RDNA Balance: Load Balancing by Isolation of Elephant Flows using Strict Source Routing

Rodolfo V. Valentim, Rodolfo S. Villaca, Moises R. N. Ribeiro, Magnos Martinello
rodolfo.valentim@aluno.ufes.br
Federal University of Espírito Santo (UFES)
Espírito Santo, Brazil

Cristina K. Dominicini, Diego R. Mafioletti
cristina.dominicini@ifes.edu.br
Federal Institute of Education, Science and Technology of Espírito Santo (IFES)
Espírito Santo, Brazil

ABSTRACT
Data center networks need load balancing mechanisms to dynamically serve a large number of flows with different service requirements. However, traditional load-balancing approaches do not allow the full utilization of network resources in a simple, programmable, and scalable way. In this context, this paper proposes RDNA Balance that exploits elephant flow isolation and source routing in core nodes. Flow classification operations are performed on the edge using features of the OpenFlow protocol. The results show that with this approach it is possible to provide a simple, scalable, and programmable load balancing for data centers.

1 INTRODUCTION
Data Center (DC) administrators have to attend to several flows with conflicting network requirements due to the great variety of services and applications that needs to be executed in DCs [5]. These problems demand the existence of an efficient mechanism for Traffic Engineering (TE). Such TE mechanism needs to be able to classify the existing flows, isolate flows with conflicting network requirements and assign paths in a way that meets the flow requirements without compromising other flows.

Usually, Data Center Networks (DCNs) have complex and rigid routing mechanisms relying in a table-based routing. This approach has known scalability issues [3] and brings a high level of complexity to manage the network state.

Modern DCNs present intense communication between servers and load balancing mechanisms and have to deal with routing in the core, which can be quite complicated. On average, every 1 ms, 100 new flows arrive the DCN; flows up to 25s are responsible for more than half the amount of traffic; and only 0.1% of the total of flows lasts longer than 100s, accounting for almost 20% of the data volume transferred [5]. Another important observation is that a small fraction of the links experiences much greater losses than the rest of the network. It is thus demonstrated that alternative paths can be used to avoid losses and to improve the quality of service provided for network flows in a DC [1].

2 RDNA BALANCE
This work elaborates a solution for the load balancing problem in DCNs, named RDNA Balance. We leverage the SSR mechanism used in the RDNA architecture, based on fabric core nodes and programmable edge nodes, to perform load balancing at the source in a reactive and centralized way. The mechanism aims to isolate elephant flows from mice flows to improve the bandwidth available to the elephant flows while decreasing the latency experienced by mice flows.

Source Routing (SR) mechanisms decrease the size of routing tables and reduce the overload in the control plane when compared with traditional approaches [7]. Strict Source Routing (SSR) methods allows to specify the entire underlay path of a flow using a routing information in the packet. According to [4], such routing solutions using SSR reduce the amount of flow rules installed in the network core. The lower the number of flows the better is the scalability.

The use of SSR in a DCN is a promising way to tackle TE problems. The Residue Defined Network Architecture (RDNA) is a network architecture proposed by [6] that explores the Residue Number System (RNS) and SSR to perform packet routing in the network core. RDNA separates core and edge elements of the network. The core elements are simple and route packages using module operations without tables.

The RDNA architecture is composed by the three elements represented in the Figure 1(a): i) RDNA Controller, a logically centralized controller used to configure polices and manage the switches; ii) edge switches, which insert route identifiers in the packets specifying a path to the flow; and iii) core switches that forward packets based on a modulo operation (remainder of division) between the route identifier and the switch identifier. For instance, a package with route identifier \( R = 133 \) when enters a switch with identifier 11 will be forwarded according to the modulo operation \( < 133 >_{11} \). As the result is 1 the package will leave the Switch 11 by the port 1 and enter the Switch 19.
In Figure 1(a), the interaction between RDNA Balance and a RDNA based DCN is presented. In order to simplify the communication, all the interaction is done via the RDNA controller. The four functional blocks and the database represent the main functionalities required to perform an effective load balancing via RDNA Balance.

All functional blocks have well defined functions in the load balancing task: i) Data Collector monitors the DCN and collects topological and network usage data; ii) Flow Classifier classify flows by their types; iii) Decision Maker take load balancing decisions according to the current network state and; iii) Route Manager acts over the network via RDNA Controller based on the load balancing decision. This work focus on the the Route Manager, i.e., the mechanisms on the network devices in order to obtain a reactive, centralized and host-based load balancing.

In RDNA Balance, when the Decision Maker decides to migrate a flow based on the current state of the network it communicates the Route Manager. The task to migrate a flow from one route to another involves the installation of two OpenFlow rules in the edge switch source and destination. The rule in the source inserts the route ID field “Source MAC Address” in the packet’s Ethernet header and the rule in the destination restores the package header. In a traditional network, the task to reroute a flow would require to change in the state of all the switches in the path between source and destination which takes longer than the RDNA Balance approach and it becomes harder to manage as the number of hops in the path increases.

A prototype was implemented using OpenStack as the cloud manager framework. Edge and core switches were virtualized using a customized version of OpenvSwitch (OvS), in which the forwarding is based on modulo operations, implemented at kernel-level by modifying the OvS implementation. Each switch is represented as different OvS bridges.

The first test verified the packet loss during route migration. In this experiment, a UDP elephant stream with different bandwidth requirements was created, ranging from 100 Mbps to 800 Mbps, doubling the throughput at each iteration. The original route is: VMS1 → S11 → S19 → S17 → VMD2. After 50s, the route is migrated to a path with the same length: VMS1 → S11 → S13 → S17 → VMD2. In Figure 1(b) shows the throughput at the destination VMD1 for each one of the evaluated throughput sent by VMS1.

Consider the scenario proposed in Fig. 1(c) where two flows share the same link: Flow 1 uses UDP protocol with a throughput of 930Mbps and is characterized as a elephant flow; Flow 2 sends a ICMP message every 1s and is characterized as a mice flow. Flow 1 is generated using pktgen which is capable of generating packets of specific sizes at a constant rate. Pktgen generated UDP packages at a rate of 81274pps (packets per second) and packet size of 1518Bytes. At this rate, Flow 1 saturated the physical limit of 930 Mbps link. The rate and packet size where chosen according to [2, 8].

At 30s, the path of Flow 1 is migrated from VMS1 → S11 → S19 → S17 → VMD2 to VMS1 → S11 → S13 → S17 → VMD2. After the migration, Flow 1 and Flow 2 do not share the same link anymore and the latency immediately decreases from 13ms to 0.7ms. The latency of Flow 2 during the experiment can be checked at Figure 1(d).

3 CONCLUSION

The results show that the mechanism proposed by RDNA Balance is able to migrate routes with low data loss rate, without compromising the communication between servers. Besides, the results show the mechanism offers flexibility for path selection, since the migration is simple and manageable. Future work can fill the gaps in the congestion detection, flow characterization, and queue overflow detection.
REFERENCES

[1] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. 2010. Understanding Data Center Traffic Characteristics. SIGCOMM Comput. Commun. Rev. 40, 1 (Jan. 2010), 92–99. https://doi.org/10.1145/1672308.1672325

[2] Raffaele Bolla and Roberto Bruschi. 2006. RFC 2544 performance evaluation and internal measurements for a Linux based open router. In High Performance Switching and Routing, 2006 Workshop on. IEEE, 6–pp.

[3] Xin Jin, Nathan Farrington, and Jennifer Rexford. 2016. Your Data Center Switch is Trying Too Hard. In Proceedings of the Symposium on SDN Research (SOSR ’16). ACM, New York, NY, USA, Article 12, 6 pages. https://doi.org/10.1145/2890955.2890967

[4] Sangeetha Abdu Jyothi, Mo Dong, and P Brighten Godfrey. 2015. Towards a Flexible Data Center Fabric with Source Routing. (2015).

[5] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. 2009. The nature of data center traffic. In Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference - IMC ’09. ACM Press, New York, New York, USA, 202.

[6] Alextian Liberato, Magnos Martinello, Roberta L. Gomes, Arash Bel-dachi, Emilio Salas, Rodolfo Villaca, Moises R. N. Ribeiro, George Kanellos, Reza Nejabati, Alexander Gorodnik, and Dimitra Simeonidou. 2018. RDNA: Residue-Defined Networking Architecture Enabling Ultra-Reliable Low-Latency Datacenters. IEEE Transactions on Network and Service Management 4537, c (2018), 1–1. https://doi.org/10.1109/TNSM.2018.2876845

[7] Magnos Martinello, M.R.N. Ribeiro, Rafael Emerick Z. de Oliveira, and Romulo Vitoi. 2014. KeyFlow: A Prototype for Evolving SDN Toward Core Network Fabrics. Network, IEEE 28 (03 2014), 12–19. https://doi.org/10.1109/MNET.2014.6786608

[8] Chip Popoviciu, A Hamza, G Van de Velde, and D Dugatkin. 2008. IPv6 benchmarking methodology for network interconnect devices. Technical Report.