High performance in-kernel SDN/OpenFlow controller

Pavel Ivashchenko
Applied Research Center For Computer Networks
pivaschenko@arccn.ru

Alexander Shalimov
Applied Research Center For Computer Networks
ashalimov@arccn.ru

Ruslan Smeliansky
Applied Research Center For Computer Networks
rsmeliansky@arccn.ru

Abstract

This paper demonstrates capabilities of the in-kernel OpenFlow controller which leverages abilities of the contemporary multicore systems with reducing host network communication overhead in Linux OS. The measurements show the in-kernel controller has the fastest throughput and the lowest latency almost two times less comparing with 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 the whole network. 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 with 7 billions flow requests per second. However, this is not enough for data centers where we need several times higher performance [2]. According [5], for the small data centers with 100K hosts and 32 hosts/rack, the maximum flow arrival rate can be up to 300M with median rate around 10M and minimal rate around 1.5M. In the large-scale networks the cituation can be tremendously worse.

There are two non-mutually exclusive ways to cover this performance gap. The first way is to use multiple instances of a controller collaboratively managing the network and forming a distributed control plane [2, 4]. 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 point of view, an OpenFlow controller is a TCP server running in Linux 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: hierarchical vs linear address translation. Our measurements show that for memory intensive application this difference is up to 4 times. These issues can be avoided 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 Architecture

Our openflow controller has 3 logical parts.

- **Server**. Server kernel thread listens to a socket, accepts new connections from switches, and evenly distributes connections between frontends.

- **Frontend**. Frontend threads initialize connections and check their correctness: the version of OpenFlow, hello, features reply. The correctness of headers are checked for every messages in the input buffer until a features reply OpenFlow message will be sent. If all verification is done, connections go to backeds.

- **Backend**. Backend threads work with switches and applications. They do the main job on sending and receiving OpenFlow messages. Inside the thread we use poll() to wait for changes in the sockets' descriptors. We use per-switch input and output buffers 512 Kbytes length.

This three-tier architecture protects the controller from the attacks from control plane when someone tries to initialize many dummy connections to the controller (like SYN flood).

3 Experimentation results

For performance evaluation we use the methodology described in [3]. One different is that the performance of single cbench is not enough to fully load the in-kernel controller. So, our test-bed consists of two servers connected with two 10Gb links and two cbenches generating the packetin messages over these two links.

The figure 1 and table 1 shows the renewed throughput and latency numbers for the existing controller against the in-kernel controller. The throughput of the in-kernel controller is 25Mfps that 4.5 times higher than Beacon. The latency of the in-kernel controller is 45us that 1.5 times less than Beacon.

We also compare the performance of the in-kernel OpenFlow controller and Beacon against flood attacks. The controllers run on 2 threads. the flood generator sends 3M dummy TCP connections per second. The Beacon's performance decreases almost to zero (5 times less). The in-kernel controller shows almost the same performance (less than 2%).

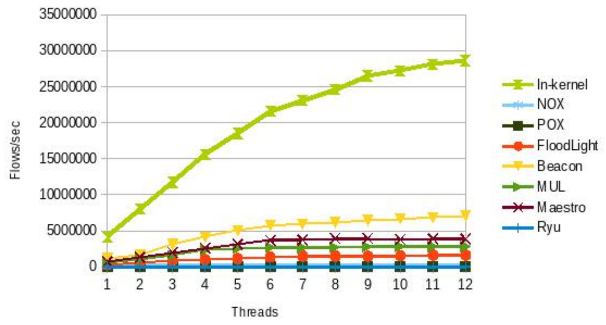

Figure 1: The average throughput achieved with different number of threads (with 32 switches, $\mathbf{10^5}$ hosts per switch)(Intel(R) Xeon(R) CPU E5645 2.40GHz)

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)

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

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