you have a platform for a router, or a VPN, or a firewall, and these are just sample workloads. The slide just shows some examples, but the same principle applies to any networking workload. Really, on the left-hand side, you're going back to a time where you had a custom solution, custom hardware and software for a particular application. Throughout the last number of years, the industry has been transitioning to more of a software-based workload model, running on general-purpose hardware, which is what's shown on the right-hand side of the slide. So you can have your general-purpose off-the-shelf hardware platform, and run a number of virtualized applications on those. Those software-based applications are more portable, can be deployed in a wide variety of different places, more flexible. So really, you're moving from a model, where you had proprietary hardware and software, to a model where you have an open standard platform, running a diverse set of workloads. Now to do that, you need a software infrastructure layer that supports that, and that's where DPDK comes into play.

So just looking at that, in a little bit more detail, we refer within Intel to a strategy of Workload Convergence, and what that means is trying to identify the main different types of workloads, and then looking at how you converge those onto general-purpose hardware. So the workloads are identified on the left-hand side. You can see application, control, packet, and signal processing, and the transition of

those from custom hardware to general-purpose hardware is shown as the slide moves from left to right. So you can see in terms of packet processing, the main driver in terms of transitioning a packet processing workload from custom hardware onto general-purpose hardware was the Data Plane Development Kit (DPDK), and that's the whole reason for its existence, the whole reason for having it, the reason it was created initially and has continued to evolve over the last number of years. It's all about taking networking workloads and migrating them from a model where you have custom proprietary hardware solutions to a model where you have a software-based workload that you can deploy in a flexible way on a number of different platforms, on commercial off-the-shelf hardware.

So just looking at a slightly more technical level, at the specific problem that DPDK is trying to solve, so why can't you just take your application software, run it on top of the Linux kernel, run it on an off-the-shelf piece of hardware, and you're done? Why do you need anything else? So what the graph is showing is it's showing, on the vertical scale, throughput in terms of packets per second, and on the horizontal scale, it's showing different packet sizes. And the different colored lines, the yellow, the green, and the blue, are showing the packets per second versus packet size for different speeds for 10-gig, 40-gig, and 100-gig. And you can see as you move to the left, the number of packets per second, as the packet size decreases, the number of packets per second increases sharply. So for a 10-gig interface with 64-byte packets at line rate, you're talking 14.88 million packets per second. A 100-gig interface, you're talking 10 times that, so 148.8 million packets per second.

So if you look at the two tables just immediately underneath the graph, and look at the right-hand one, which is the heading rows in green, what that's showing is for a larger packet size-- not particularly large, but relatively, kind of a mid-sized packet, 1024 bytes, and it's showing for a 40-gig interface, and the key thing that it's showing is the last line. So assuming a two gigahertz clock cycle, how many-- if you're trying to do line rate for 40-gig packets, 1024-byte packets, on a two gigahertz CPU, you get a budget of 417 cycles per packet. So you need to be able to process each packet in 417 cycles to be able to keep up with a 40-gig line rate.

If you look then at the left-hand table where the header row is in the dark blue, that's showing the equivalent data for 64-byte packets. So typically a comms workload might be dealing with much smaller packet sizes. And what you see, of course, as you move to that smaller 64-byte packet size, is the number of packets goes up dramatically. So at a 40-gig interface, you're up to just under 60 million packets per second, and on a two gigahertz CPU, that gives you a budget of 33 cycles per packet. The 33 cycles per packet is obviously very little. It gives you very little margin for error. You need a very efficient software implementation to be able to keep up with that rate, to be able to process packets that quickly. To do anything in 33 cycles per packet, you need maximum efficiency.

So that's really the problem that DPDK is trying to solve. How can you be that efficient in software? If you just run on top of the Linux kernel, you won't get anywhere near those rates. DPDK is much more streamlined, removes some of the overheads, and allows you to be much more efficient in terms of the packet processing, the overhead of processing each packet, so allows you to develop an application that deals with much higher throughput, much higher packet rates.

Some of the key ways-- a very high level, but some of the key concepts in DPDK that help you to do that are called out in the text. So some of the key things with the running in polled-mode, rather than interrupt mode, so there's a large overhead to processing interrupts for each packet. By switching to polling mode, we're able to process packets much more efficiently and at a much faster rate.

Another item would be avoiding the overhead of the Linux scheduler, and task switching. So we bind a DPDK thread to a logical core. There's no scheduling, there's no context switching, so much more efficient. Also concepts like processing packets in bulk. So process a batch of packets in one go and amortize the cost of some of the operations of all of the packets in that batch, rather than do the same operations for each packet individually. So you can see at a high level, there's a number of key concepts in DPDK that help it to achieve the efficiency, and therefore the throughput rates that are required to reach the kind of packet rates that a modern networking application requires.

So looking then in DPDK in a little bit more detail, so we do have a slide which shows all of the software libraries and all of the drivers in DPDK, but there are so many now that it's a complete eye chart. It's very, very difficult to read. So what this slide is showing is the high-level grouping. So each of these items on the slide represents a whole collection of libraries, and there's much more detail behind them.

So basically, you can view this in two separate layers. So the top layer in blue is the software libraries that are present in DPDK. So there's a set of core libraries that do things like memory management, software rings, and timers. Basic functions like that. There's a set of libraries that perform functions related to packet classification. So they could be hashing/exact match, ACLs, et cetera. There's a set of libraries that accelerate typical functions that a networking application might want to do. So things like IP fragmentation, reordering, power management, et cetera. There are libraries that focus on the statistics collection and reporting; on QoS scheduling, metering, policing. The packet framework is a selection of libraries that allow you to build complex pipelines in a streamlined way. And then there are libraries as well to deal with protocol acceleration for security protocols. The top layer, the blue layer, is all of the software libraries.

The bottom layer then in green is the device APIs. So DPDK deals with a very wide range of device APIs. They can be categorized by the function that they're providing. So we deal with Ethernet devices for I/O, for crypto accelerators, for compression, for event handling. BBDev is baseband device for wireless baseband processing. Rawdev is raw drivers that don't have a specific device class yet. And DMAdev is for DMA acceleration. So all of these deal with the underlying hardware drivers. They're basically API layers on top of a set of drivers performing the functions that are called out here.

And just to look at a couple of those in a little bit more detail. So Ethdev, the original API that was in DPDK from the very beginning, is all about supporting I/O. It supports a very wide range of I/O options. So it supports the full range of internal Ethernet products, the full product portfolio there, and obviously equivalent products from other vendors as well. It also supports a wide range of virtual interfaces. So if you want abstraction, you want to be able to write an application that's not tied to any specific ethernet device, something that could work in an NFV environment where maybe it's running on top of a vSwitch. There are various virtual interfaces supported in DPDK. The most widely used is Virtio. We see increasing interest in AF_XDP, which is a high-performance interface from the kernel drivers up to a user space application, and it's been in the kernel for a couple of years now. There's support for the VMXNET3 interface for VMware. There's support for specific interfaces from different cloud providers. So there's a driver for the Amazon Elastic Network Adapter. There's one for Microsoft's NetVSC. There's a driver that's in progress for the Google Virtual NIC. So a huge range of different I/O interfaces supported. So you can pick and choose the ones that are most suitable for your application. If you want maximum performance, you could use a physical device driver for one of the Ethernet NICs. If you want portability, you could use a virtual interface like Virtio AF_XDP. The Ethdev API also includes functions for advanced capabilities, such as flow matching. That's rte_flow API. And for QoS, that's the metering, policing, and scheduling APIs. So these help in terms of being able to program Ethernet devices to match flows or do quality of service, things like that.

So the Cryptodev API was the first acceleration API besides Ethdev supported in DPDK, added several years ago now, probably seven or eight years ago. It supports both symmetric and asymmetric crypto algorithms, and it supports a combination of hardware and software acceleration for crypto operations. So you can use the same Cryptodev API regardless of whether you're going to use hardware acceleration via something like Intel QuickAssist Technology, or whether you want to perform those crypto operations in software, and the Cryptodev infrastructure will take care of whether to perform that operation in hardware, using, like I said, something like Intel QuickAssist Technology, or whether to do it in software using something like the Intel Multi-Buffer IPsec library. So the concept, and this applies to all of the other acceleration APIs in DPDK, is that the API is generic, and it supports then both hardware and software acceleration implementations underneath that API.

And that concept is extended. I won't go through all of them just for time purposes, but it's extended throughout the remainder of the device APIs in DPDK. The most recent one added was DMAdev. That's for acceleration of DMA operations. So that supports hardware acceleration, such as the Intel Data Streaming Accelerator, and the Intel I/O Acceleration Technology. That's being integrated now into Vhost in Virtio. So we can use that DMA acceleration to offload packet copies in Virtio, and the aim then is to integrate that into Open vSwitch, and to other vSwitches, to help accelerate those through DMA acceleration. And it can be used in other areas as well, acceleration of packet copies in memif, and other potential use cases as well.

So, oops, I've gone one slide too far. Apologies for that. So this is just showing an example. So if you take all those pieces, there's lots and lots of building blocks in DPDK, and you look at a typical workload, a typical application pipeline, it has a large number of different steps, and what this diagram is trying to show is an example of a pipeline for a networking application, and then to identify the blocks

that can be handled by the NIC. So those are in light blue on the left-hand side, and then the darker blue box on the right-hand side is showing what needs to be done on the CPU core. And within that, the yellow is showing the areas that can be accelerated via DPDK, and the green is the application logic. So you can see, because of the breadth of DPDK, there's a large number of areas that it can accelerate, and this is, again, far from an exhaustive view. Like I said, there's a very large number of libraries and drivers in DPDK right now, so you can't really show an exhaustive list of everything that you can do with a framework.

Looking at the networking evolution of the networking industry, you can see that it's going through a continuous evolution, and the question we get sometimes is, well, does DPDK only work in a bare metal appliance model? Does it only work in NFV? And if you look at the evolution of the industry, broadly, it's been evolving from the bare metal appliance model through virtualization and NFV, to what we termed cloud-ready, which is running applications in the cloud, to a true cloud-native model where you're decomposing those applications into individual microservices that can be deployed and scaled independently. And all of those transitions are happening to help improve automation, agility, scalability, and the goal with DPDK is it has been evolving, and it continues to evolve, to meet those needs. So as the industry evolves, DPDK has enhanced and evolved accordingly to meet those needs.

So looking at just some of the key enhancements that have been made to DPDK over the last few years to support that evolution, it's things like-- and I talked about this a little bit earlier when we were talking about the Ethdev API, but it's things like adding virtual and cloud interfaces. So Virtio, like I said, is very widely used. We have support for the latest Virtio 1.1 specification. We have Virtio user support, which allows you to run the Virtio interface in a container. We have vDPA, which allows you to leverage hardware acceleration for Virtio. We have AFX_DP, VMXNET3, Amazon ENA, NetVSC, and Google VNIC in progress, so there's been a lot of work on interfaces, abstracted interfaces, and cloud-based interfaces in DPDK.

The second major bullet there, transparent acceleration, so when you are deploying an application in a wide variety of different physical hardware, you don't necessarily know what capabilities will be present on the target hardware. You need a flexible model that doesn't rely on hardware acceleration having to be there, but that can use hardware if it's available and could fall back to software if it's not. So the acceleration APIs, Cryptodev, and the others that I mentioned before, are all designed on the principle that the same API supports both hardware and software, and that helps to provide more flexible deployment options.

And there's been a lot of work over the last number of years on improving flexibility and ease of use. So there was a big change made a couple of years ago to support dynamic memory management so you could scale up or down memory required by DPDK; Live migration, and a lot of work on Kubernetes integration, so including operator work, VNIs, the whole range there, and a whole lot more. I mean, these are just examples of some of the things that have been happening in DPDK, over the last number of years, to support that industry evolution that I showed in the previous slide. And there are references on the right-hand side to further information on most of those topics.

Release cadence. DPDK, like I said, is a well-established project. It has a very stable release model. Up until last year, we were doing four releases per year. The release numbering is based on the first two digits of the year and the second two digits for the month. So the first example there on the slide, 21.02 was in February of 2021, and throughout 2021, there were four releases. We've made a trial change this year to move to three releases instead of four. So this year, there will be releases-- there was a release, rather, in March. There will be a second one in July, and there'll be a third one in November. We'll try that out this year. If it's successful, then it will be continued into future years. If not, we'll revert back to the four releases. It's really all about trying to strike a balance between how much overhead each release involves in terms of validation, maintaining their workload, et cetera, and trying to reduce that by reducing one release cycle per year.

One of the key things to note here is that the last release every year, that dot 11 release in November, is a Long-Term Support release. So a lot of customers, a lot of users want stability. They want to know that if they adopted a DPDK release, and they find a problem with it, if there's a bug and they need a fix, that they're going to be able to get that easily. So the Long-Term Support releases are maintained for two years. Any bug fixes are backported into new versions of those Long-Term Support releases, but the critical thing is that the ABI of those Long-Term Support releases is fixed. It doesn't change. So changes are only backported into the LTS if they don't change the ABI,

so you know that your application, you have a degree of confidence, when you pick up a new LTS release, that your application is going to work with it, and you get bug fixes backported for a two-year period. So that's proven very successful. We see broad adoption of the LTS releases. They're by far the most widely-adopted releases out of the DPDK release cycle.

Just a couple of examples, although a very, very small sample, but if you go and do a quick search in Google, you'd throw up very many more. But this is really just trying to call out a couple of examples, which show, really, the breadth or the range of different use cases. So originally, DPDK was targeted very much at telco workloads. We do see a wide range of usage in other industries and other areas too now, but the roots of it are still in telco. That's the heart of it. That was the original origin of it. So you can see-- and these are just, like I said, random examples of people who have publicly published white papers describing their use of DPDK. So you can see just the range of applications such as vRAN, UPF, firewall, SBC, so DPDK's widely used across a very wide range of networking applications. So, like I said, these are just a couple of examples just to show that breadth of usage.

And then, further information. So everybody wants to know, always wants to know, where they can get more details. There is very extensive documentation available. So it's not all listed here by any stretch. There's a large amount of documentation available. There is a very comprehensive set of release notes. There is a very detailed programmer's guide. There are guides for each type of driver, so network interface drivers for the Ethernet devices, crypto device drivers, compression, baseband, et cetera. So, whatever capability exists in DPDK, if you go to the documentation page, the release notes is a good reference in terms of what the latest changes are. The programmer's guide is very big, so it's definitely not bedtime reading, but it has great detail on every aspect of DPDK, all the major libraries, all the major APIs. It's a really good starting point.

There are-- and it's been a little bit interrupted over the last couple of years with COVID, but we have a series of DPDK Summit events. Back in 2019, we were doing regular annual events in North America, in Europe, in PRC, and sometimes in India, Japan, and other regions as well. So the talks from previous events are available on the DPDK website. There's some useful information in those, and upcoming events will be published there when they're scheduled. We are looking at planning our first hybrid, partially in-person and partially virtual, event in September this year. Hopefully, that will be what's referred to as the DPDK Userspace event, which is our community gathering event, which will hopefully be in Bordeaux in the September timeframe.

There's a lot of information available on testing and CI. So we have a DPDK Community Lab in the University of New Hampshire. What that does is it takes every patch that's submitted to DPDK, and it runs it through a set of CI and performance tests to make sure it's passing basic CI tests, and to make sure it's not degrading performance. So that gives us a degree of confidence, when patches are applied, that they're not going to have an adverse impact on existing functionality around performance.

We have the DPDK Test Suite. So that's a test framework and a large collection of test cases that get executed on every release, and we have a number of publicly-available performance reports. So they show you what performance level you can expect for some basic use cases out of DPDK.

There is training available. So I think Shawn mentioned in the intro the Intel Network Builders University. There is a section of that around DPDK.

And then there's other information that's available. There's a DPDK white paper, there's a roadmap page on the website, and various other collaterals as well. So it's a very comprehensive set of documentation. It's a well-documented project. There's a lot of information there. There are a lot of videos available on YouTube. So if you're thinking about getting started with it, there's a great starting point in the documentation and in the training information there.

So that's the last slide I was going to present. I guess that means we're onto time for questions now if there are any questions.

Shawn Li

Hey, appreciate it. Thank you. Thank you. We have a couple of questions, Tim, for you.

How much performance can I get with DPDK?

Tim O'Driscoll

Okay. So I mentioned that we have a set of public performance reports. I can share those briefly now. So I'm just going to switch the screen.

Okay, so on the dpdk.org website, there's a section for performance reports. They're typically posted per release. You can see the top three there. There's an Intel NIC Performance Report, Virtio/Vhost one, and a Crypto Performance Report. They contain a lot of detail on performance of different test scenarios.

So just looking at those briefly, so this is the NIC Performance Report, if you take a sample test case, so this one, for example, is using the Intel 800 series Ethernet controller, and it's doing a simple L2 forwarding test, and you can see the performance rate that you can achieve with that. So there are different combinations, but this is a 64-byte packet with turbo boost enabled running at 2.9 gigahertz. It's doing 125 million packets per second. There are variations on this, I think. So if you enable hyper threading, you can see what the impact of hyper threading is. There's a whole range of different performance tests there.

Is the same thing available for crypto? So again, this shows performance with hardware acceleration using Intel QuickAssist Technology, and performance with software crypto using the IPsec Multi-Buffer Library. So for example, the crypto performance, this is the performance you get using QAT, and this is, I think, the equivalent performance using software acceleration. And the same thing, again, with the Virtio one. I won't go through that just for time purposes. But those are great, those public-- they're all publicly available on the dpdk.org website. Those public performance test reports are a great starting point in trying to understand the performance. These are obviously just DPDK tests. I mean, you would need to calculate or factor in the overhead of your particular application, which is obviously user specific, but that's the best starting point in terms of publicly available performance data.

Shawn Li

Great. Thank you, Tim. What are the most common use cases?

Tim O'Driscoll

Yes, that's a good question, and it's a difficult one to answer because it's a very wide range. When DPDK was originally created, probably 12 years ago at this stage, it was targeted very much at telco workloads, and that's still the most commonly used. So we would see strong usage in RAN. We would see strong usage in wireless core. We see usage in wireline and cable. We see usage in routers and switches, in enterprise applications, in security appliance, firewalls, et cetera. There are other use cases. We have seen examples of DPDK being used in high-volume financial trading, and things like that. So the use of spreads beyond the original target, but the original target was very much networking workloads, workloads that are trying to deal with small packet performance and need to be able to do that as efficiently as possible. So the origin of it would be wireless access, wireless core, extending then into routers and switches and wireline, cable, et cetera.

Shawn Li

Good, thank you. Next question. Can I contribute to the DPDK?

Tim O'Driscoll

Yes, actually, that's a good question. Yes. So it's a completely open project. It's a project that's maintained by the Linux Foundation, so it's completely open source, freely available for anybody to use, or anybody to contribute. I'll share, again, for one second. I've just got to find the right place. So I talked about the documentation before, so this is the full list of the documentation that's available, and you can see everything through getting started guides, the programmer's guide that I mentioned, the release notes. But there's one here called Contributor's Guidelines. So as you can see, that covers things like coding style, ABI policy, when you can and can't change the ABI. It includes guidelines for unit testing, so what unit tests you need to provide with your patches. And there's a section here then

contributing to DPDK. So this really talks about the whole development process. So how to prepare the patches, what checks you should do on patches before submitting them, what the review process is when you submit them, how you monitor progress, et cetera, when you might have to do rework, things like that. So it's all quite nicely called out in that Contributor's Guideline document, which is available on the documentation page, which is in turn available from the dpdk.org homepage.

Shawn Li

Good, thank you. Thank you. The next question. Well, related to this one. What percentage of DPDK new content comes from Intel versus external contributors? Do we know what external companies provide the most new content?

Tim O'Driscoll

Yes, so at every DPDK release, the main maintainer, Thomas Monjalon, who is an employee of Nvidia, when the release is complete, he sends out a release email to the DPDK announce list. He does a good job in that of calling out all of the contributors to the release, and he breaks down, actually, the number of patches contributed by each company. So typically, we see Intel being the largest contributor. The percentage of contributions that we make varies per release, but typically, I would say it's around 30% or so of the contributions in a release come from Intel. And I'm quoting that figure from memory so take that with a grain of salt, but typically, Intel is the largest contributor, closely followed by-- we would see then significant contributions from Nvidia. We would see significant contributions from Marvell, from NXP, and a number of other vendors. So the exact ratio varies based on the release. In some cases, one company contributes more, and another one less, but typically, Intel is still the biggest contributor. Like I said, I would guess 30 to 40% of the contributions maybe on a typical release come from us.

Shawn Li

Great, thank you, Tim. Next question. Are there any special hardware requirements?

Shawn Li

Good question. Okay, I'm going to share again, because there is a list of tested hardware platforms in the release notes, I believe. So one second just to find it.

So again, I'm going back to the main documentation page, into the release notes, and you can see the release notes go back a long way, and they'll tell you what's changed in each release. So looking at the release notes for the latest release, it obviously lists out the new features, it lists out API and ABI changes, but the piece I wanted was the tested platforms.

So this isn't an exhaustive list of everywhere that it will run, but these are just the platforms that it's been tested with, so you can see a long list of different CPU types, a long list of different operating systems, a long list of different Ethernet devices as well. So there's a lot of detail in the release notes on the tested platforms. These are the Intel-tested platforms at the beginning, but you can see there's equivalent information from Mellanox, Nvidia, and others further down as well.

So like I said, this isn't an exhaustive list of everywhere that it will run, but this is a list of everything that the last release was tested on. So it gives you a good frame of reference, and you can see the breadth of platforms that it's tested on. It's a fairly wide range of platforms. It's a wide range of OSs. It's a wide range of NICs. One of the challenges that we always have with each DPDK release is just the number of combinations. So we have different NICs, different operating systems, different compilers, different CPUs, and trying to minimize the number of combinations and get them down to a manageable level can be tricky sometimes.

Shawn Li

Okay, next question. Thank you, Tim. How much performance degradation HQoS cause?

Tim O'Driscoll

Yes, so that's a good question, and I can't give a specific answer, but HQoS can be intensive in terms of CPU requirements. There are implementations in software within DPDK. There's also the ability-- if the ethernet device, the NIC supports it to be able to offload some of that to the NIC. So that obviously offloads some processing from the host CPU. But, yes, HQoS can be a significant part of an application pipeline. So yes, I don't have specific figures. It would need to be tested on a case-by-case basis, but it does depend on whether you're doing HQoS in software using the DPDK software libraries, or whether you're offloading some of that to an Ethernet device that's capable of performing some of those capabilities. An example might be the Intel's 800 series Ethernet controller, which does support some of the HQoS functionality.

Shawn Li

Okay, thank you. Oh, one long question, probably short answer is, your PDF attachment has pop-up bubbles obscuring important parts of the slides or diagrams? Can we get the clean or non-obscured version?

Tim O'Driscoll

We can check that. We can double-check the PDF. Sometimes when you generate a PDF, it doesn't necessarily generate the way that you want. So hopefully, it's okay, but we'll double-check that and make sure, and if there is an issue with it, then we can regenerate a new version. Yes, good point.

Shawn Li

Good, thank you. The last question. Does DPDK include TCP/IP stacks?

Tim O'Driscoll

Oh, good question. If I go back a couple of slides, give me one sec to find the right one. So the short answer to that is no. DPDK doesn't include a TCP/IP stack. But I did talk early on about if you look in the bottom right-hand corner of this slide, the Open Source Ecosystem, so there are a number of other open-source projects that build on top of DPDK, and a number of those do provide TCP/IP stacks. So there are more. Again, this slide is just showing a small sample. If you go and search, there's a whole range of TCP/IP stacks built on DPDK. But the primary one in terms of adoption that we know of would be the vector packet processing project in the FD.io. So that includes a very comprehensive networking stack. VPP can be used with or without DPDK, so it has a small set of native drivers directly in VPP. If you're using certain device types, if you want a broader range of I/O interfaces into VPP, then it could be used with DPDK. So you get the benefits of the breadth of I/O interfaces that DPDK supports, coupled with the networking stack from VPP. There's also F-Stack and mTCP. So there are a number of options, but DPDK itself doesn't have a TCP/IP stack.

Shawn Li

Good, thank you. That's all the questions. I really appreciate your time. Thank you for joining us today, and please give our team a rating for the live recording, so we can continually improve the quality of the webinar. You are welcome to join us the next time Wednesday, July 13, at 8:00 a.m. Pacific for a session of the "Reference Implementations – Free, Customer-Deployable Apps for Customer Use Cases". You can find the link for registration in the Attachments tab.

Thank you again for joining us today. This concludes our webcast. Thank you.