The Runos OpenFlow Controller

Alexander Shalimov
Applied Research Center for Computer Networks
Lomonosov Moscow State University
ashalimov@arccn.ru

Sergey Nizovtsev
Applied Research Center for Computer Networks
snizovtsev@arccn.ru

Danila Morkovnik
Applied Research Center for Computer Networks
Lomonosov Moscow State University
dmorkovnik@arccn.ru

Ruslan Smeliansky
Applied Research Center for Computer Networks
Lomonosov Moscow State University
smel@arccn.ru

***Abstract*—The Runos is a C++ OpenFlow controller that has been developing since 2014 in order to answer on the well-known question "*Could an OpenFlow controller be both easy to develop applications for and also high performance?*" [1]. The controller includes the most fruitful techniques from the latest research on simplifying SDN programming such as Pyretic and Maple and combines them in right way to achieve high performance and production quality, programmability, usability. Runos is widely used in different POCs showing interests for third-party developers. The project is in http://arccn.github.io/runos/.**

I. Introduction

The performance of OpenFlow controllers that has been widely discussed for a couple years [2]. But despite the fact of that there are a lot research on SDN programming and policies, they are not used in production SDN controllers. Applications for modern controllers are still developed as solid block of code created by single group of developers. The controllers still have a problem with running multiple third party application independently and need a time for manual connecting them in one execution pipeline (parameters modification, interfaces binding, assigning different flow tables, etc). The controller also don't have built-in flow rules conflict resolution system and thus applications might send overlapping rules. The simplest example is having two applications: forwarding and span. Forwarding application says this flow should go over port 1, while Span application wants the flow to be mirrored over port 5.

This paper describes Runos OpenFlow controller: architecture decisions, performance evaluation, programmable simplicity, and two usecases on using the controller.

II. Design

A. Language and tools

In order to enable high performance solution we use C++11 language. Comparing to Java, C++ produces fully compiled code reducing intermediate overheads and can achieve lower delays: 1. C++ has wide low level network primitives that fit high performance requirements; 2. we also can use fast packet processing libraries like DPDK and Netmap; 3. C++ has more advanced multithreading locking techniques. As a runtime we use QT and Boost.ASIO. They provide us wide inter- and inner- thread communication with dynamic decisions on having separate event loop for modules. This simplifies multicore scalability: we run all modules as logical QT threads and depending on number of physical cores the runtime chooses what logical threads should be combined to run on a single physical thread (based on configuration file). QT also has in-build signal/slots mechanism allowing easily subscribe on controller services from new applications.

B. Architecture

In order to make the controller faster, we have to rely on multicore system with threading mechanisms. Runos runs the dedicated number of worker threads (libfluid - http://opennetworkingfoundation.github.io/libfluid/) to communicate with network switches and the dedicated number of threads to run network applications. We use run to completion model with pipelines. Applications say their requirement on when they should be executed (e.g., after firewall, but before load balancer). The controller determines the final order and creates the pipelines (see next section). The pipeline executes in libfluid threads' context. Applications register their own short callbacks in the pipeline (e.g. hash lookup based on mac learning table). For deeper processing the OpenFlow message should be moved to the associated QT application thread.

III. Features

A. Speed

Figure 1 shows the renewed throughput for the existing controller against Runos with 8M flows per second (the line named easy). In our previous work [3], we have created in-kernel and fusion controller where the controller resides as a module in the Linux kernel that has the fastest performance. This also shows how improving application programmability affects overall performance of the controllers)from the inkernel version down to the easy version).

B. Modularity

Effective multi-threading. Runos uses multi-thread asynchronous task from Boost.Asio (together with QT SDK). It spawns a fixed number of threads running Boost.Asio tasks

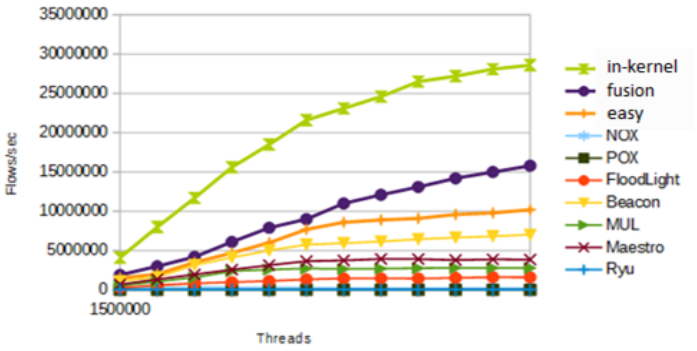

Fig. 1. The average throughput achieved with different number of threads (with 32 switches, $\mathbf{10^5}$ hosts per switch)(Intel(R) Xeon(R) CPU E5645 2.40GHz)

synchronized by strands to reduce context-switching overhead (strand is defined as strictly sequential invocation of event handlers i.e. threads without explicit locking). This approach is often called as "green threads" or "userspace threads". It allows to take advantage of both preemtive (kernel) and cooperative (userspace) scheduling models.

Support of user-defined custom OXM fields. With Runos we can add arbitrary fields types if the custom switch supports them. Pluggable OXM field architecture lets us to define our-own fields and use them together with flow rule compiler. In the future, we can easily integrate with P4 for defining exact packet processing logic in the switch.

C. Programmability

Algorithmic policies. Maple engine [4] is tightly integrated with Runos and makes applications shorter, maintainable and reliable. Maple generates and updates flow entries for programmers and ensures no rule collisions. A programmer uses load, compare or rewrite operators in imperative manner (see next section). Runos will generate correct minimal-match flow rules for switches.

Client-friendly API using EDSL grammar. Runos lets you think that packet's header fields are just variables. We can index, compare or modify them without knowledge about underlying Maple engine, openflow protocol details, and table features. Basically, instead of manually using Maple's load, match, compare primitives we encode them in C++ operators: *pkt[eth_src] == eth_addr* (this will produce Maple's compare primitive for programmers). Also thanks to Boost.Proto expression templates, Runos gets the whole expression and does some compile-time symbolic optimizations that reduces number of flow rules. For example, $if\ (eth_{src}\ ==\ A\ ||\ eth_{dst}\ ==\ B)\ doA\ else\ doB$. Runos sees the entire expression and understands that it's enough to use three openflow rules to express this policy instead of four as in raw Maple engine that works with all comparison separately.

Pyretic-like composition of modules. You can use parallel and sequential modules composition operators to statically configure packet handling pipeline. Based on Pyretic style policies [5], Runos arranges packet handlers into arbitrary tree-like processing factory using static definitions. This is a step further from linear modules composition as in other controllers.

D. Applications

Enterprise/Campus Networks The Runos is used as main controller for EasyWay management system [6]. Easyway implements semantic network management where administrators simply say what they want from the network but not how and a SDN/OpenFlow controller will automatically program the network elements. All low level details are hidden from administrators (e.g., choosing IP addressing scheme). From Runos point of view, Easyway is a network application that widely leverages different services like like device and link monitoring, topology, routing with QoS support, DNS proxy, DHCP proxy, ARP proxy, BGP, load balancing, firewall, ACL, NAT.

WAN segment (Service Provider) The Runos is also used to manage dedicated WAN segment. It is green field deployment where we have two types of physical devices: two distribution OpenFlow switches (400Gbps) connected with set of aggregation OpenFlow switches (40Gbps). One aggregation switch is for one place (city, town, village). The goal is to implement all required services for end-host communication. Runos has the following list of applications: L2 pseudo wires (PBB based), bridge domain, multiple pathes (active/standby), VLAN enumeration, hierarchical QoS, L3 multicast, BFD extension for OpenFlow.

IV. Conclusion

The Runos OpenFlow contoller introduces novel combination of the latest techniques on simplifying programming with SDN together with high performance and quality. The controller provides carrier grade performance, understandable programming model for network developers, and composition language suitable for network administration.

Acknowledgment

This research is supported by the Ministry of education and science of the Russian Federation, Unique ID RFMEFI60914X0003.

References

[1] David Erickson, The Beacon OpenFlow Controller, Proceeding of the ACM SIGCOMM HOTSDN 13, Hong Kong.

[2] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, R. Smeliansky, *Advanced Study of SDN/OpenFlow controllers*, Proceedings of the CEE-SECR13: Central and Eastern European Software Engineering Conference in Russia, ACM SIGSOFT, October 23-25, 2013, Moscow, Russian Federation

[3] Shalimov A., Ivashchenko P. Inkernel offloading of an SDN/OpenFlow Controller Proceedings of the Modern Networking Technologies (MoNeTec), IEEE, Moscow, Russia, 2014

[4] Andreas Voellmy, Junchang Wang, Y. Richard Yang, Bryan Ford, and Paul Hudak. Maple: Simplifying SDN programming using algorithmic policies. In SIGCOMM, 2013

[5] J. Reich, C. Monsanto, N. Foster, J. Rexford and D. Walker "Modular SDN programming with pyretic", USENIX Mag., vol. 38, no. 5, 2013

[6] A. Shalimov, D. Morkovnik, S. Nizovtsev, R. Smeliansky EasyWay: Simplifying and automating enterprise network management with SDN/OpenFlow// 10th Central and Eastern European Software Engineering Conference in Russia, CEE-SECR 2014, ACM SIGSOFT, Moscow, Russia.