The Lagopus SDN Software Switch

Here we explain the capabilities of the new Lagopus software switch in detail, starting with the basics of SDN and OpenFlow.

3.1 SDN and OpenFlow

Those engaged in network-related work are likely to have heard the word SDN come up. Conventional network construction involved manually setting each dedicated hardware device, such as routers and switches, that is installed as a network component. In contrast, the basic concept of Software-Defined Networking (SDN) is to configure settings for network components via software, using information centrally managed through a system called an orchestrator.

According to the Open Networking Foundation (ONF), which is widely promoting SDN and OpenFlow, SDN components can be categorized into three layers (Figure 1).

- **The Application Layer**
- **The Control Layer**
- **The Infrastructure Layer**

The application layer governs the centralized management and user interfaces that serve as the core of SDN. The components of this layer can handle a wide range of management tasks. They cover cloud solutions, or more specifically virtual machines, in addition to SDN. In terms of software this corresponds to orchestrators such as OpenStack and CloudStack.

The control layer serves as a bridge between the application layer and infrastructure layer. From the perspective of the application layer this layer provides network services. Its main role is replacing process requests from the application layer with specific configuration instructions to configure the settings for each component in the infrastructure layer. In terms of software, frameworks such as Floodlight and Ryu are used.

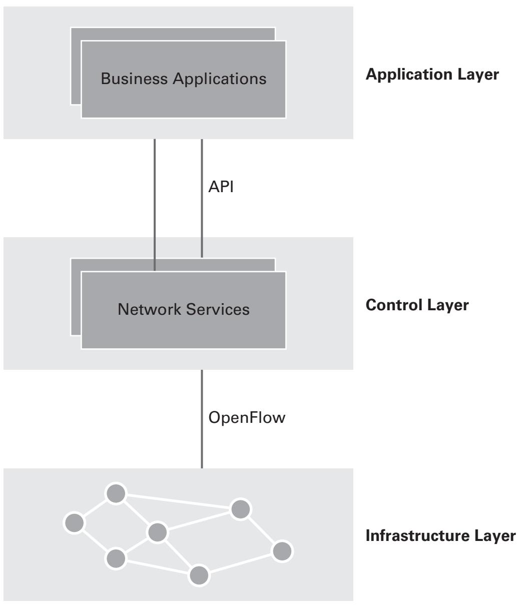

Figure 1: SDN Components

The infrastructure layer, as the name suggests, governs network infrastructure. This corresponds to the components that send and receive customer network packets. In a broad sense, the components in this layer are like routers or switches that enable software control from the control layer. However, in a more limited sense, they are components that can be controlled via vendor-neutral APIs and protocols.

The OpenFlow Switch Specification is a well-known switch specification that enables control via vendor-neutral protocols. A control protocol called the OpenFlow protocol is also defined under this specification for use between controllers and switches. OpenFlow is defined as an open standard, so it is possible to implement hardware or software freely referring to this specification. In addition to hardware switch products, the infrastructure layer components that support OpenFlow include a number of software switches implemented as software.

3.2 OpenFlow

OpenFlow was designed at Stanford University in 2008, and the specification was subsequently expanded and modified, leading to the current open technical specification defined by the ONF. The specification has version numbers assigned, with OpenFlow 1.0 currently the most commonly used. It has been expanded a number of times since then, but although versions 1.1 and 1.2 were developed, upgrades were released before these became popular, so they are not often used. Implementations of OpenFlow 1.3 are often seen, as it is slated for long-term support. At the time of writing the latest version is OpenFlow 1.4, but because the increased complexity of this specification has made it harder to implement, there are currently hardly any implementations that support it. The OpenFlow specification does not take compatibility between versions into consideration, so the versions used between controllers and switches must match.

The concept of OpenFlow begins with the separation of the control plane and the data plane. The data plane processes and transfers packets at high speed according to the rules applied, while the control plane applies these rules. In this case, rules are what distinguish the nature of incoming packets. For OpenFlow, each rule and the corresponding process are paired together in what is known as a flow entry. For example, it is possible to apply a rule for packets sent to TCP port 80 at the destination IP address 192.168.0.1 received over LAN port 1. A collection of flow entries is called a flow table. When a flow entry corresponding to packets processed via OpenFlow is found on a flow table, the associated action is executed. These actions include simply sending the packets to a specified LAN port, as well as modifying the packet headers by altering the destination IP address and inserting a VLAN tag. Actions can be specified freely within the scope of the specification. It is also possible to prepare multiple flow tables, and specify an action that causes the next flow table to be processed.

In addition, you can specify actions that send packets to the control plane, or in other words the controller. The controller receives packets from the switch, and enables more complicated processing to be executed. Additionally, the controller can instruct the switch to send arbitrary packets. Figure 2 gives an overview of these points.

This demonstrates how OpenFlow was designed to enable complicated packet processing while maintaining high speed through coordination between the controller and switch. However, the extension of the OpenFlow specification itself is causing the data plane to become increasingly complicated.

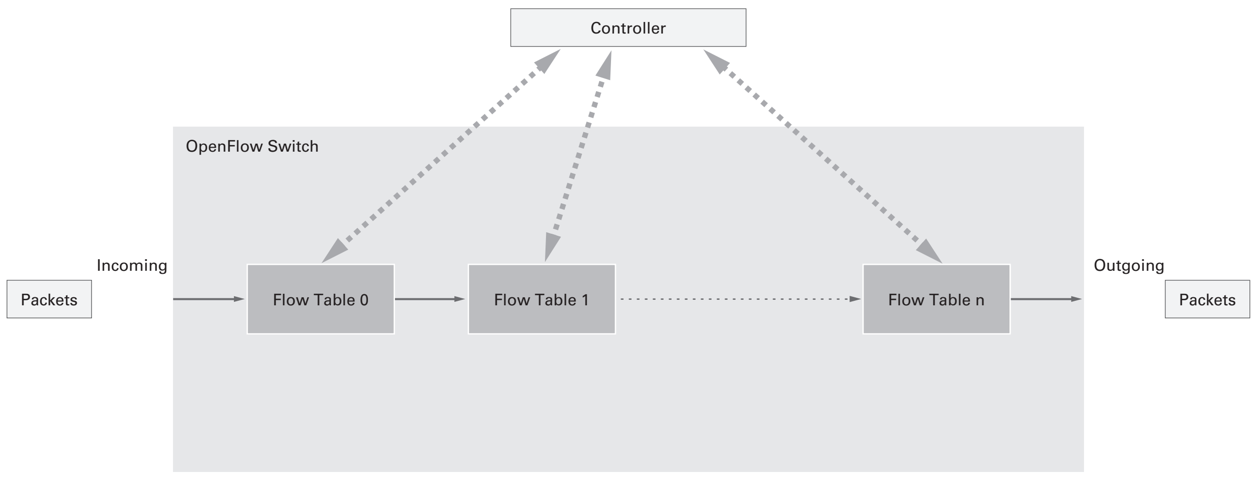

Figure 2: The Concept of OpenFlow

3.3 Software Switches

Broadly speaking, methods for implementing Ethernet switches that support OpenFlow can be grouped into three categories.

The first category is hardware switch products that use ASIC. While these products have large switching capacity, and can provide the performance to support large-scale networks, the number of flow entries and tables that can be handled is considerably limited, and the devices are also expensive. Additionally, although the OpenFlow specification includes items that are required as well as those that can be omitted, it is necessary to note that on hardware switch products the difficult processing of implementations is often skipped.

The second category is products that use network processor units (NPU). These use processors that incorporate special functions for network processing, and are equipped with firmware that takes advantage of these functions. Compared with switches that use ASIC, there are fewer restrictions on the number of flow entries and tables, as well as the items implemented. Also, although their speed is not always on the same level as hardware switches, they can be applied to large-scale networks. They are also expensive, however, and it is difficult to justify the cost for use in medium or small-scale networks.

The third category is software switches. In this case the features of OpenFlow are implemented using programs run on a universal server OS (for example Linux). These can also be run on a virtual machine, meaning they have good compatibility with cloud solutions. While they can provide all the functionality of OpenFlow depending on the implementation, their performance hasn't even been in the same league as ASIC and NPU in the past, and this was a major issue.

The speed when processing 64 byte packets at wire-rate on a 10GbE network interface is 14.88 M frames/sec (fps), which comes to 67 nanoseconds of time allowable per frame. If the processor clock is 2 GHz, each clock cycle is 0.5 nanoseconds. Roughly speaking, if we assume one command per clock cycle, the processing of a packet must be completed within 134 commands or packet transfer at wire-rate will not be possible. The software switch implementations that existed in 2012 were only able to process a few hundred thousand to a million frames per second at most, so the development of a new software switch implementation to resolve performance bottlenecks was investigated. This led to the development of the Lagopus SDN software switch.

3.4 The Lagopus SDN Software Switch

Lagopus is a software switch based on the OpenFlow 1.3 specification. Development began in 2013, and in July 2014 the source code was released under the Apache License. As of October 2014 the latest version is 0.1.1, and development is still ongoing. The main features of Lagopus are listed below.

- **A software switch for x86_64 Linux**
- **Single process, multi-threaded operation (two or more cores required)**
- **Mainly uses the C development language**
- **Based on OpenFlow 1.3.4**
- **Implements most of the options described in the OpenFlow specifications**
- **10 GbE wire-rate performance via packet forwarding with MPLS-VLAN mapping**
- **Supports large-scale configurations with up to 1 million flow entries and 254 tables**
- **High-speed packet processing via the user space using Intel DPDK**
- **Open source software usable for both commercial and non-commercial purposes, with the source code released under the Apache License**

In this report, we examine the performance of this implementation in detail. First, we will describe the main points that have a significant effect on software switch performance.

3.5 Software Switch Bottlenecks

3.5.1 Receive Interrupt and Transmission Complete Interrupt Processing

When the network interface card (NIC) detects the receipt of packets while the CPU is executing various processes, the NIC stores the packet byte string in buffer memory and issues an interrupt to the CPU. When the CPU detects an interrupt from the NIC, it temporarily saves the current process state (context) before dealing with the interrupt. The minimum amount of interrupt processing is executed, such as retrieving the packet information stored in the buffer memory, and putting it on the receive queue of the network stack. When this processing is complete, the saved context is restored, and processing from before the interrupt was issued is resumed. The act of switching from one context to another is called a context switch, and this creates a large load when transferring packets at high speed. Because generating an interrupt for each packet has a significant effect on performance, there are NIC functions designed to reduce the load by issuing interrupts once a certain number have accumulated.

3.5.2 Copying Packets to Memory

Received packets are first stored to the memory managed in the OS kernel space, but this is not normally directly readable by programs in the user space. The read (2) or recv (2) system calls copy the content of received packets to a memory space secured in advance by the user program. Conversely, the write (2) or send (2) system calls copy the content prepared by the user program to the memory space in the kernel. Because 14.88 million frames are sent or received each second at a wire-rate of 10GbE, for 64 byte packets a total of 1.9 GB (14.88 million x 64 x 2) of memory copying occurs per second. DDR3-1600 memory has a theoretical transfer rate of 12.8 GB per second, so even simply applying these results in approximately 15% of the total time being consumed for memory copying alone.

3.5.3 OS Process Scheduling

A number of processes are run in the user space of an OS, but as you can see by looking at the ps commands, even with the current popularity of multi-core processors we are not yet at the point where enough CPU cores are available. Consequently, the OS switches between running processes as appropriate to make it appear as if they are all running at once. Because switching is carried out at the millisecond level, it takes place too quickly for users to notice during normal use. However, if this kind of OS process scheduling occurs while a software switch is running, performance is naturally greatly impacted.

3.5.4 Memory Block Allocation and De-allocation, and Page Faults

Regarding the memory blocks used when reading or writing, it is not efficient to allocate them each time packets are received and de-allocate them each time transmission is completed. Considering that the next packets will be received again straight away, reusing the same space can alleviate performance degradation. However, not all memory in the user space is placed in actual memory. The OS maps user space memory and actual memory as necessary. This happens the instant a program attempts to access that memory. This system is called on demand paging, and the process triggers an exception the moment that a program attempts to access memory. Memory mapping is then executed during exception handling, and the program continues processing as if nothing has happened after it is restored. This exception is known as a page fault. It is obvious there is a large difference in processing time between cases in which memory mapping is executed when a page fault occurs, and cases in which memory mapping has already been carried out.

3.5.5 Exclusive Access to Resources

Above we explained that packet processing is executed by referring to a flow table under OpenFlow, but naturally it is also possible to add, change, and delete entries to a flow table during packet processing. However, a flow entry currently being referenced that matches a received packet must never be deleted. For this reason, the execution of packet processing and the addition, change or deletion of entries to a flow table must be mutually exclusive. In addition to flow tables, other resources that require exclusive access include LAN port counter information. A mechanism called a readers-writer lock is usually used for this, but this lock generates comparatively heavy processing, and repeated locking and unlocking of each packet would have a significant effect on transfer performance.

3.6 Resolving Bottlenecks With Intel DPDK

There are already a number of attempts to resolve several of the bottlenecks mentioned above, but here we will discuss the Intel DPDK initiative used with Lagopus. Intel DPDK (Data Plane Development Kit) is a library that provides useful API for programming data planes. It serves as a framework for enabling high-speed packet I/O. At the time of writing, the latest version is 1.7.1. The main features of Intel DPDK are as follows.

- **Runs on x86 and x86_64 Linux as well as FreeBSD**
- **Provided under a BSD License, so it can be used for both commercial and non-commercial purposes**
- **Poll Mode Driver (PMD) enables sending and receipt of packets without using interrupts**
- **Zero-copy via user space I/O**
- **OS scheduling avoidance designed for multi-core based on core fixing**
- **Page fault deletion using hugepage**
- **Optimized libraries including Longest Prefix Match, hash, and ring buffer**
- **Many sample applications included**

3.7 Performance Improvements in Lagopus

The use of Intel DPDK with Lagopus resolves many performance bottlenecks. In particular, PMD and core fixing contribute greatly to improved performance. The thread configuration of Lagopus is also quite distinctive (Figure 3).

The Lagopus data plane is divided into threads for handling packet I/O and threads for handling OpenFlow processing. Furthermore, it is possible to specify incoming (RX) and outgoing (TX) threads for I/O, and multiple threads for OpenFlow processing (workers) upon startup. These threads are each fixed to run on separate cores to prevent other processes being allocated to them, maintaining performance. A ring buffer provided by DPDK that does not require lock processing is used for communication between threads.

Bottlenecks are mainly resolved using Intel DPDK for basic packet I/O, but for OpenFlow processing Lagopus has its own performance improvements. The internal structure of the Lagopus flow table is shown in Figure 4.

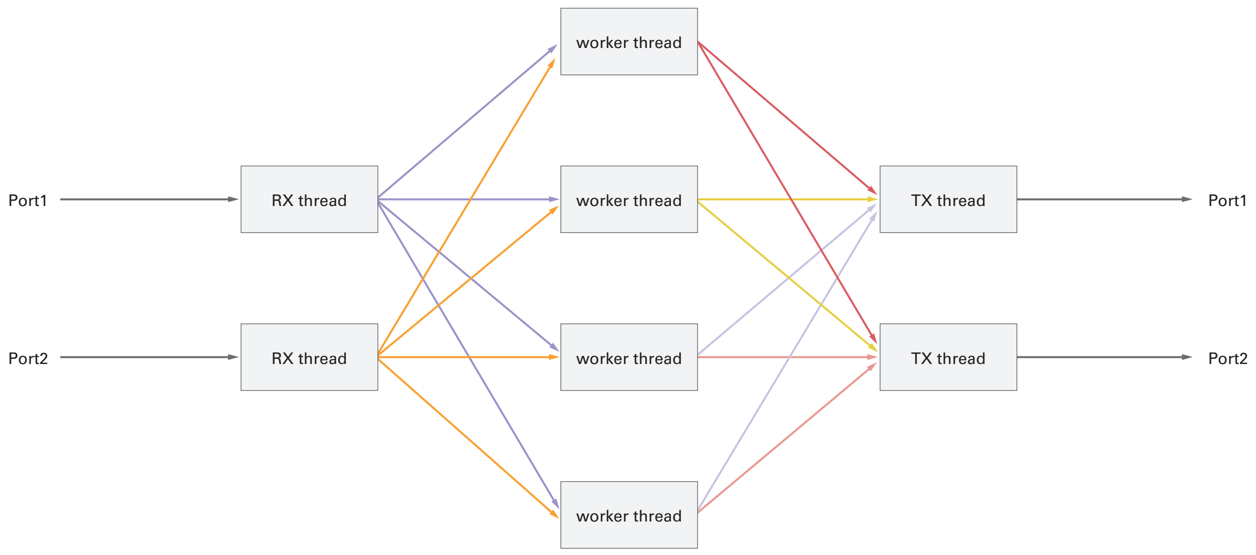

Figure 3: Lagopus Thread Configuration

Because it is necessary to output flow table details in response to requests from the controller, a flat table in line with the OpenFlow specification is available, but Lagopus also provides another table with a structure suitable for flow lookup. This structure is currently fixed, and not every flow table enables optimal lookup. However, it facilitates high-speed lookup of typical packet headers, such as branching by VLAN ID, MPLS label, or IPv4 destination address. A flow cache system is also implemented to cache the information for packets that have already been looked up once. When the same packets are received, flow entry information is obtained from the cache and the lookup itself is skipped.

Due to synergy from the above performance improvements, Lagopus was able to achieve 10GbE wire-rate performance (14.88 Mfps) for packet forwarding with packet header modification via OpenFlow, in a test environment using Intel Xeon processors with a flow table that contained 100,000 entries. The current focus for Lagopus is improving performance with more complex flow entries, maintaining performance when processing more ports simultaneously, and providing support for higher speed interfaces.

3.8 Conclusion

The Lagopus SDN software switch is highly compliant with the OpenFlow specification, and it enables dramatically improved performance in comparison to conventional software switches. Lagopus is currently prototype quality software that has not been sufficiently tested or validated outside a test environment. However, development continues with the assumption that it will see widespread adoption in real environments. If you are interested in learning more, please take a look at the website (http://lagopus.github.io).

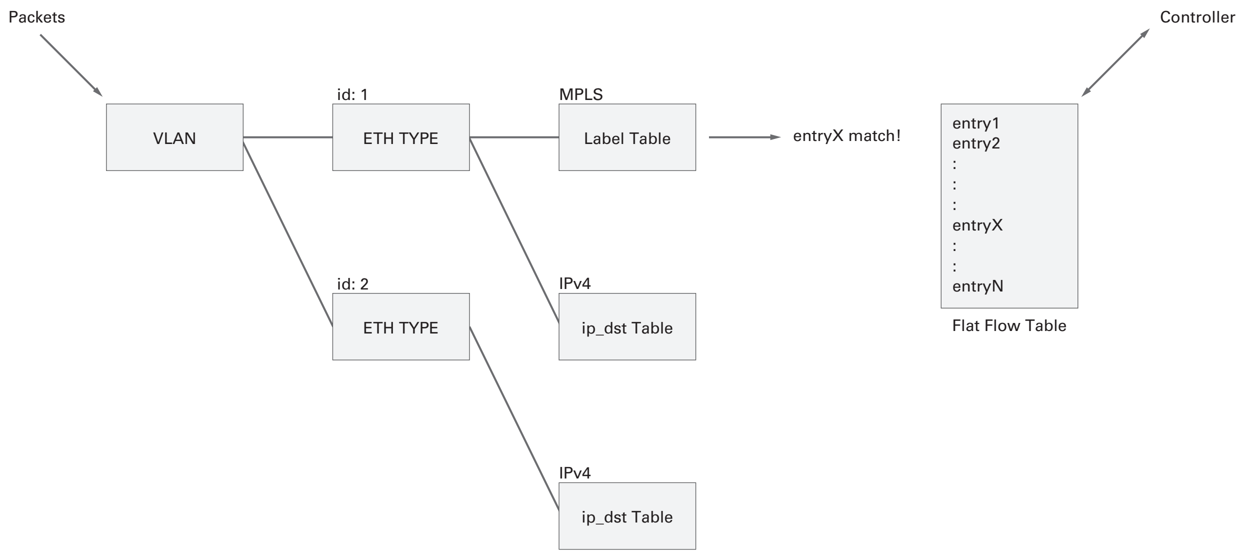

Figure 4: The Internal Structure of Lagopus Flow Tables

Author:

Masaru Oki
Research & Development Department, Stratosphere Inc.