High performance in-kernel SDN/OpenFlow controller

Pavel Ivashchenko, Alexander Shalimov, Ruslan Smeliansky
Applied Research Center For Computer Networks
pivaschenko@arccn.ru, ashalimov@arccn.ru, rsmeliansky@arccn.ru

Abstract

This article demonstrates capabilities of the in-kernel openflow controller which leverages ability of the contemporary multicore systems with reducing host network communication overhead in Linux OS. The measurements show the in-kernel controller has throughput 4 times more and latency two times less that the existing OpenFlow controllers.

1 Introduction/Motivation

SDN/Openflow is the most innovative technology in the area of computer networks of recent years [1]. It allows us to automate and simplify network management: fine-grained flows control, observing the entire network, unified open API to write your own network management program, and so on. All control decisions are done first in a centralized controller and then moves down to overseen network's switches. In other words, the controller is a heart of SDN/OpenFlow network and its characteristics determine the performance of whole networks. The controller throughput means how big and active our network can be in terms of switches and hosts. The response latency directly affects network's congestion time and end-user QoE.

The latest SDN/OpenFlow controllers performance evaluation [3] shows that the maximum throughput was demonstrated by the Beacon OpenFlow controller [4] with 7 billions flow requests per second. However, this is not enough and in data centers we need several times more performance [2]. According [6], for the small datacenters with 100K hosts and 32 hosts/rack, the maximum flow arrival rate can be up to 300M with median rate around 10M and minimal rare around 1.5M. In the large-scale networks the cituation can be tremendously worse.

There are two non-mutually exclusive ways to cover the performance gap. The first ways is to use multiple instances of controller collaboratively managing the network and forming a distributed control plane [2, 5]. But this brings a lot of complexity and overheads on maintaining a consistent network view between all instances.

The second way is to improve single controller itself by leveraging ability of contemporary multicore systems and by reducing existing bottlenecks and overheads. The network layer of OpenFlow Controllers is the most time consuming part [2]: reading incoming OpenFlow messages from the NIC and communicating with OpenFlow switches. For the last task the common approach is to use multithreading. One thread listens the socket for new switch connection requests and distributes the new connections over other working threads. A working thread communicates with the appropriate switches, receives flow setup requests from them and sends back the flow setup rule. There are a couple of advanced techniques. For instance, Maestro distributes incoming packets using round-robin algorithm, so this approach is expected to show better results with unbalanced load. The first task usually relies on the runtime of chosen programming language. Let's consider this problem in details.

From the system view, a OpenFlow controller is a TCP server running in Lunix userspace. Every system call (malloc, free, read and write packet(s) from the socket, etc) leads to context switching between userspace and kernel space that requires additional time. Approximately this time for FreeBSD Linux is 0.1ms and takes 10% time for whole system call. Under the high load this leads to significantly time overhead. Moreover, the userspace programs work in virtual memory that also require additional memory translation and isolation mechanism. These issues can be avoid if the OpenFlow controller will reside in the Linux kernel.

In this paper, we presents a novel OpenFlow controller that works as a module inside the Linux kernel and has fastest throughput and the lowest latency comparing with all existing OpenFlow controllers.

2 Arhitecture

Our openflow controller has 3 logical parts.

- **Server**

  Server is a kernel thread. It listens socket, accepts new connections from switches and finds appropriate backend's thread. Sever distributes connections between backend's threads evenly. Then it gives away accepted to frontend.

- **Frontend**

Frontend checks switches for different errors, for example openflow version, correctness of openflow messages such as hello, features reply. It checks correctness of headers for every messages in the input buffer until a features reply is not sent. After that frontend checks uniqueness of datapath ID and writes necessary information about switch to special data structure. If all verification is done, frontend gives away connection to appropriate backed's thread.

- **Backend**

  Number of backend's threads set in config file. Backend is a final stage of connection's journey through controller arhitecture. Backends work with switches and applications. They send and receive openflow messages. Poll wait for some event on a file descriptors of switch's sockets. Then we read input buffer and fill output buffer. Output buffer will be send if one is overflow or end the input buffer.

This three-tier architecture protects from unnecessary work high-speed backends.

3 Experimentation results

The figure 1 shows the maximum throughput for different number of available cores per one controller. The single threaded controllers (Pox and Ryu) show no scalability across CPU cores. The performance of multithreaded controllers increases steady in line for 1 to 6 cores, and much slower for 7-12 cores because of using hyper threading technology (the maximum performance benefit of the technology is 40%).

The average response time of the controllers demonstrates insignicant correlation with the number of connected hosts. For the average response time with one connected switch and 10^5 hosts see Figure 2. The smallest latency has been demon- strated by In-kernel, MuL and Beacon controllers, while the largest latency is typical of python-based controller POX.

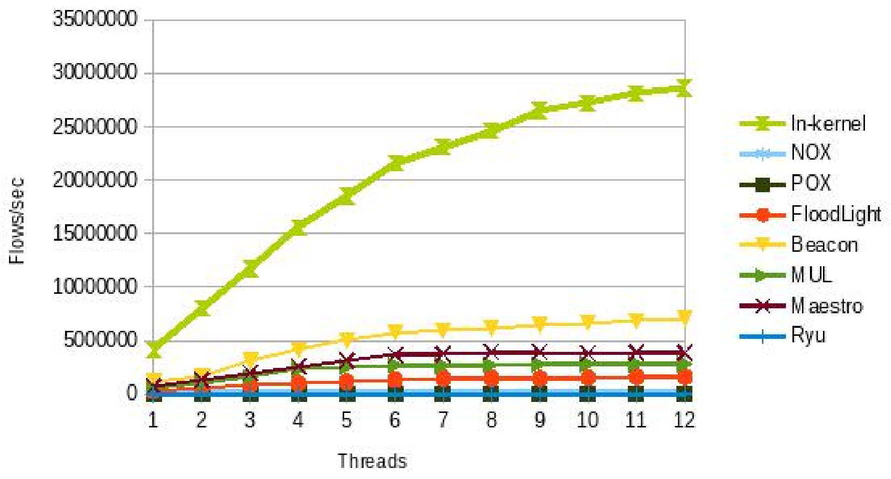

Figure 1: The average throughput achieved with different number of threads (with 32 switches, 10^5 hosts per switch)(Intel(R) Xeon(R) CPU E5645 @ 2.40GHz)

In-Kernel	45
NOX	91
POX	323
Floodlight	75
Beacon	57
MuL	50
Maestro	129
Ryu	105

Table 1: The minimum response time (10^{-6} secs/ flow)

We compare performance of our In-kernel openflow controller and beacon with stress as flood(3M connections per second). The controllers were 2 threads. Flood generated by cbench with special parameters. Beacon without flood show 4M flows/sec, In-kernel - 16.9 M flows/sec. Performance beacon with flood attack is wane (1 M flows/sec), but our controller show 16.7M flows/sec. (CPU E3-1240 V2 @ 3.40GHz)

References

[1] M. Casado, T. Koponen, D. Moon, S. Shenker. *Rethinking Packet Forwarding Hardware.* In Proc. of HotNets, 2008

[2] A. Shalimov, R. Smeliansky, *On Bringing Software Engineering to Computer Networks with Software Defined Networking*, Proceeding of the 7th Spring/Summer Young Researchers' Colloqium on Software Engineering (SYRCoSE 2013), May 30-31, 2013, Kazan, Russia

[3] 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

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

[5] Advait Dixit, *Towards an Elastic Distributed SDN Controller*, Proceeding of the ACM SIGCOMM HOTSDN 13, Hong Kong.

[6] T. Benson, A. Akella, D. Maltz, *Network traffic characteristics of data centers in the wild*, IMC, 2010