Zebra 2.0 and Lagopus: newly-designed routing stack on high-performance packet forwarder

Kunihiro Ishiguro∗, Yoshihiro Nakajima†, Masaru Oki‡, Hirokazu Takahashi†
∗ Hash-Set, Tokyo, Japan
† Nippon Telegraph and Telephone Corporation, Tokyo, Japan
‡ Internet Initiative Japan Inc, Tokyo, Japan
e-mail: ishiguro@hash-set.com, nakajima.yoshihiro@lab.ntt.co.jp, m-oki@iij.ad.jp, takahashi.hirokazu@lab.ntt.co.jp

Abstract

Zebra 2.0 is the new version of open source networking software which is implemented from scratch. Zebra 2.0 is designed to supports BGP/OSPF/LDP/RSVP-TE and co-working with Lagopus as fast packet forwarder with Open-Flow API. In this new version of Zebra, it adapts new architecture which is mixture of thread model and task completion model to achieve maximum scalability with multi-core CPUs. Zebra has separate independent configuration manager that supports commit/rollback and validation functionality. The configuration manager understand YANG based configuration model so we can easily add a new configuration written in YANG. Lagopus is an SDN/OpenFlow software switch that is designed to achieve high-performance packet processing and high-scalable flow handling leveraging multi-core CPUs and DPDK on commodity servers and commodity NICs. Lagopus supports match/action-based packet forwarding and processing as well as encapsulation/decapsulation operation for MPLS and VxLAN. The inter working mechanism of the user space data plane of Lagopus and the network stack in kernel allows easy integration with Zebra 2.0 and its OpenConfigd.

Keywords

Routing stack, BGP, OSPF, LDP, RSVP-TE, Packet forwarder, YANG, OpenConfig.

Introduction

Zebra 2.0 is a sequel to GNU Zebra [5] / Quagga [13] which is widely used an open source routing software. Since first GNU Zebra was designed in 1996, the architecture start showing it's age. In past 20 years, there were a lot of technology evolution. Here we have re-designed a 21st century routing software from the ground up to match to the today's industry needs.

In this paper, we describe the design and implemenation of Zebra 2.0 and its integration to a high-performance software switch called Lagopus.

In the next section, we mention important issues to be solved in the first GNU Zebra. In Section III, we describe the basic design and implementation of Zebra 2.0. In Section IV, we present Lagopus software switch and its integration to Zebra 2.0. Finally, we conclude the paper.

First GNU Zebra architecture and its issues

When we designed the first GNU Zebra, the biggest ambition was to make multi-process networking software work. The first GNU Zebra is made from a collection of several daemons that work together to build the routing table. There may be several protocol-specific routing daemons and Zebra's kernel routing manager. Figure 1 shows the architecture of the first GNU Zebra. RIB (Routing Information Base) / FIB (Forwarding Information Base) and the interface manager are separated into an isolated process called 'zebra'. All of protocol handling is also separated to it's own process such as 'ripd', 'ospfd', 'bgpd' and so on. This makes easier to add a new protocol modules without re-building the whole staff. It increases the system stability - one process crash does not affect to other process. Most importantly it extend possibility of efficient use of multi-core CPU and / or distributed environments resources. The architecture succeeded and we believe that it shows the architecture works in various operational environments. At the same time there was several problems to be solved.

One of them is a performance issue. All of protocol modules are single threaded because matured threading library did not exist when we designed the first GNU Zebra. Therefore there must be careful thoughts about event management and non-preemptive manner of task execution. This is really annoying overhead to the programmer.

Another issue is to support various virtualized environments, such as VRF (Virtual Routing and Forwarding), VM (Virtual Machine) and Containers. When we designed the original Zebra architecture, virtualized environment was not so common as today. Thus we did not take into account of the possibility of the software runs on a virtualized environment.

Adding to that in past couple of years there are several new packet forwarding architectures and technologies, such as DPDK (Data Plane Development Kit)[3], OF-DPA (Open-Flow Data Plane Abstraction)[9] emerged to industry. That needs tight integration with networking software today.

Design and Implementation

We have designed Zebra 2.0 to achieve high-performance network protocol handling and high-extensible for various platform. Figure 2 shows the architecture overview of Zebra 2.0.

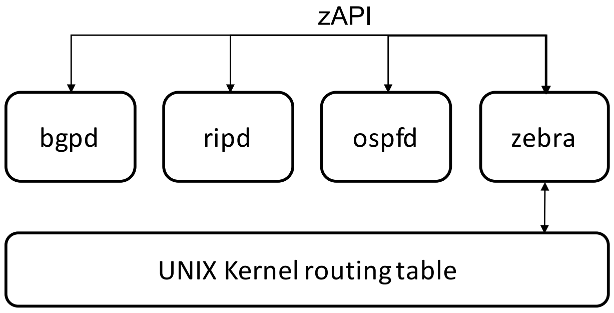

Figure 1: The architecture of the first GNU Zebra.

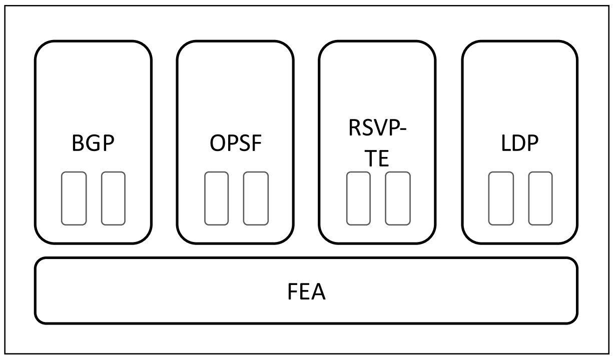

Figure 2: The architecture of Zebra 2.0.

New process and thread model

The biggest bottle neck of the original design was protocol handling affect to packet in/out and keep-alive timer. For example, a large SPF (Shortest Path First) calculation is required in a large-scale network on OSPF, OSPF can't handle packet in/out and keep-alive timer without voluntary yield of the execution of SPF calculation because it is single threaded. This lead OSPF to Hello time out. That was the typical scenario of the performance issue.

Zebra 2.0 employs totally new process and thread model to achieves drastically improved performance and stability. In Zebra 2.0, the protocol handler and packet in/out (including keep alive timer) handler are executed in a separate thread. By default, when a new protocol module is instantiated, two threads are created. One is for the protocol handling, the another one is for packet in/out and time out handling. In addition, a lock-free queue between these two threads is introduced.

When a control packet comes in, the packet is handled by a packet handler thread. The packet handler thread is supposed to update keep-alive timer and to validate the packet, then the thread puts into the shared lock free queue. With this architecture, protocol handling, such as SPF calculation, large scale BGP table updates, do not interfere with keep-alive timers.

As we learned from nginx[7] and node.js[8] achievement, the task completion model with single thread has a lot of advantage over to multi-thread model in performance stand point. In Zebra 2.0 we combine the best part of task completion model and multi thread model.

Separation of Configuration Manager

In Zebra 2.0, configuration manager is not part of the implementation. Today's industry require having a single integrated configuration manager which supports commit/rollback with well designed configuration database. The configuration manager handles all of the configuration on the box or even boxes. To do that, we decouple configuration manager from Zebra 2.0 then we introduce its own independent software package called 'OpenConfigd'.

The OpenConfigd generate configuration schema described in YANG[1]. Several interfaces and APIs such as CLI (Command Line Interface), NETCONF and REST API are automatically generated by OpenConfigd with YANG model. OpenConfigd integrates OpenConfig[11] model, that aims to develop programmatic interfaces and tools for managing networks in a dynamic and vendor-neutral way.

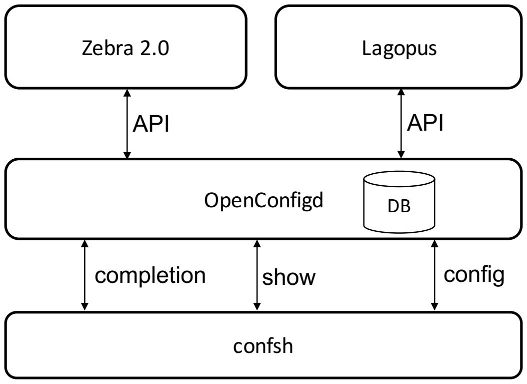

Figure 3: The architecture of the OpenConfigd.

OpenConfigd has user interface shell called 'confsh'. It communicates the configuration manager to show possible completion and show to execute the command. In Zebra 2.0, all of the protocol modules has configuration set/unset/validate callback functions which will be invoked from OpenConfigd. OpenConfigd use gRPC for transport to allow easy micro-service integration.

Forwarding Engine Abstraction

In past couple of years, new forwarding technology was invented. DPDK is one of the example. It delivers the packet directly from device buffer to user-space packet forwarder. To handle the architecture, we need to have well designed FEA (Forwarding Engine Abstraction). Other example is OF-DPA that wraps hardware functionality provided by merchant switch silicon with OpenFlow like APIs. With OF-DPA, hardware forwarder configuration is much easier than it used to be. Software forwarder and hardware forwarder is pretty different. Therefore it require different API calls with different kind of objects as arguments. In addition, sometimes it require different API call sequences. Much worse, there might be a situation of supporting both software forwarder and hardware forwarder at the same time to handle slow path.

In Zebra 2.0, new packet forwarding abstraction layer called FEA is introduced. FEA has sets of API which can automatically handle various different configuration underneath of packet forwarders. FEA provides primitive control and configuration API for packet forwarder including interface/port management, bridge management, routing table, ARP table. A protocol modules must attach FEA and call FEA APIs for packet forwarder controls.

Virtualization

Virtualization support is inevitable to today's networking industry for an execution environment of Zebra 2.0. A typical situation is running a networking protocol instance on virtual machine. Other case is running a networking protocol instance in a container-based environment such as Docker[2].

In addition, network virtualization support is also inevitable today for many use cases. The required network virtualization functionalities includes VRF, network name space, overlay networking and so on. Sometimes, a logically-centralized protocol handling or control for network virtualization is required

Hence all these require is the separation of networking protocol instance from underlying networking stack. In Zebra 2.0, packet forwarding instances called FEA and protocol module instance is completely separated. To support normal Linux software forwarder with single OSPF instance, C++ class called FEALinux is instantiated, then it will be attached to OSPF instance. User can find any combination of FEA and protocol module is possible. That could be 1:1, 1:n, or n:1.

OpenFlow Support and Other Software Integration

Integration with other software is very important factor to today's software. No software can solve all of the issues by itself and Zebra 2.0 is no exception. Especially Zebra 2.0 needs to leverage state-of-the-art packet forwarder technology to realize IP routing, MPLS label switching, L3-VPN and so on. These packet forwarder is supposed to be controlled through FEA instance in Zebra 2.0 as mentioned before.

OpenFlow switch[4] are one of interesting system to be integrated to Zebra 2.0. The OpenFlow switch provides flow-based switch abstraction APIs for network application and hides complexities of switch implementation for application developers. We have developed high-performance/scalable OpenFlow software switch called "Lagopus". Zebra 2.0 leverage the outcome of Lagopus Project and Lagopus works as one of the protocol module of Zebra 2.0 to support OpenFlow switch. One merit of using Lagopus as packet forwarder, Zebra 2.0 takes advantage of DPDK for high-performance packet processing on Intel x86 servers.

Zebra 2.0 is implemented in C++11 standard. Therefore integration with Java based OpenFlow controller, such as Open Daylight[12] and ONOS[10], we use SWIG[14] to encapsulate C++ class to Java. With SWIG, it is possible to integrate with other languages such as Python and Go.

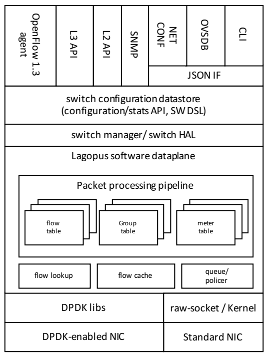

Figure 4: The design of Lagopus switch.

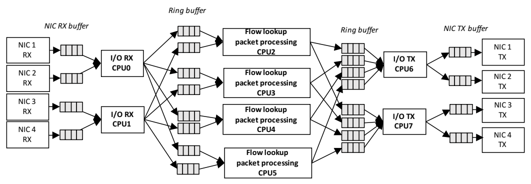

Figure 5: Packet processing on Lagopus dataplane.

Lagopus software switch

Lagopus[6] is a high-performance SDN/OpenFlow software switch with highly-programmable and flexible packet processing pipeline. Figure 4 shows the design of Lagopus software switch.

Lagopus provides match/action-based flow-aware dataplane APIs to routing applications. Lagopus supports multi-layer flow lookup and various protocol frame processing functionality in software dataplane. General tunnel encapsulation/decapsulation is available to realise overlay networking and VPLS (Virtual Private LAN Service) and VPWS (Virtual Private Wire Service) with MPLS technology.

Lagopus dataplane is designed to exploit the power of multi-core CPU and Ethernet NIC for performance and scalability. Figure 5 shows the implementation of packet processing on Lagopus dataplane with multi-core CPUs. Lagopus leverages DPDK technologies to achieve high-performance packet forwarder in user space on commodity Intel x86 servers. For user space or kernel routing stack on Linux, Lagopus provides packet in/out function of control-plane related packet escalation with tap interface that corresponds to the dedicated interface/port in Lagopus dataplane. The Lagopus dataplane achieves more than 20 MPPS (Packet Per Second) with the conditions of over-1M flow entries.

To enable more flexible swtich configuration for a network managment system, Lagopus provides lots of API and interfaces, such as OpenFlow, CLI, REST API, and SNMP.

Zebra 2.0 Integration

To integrate Lagopus to Zebra 2.0, the following adaptations are required:

- Dataplane configuration

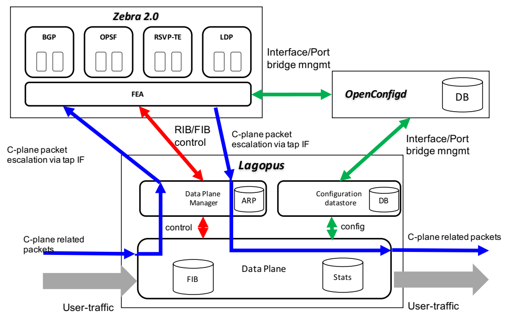

Figure 6: Zebra 2.0 and its integration to Lagopus.

The configuration of Lagopus is supposed to be achieved by OpenConfigd. OpenConfigd tells the configuration datastore of Lagopus to configure Lagopus dataplane, for example, which interfaces are used for dataplane.

- Dataplane control

  FEA of Zebra 2.0 installs RIB/FIB to Lagopus dataplane as fast packet forwarder. Lagopus dataplane manager recieves the series of request to control L3/L2 then, the manager converts the series of request to the match/action-based flow rules, which are FIB on Lagopus dataplane. The dataplane manager of Lagopus sends flow statistics for FEA.

- Control-plane-related packet escalation

  To facilitate packet in/out functionality on Zebra 2.0, a set of match/action rules for control-plane-related packet escalation and peer communication establishment, such as BGP, OSPF, RIP and LDP, is supposed to be install to Lagopus dataplane before the packet forwarding starts. If a protocol handler performs packet out to an opposite routing node, the Lagopus data plane receives the packets from Zebra 2.0 and sends them through a dedicated interface. When the new routing configuration is requested to Zebra 2.0, a set of match/action rules are installed to Lagopus dataplane dynamically.

Conclusion

It has been a long time since we've designed the first version of GNU Zebra and there are a lot of things happened in industry and surrounding environment. Here we have an opportunity to re-design Zebra, we hope that this is useful to people who are interested in networking software.

The Zebra 2.0, OpenConfigd and Lagopus integration are avialable soon.

- Zebra 2.0
  https://github.com/hash-set/Zebra-2.0
- OpenConfigd
  https://github.com/hash-set/OpenConfigd
- Lagopus
  https://lagopus.github.io/

References

[1] Bjorklund, M. 2010. Yang - a data modeling language for the network configuration protocol (netconf). Technical report, IETF. "RFC6020".

[2] Docker. https://www.docker.com/.

[3] Data plane development kit. http://dpdk.org/.

[4] Foundation, O. N. 2015. Openflow switch specification version 1.3.5.

[5] Gnu zebra. https://www.gnu.org/software/zebra/.

[6] Lagopus switch: a high performance software switch. http://lagopus.github.io/, https://github.com/lagopus/lagopus.

[7] nginx. http://nginx.org/en/.

[8] node.js. https://nodejs.org/.

[9] Of-dpa: Openflow data plane abstraction. https://github.com/Broadcom-Switch/of-dpa.

[10] Onos (open network operating system). http://onosproject.org/.

[11] Openconfig. http://www.openconfig.net/.

[12] OpenDaylight. https://www.opendaylight.org/.

[13] Quagga routing suite. http://www.nongnu.org/quagga/.

[14] Swig: Simplified wrapper and interface generator. http://www.swig.org.