ARL-TR-8373 ● JUNE 2018

US Army Research Laboratory

OpenFlow-Enabled Network Functions to Enhance High-Performance Computing (HPC) Platforms

by Venkat Dasari and Nikolai Snow

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer’s or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-8373 ● JUNE 2018

US Army Research Laboratory

OpenFlow-Enabled Network Functions to Enhance High-Performance Computing (HPC) Platforms

by Venkat Dasari
Computational and Information Sciences Directorate, ARL

Nikolai Snow
Envistacom, Atlanta, GA

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
June 2018	Technical Report	May 2016–May 2018

4. TITLE AND SUBTITLE
OpenFlow-Enabled Network Functions to Enhance High-Performance Computing (HPC) Platforms

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Venkat Dasari and Nikolai Snow

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
ATTN: RDRL-CIH-S
Aberdeen Proving Ground, MD 21005

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-8373

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Programmable network functions are described to enhance the computational efficiency of the high-performance computing (HPC) systems by making the network react and adapt to the traffic conditions in relation to the computational efficiency. In this report a micro HPC system with programmable network fabrics is described and its performance compared under various traffic conditions. Matrix multiplication is used to analyze the computational efficiency of the micro HPC system.

15. SUBJECT TERMS
OpenFlow, matrix multiplication, computational complexity, controller, slicing

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Venkat Dasari
Unclassified	Unclassified	Unclassified	UU	16	19b. TELEPHONE NUMBER (Include area code) (410) 278-2846

Contents

List of Figures

1. Introduction

The urgent need to break from stagnation in network innovation and to bring network awareness into computational framework has motivated researchers to consider new extensible protocols to ignite network innovation and to enhance the computing efficiency of the platforms attached to the network. OpenFlow has emerged as the protocol of choice to develop programmable network architectures.[1] OpenFlow separates the control plane from a network device and places it on an external centralized device called a controller (Fig. 1). OpenFlow-enabled switches become high-speed packet forwarding devices while their forwarding paths are controlled by centralized OpenFlow controllers.[2] This design allowed engineers to have total control over forwarding paths. This design has also introduced deep programmability of network devices through control plane application programming interface (API).[3–5]

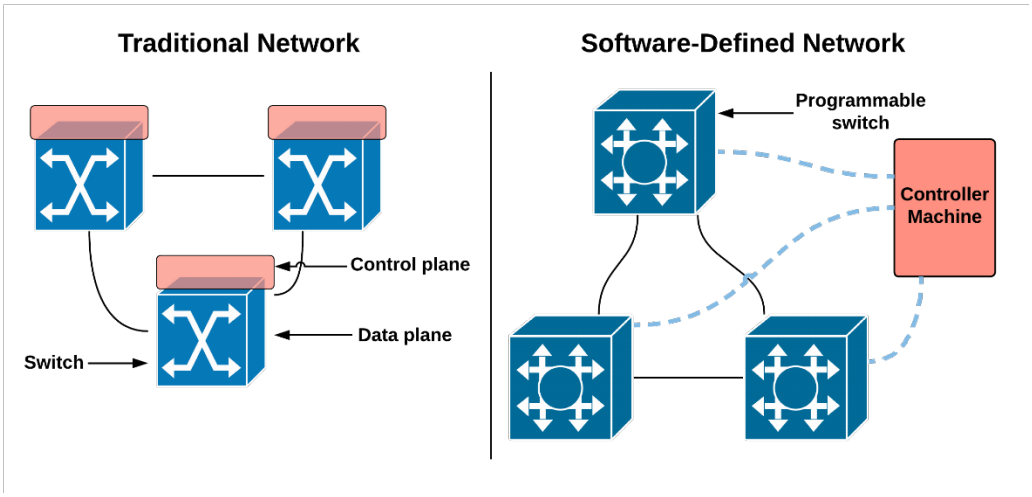

Fig. 1 OpenFlow-enabled software-defined network architecture

OpenFlow-enabled switches use flow tables and group tables to make forwarding decisions to move traffic across the network. There can be more than one flow table in a switch, and traffic lookup will be made until a match is found in any of the flow tables. Unmatched traffic will follow the configured policy or either be dropped or handed up to a controller. Flow tables contain the port information while the group tables contain the action elements, and putting both together will create a policy to be applied to matching traffic.[6] The design is not concerned with how a switch internally processes and integrates with its hardware elements as long as the OpenFlow semantics and action attributes are preserved. OpenFlow implementers

can ignore the difference in the hardware and concentrate on the OpenFlow feature to rapidly build various network topologies completely isolated from each other and run parallel experiments. OpenFlow is widely used and highly programmable protocol to build intelligent networks that can adapt to computing needs.[7] Computing platforms, including HPC systems, depend on the static networks for interconnecting their nodes and clusters. Utilizing OpenFlow in this context can enable more efficient and smarter decision making in the network for job management systems to allow quicker completion for HPC jobs.[8–11]

2. Background

Historically networks are configurable but not programmable. Most of a network's intelligence is hard coded, and its control plane is embedded in hardware along with the data plane. This static design has become an impediment not only to network innovation but also to the performance of distributed computing architectures, including HPC clusters. Currently HPC systems use classical network switches, and no ability exists to change the network functions as demanded by the computing tasks being performed by the cluster.[12] Scientists have studied and demonstrated different ways programmable networks can improve HPC clusters.[13–17]

2.1 Modeling of OpenFlow Enhanced HPC Clusters

In order to remedy the lack of adaptability and to demonstrate the benefits, a programmable switch and controller are introduced in the HPC models discussed (Fig. 2). To accomplish this, a physical cluster is built using embedded boards and a specialty board designed with many Ethernet ports to act as a switch. Typically, enterprise-grade switches can be flashed with different firmware or have compatibility with OpenFlow already; however, in this case, it was more fitting for the micro HPC to use more compact, consumer-grade hardware that more closely matches the embedded boards.

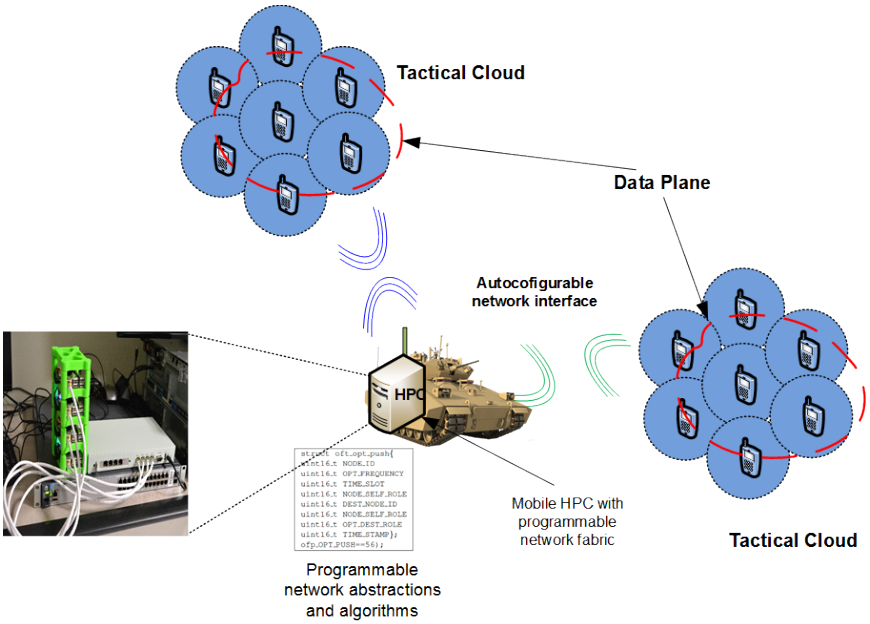

Fig. 2 OpenFlow HPC network

After installing operating systems on all of the respective nodes and the switch, the software framework is introduced. In the switch, Open vSwitch is installed and configured to act as a switch between the physical ports of the board. An OpenFlow controller, Ryu, is also installed to act as the controller of the OpenFlow switch. On the nodes, a message passing interface (MPI) is installed and a network file system is established. With this stack in place, one can run MPI jobs and control the network accordingly to implement the new framework by extending the OpenFlow protocol.

2.2 OpenFlow Extensibility through Experimenter Labels

One key feature of OpenFlow is the ability to insert new functions through experimenter labels. This feature is leveraged in our model to introduce network functions to enhance cluster computing. Experimenter extensions provide a standard way for OpenFlow switches to offer additional functionality within the OpenFlow message type space. Often times, the experimenters are used as a staging area for future OpenFlow features.[18] Experimenter extensions come in a few different object types, such as messages, instructions, actions, queues, meters, or OXM (OpenFlow Extensible matches), which the experimenter utilized in this system. OpenFlow Experimenter matches can be used to enrich a switch's flow match functionality. The modifications necessary to implement experimenter labels require changes on both the OpenFlow switch and controller to maintain compatibility. These OpenFlow experimenter modifications are generally only

made by vendors for their own proprietary features and implementations. This poses a problem for new OpenFlow developers, as the details for how to make these modifications are not made open to the public, apart from the cryptic guidelines that are given in the software repository wiki for how to make such modifications. To introduce the experimenter labels, one must add cases and methods to handle all typical situations that arise normally in a switch, such as comparison, read, or set values. Using additional labels, an evolution of traditional networks is possible; artificial intelligence markup language functions could be created and exist solely on or for network components of HPC systems.[19,20] Leveraging the new capability allows for network developers to create truly smart networks[21] that could complement existing computing, or even function as a computer itself, by incorporating intent-aware networking for computing.

3. Technical Approach

OpenFlow protocol source code was obtained and custom labels were encoded. The modified code was recompiled and installed for both the switch and the controller. The modifications necessary to implement experimenter labels require changes on both sides of a software-defined network: the controller and the switch. To make these changes, one must modify the source code in each to add the required new labels in a way that is compatible with one another. At the time of writing, modifying the controller Ryu (v4.x) is generally easier than modifying the main OpenFlow switch, Open vSwitch (v2.5.x). Modifying the controller can be as simple as adding the new labels to the OpenFlow protocol source file and adding a proper definition for those labels in terms of specifying the type length, experimenter ID, and the data variable(s) associated. The type length is a calculated field based on the experimenter ID and the number of bytes used by the remaining fields. The experimenter ID is a value that is assigned to various organizations who use experimenter labels in OpenFlow. The variable(s) associated with a given field can hold a value that can be matched on or modified using flows. Determining this information proved very difficult, as the details on implementing experimenters in OpenFlow, while it does exist, is incredibly scarce and decentralized. Some of the information was in comments of the code for the switch and the controller, and other parts were not in a large OpenFlow knowledge base but instead scattered among web pages and mailing lists. Despite these major barriers to entry, the goals of this report were accomplished and the labels were implemented. Setting up the switch to read in relevant information into the labels from the cluster was done by writing values to files and reading them across the network. Once the switch received job information from the cluster, OpenFlow is used to send the data up to the controller, where in this micro HPC, single switch case, is the same switch. At

this point, the controller should have the data from every switch in the network, and thus every node in the cluster. The controller can use this global knowledge to modify the network to best suit the jobs being run on the cluster; to this end, a metering system was created. In big data situations, network congestion can cause delays or timeouts for jobs. This job-based metering system, which is designed using the OpenFlow system, has the unique possibility to enable a QoS (quality of service) system that can entirely alleviate the job performance and stability impact that larger data jobs can have on an HPC. To test this system, a parallelized matrix multiplication MPI job was created as a benchmark. This benchmark was run at different core counts and different computation complexities while background traffic was being sent across the cluster to simulate a big data job. The background traffic levels were also modulated along with the other parameters to determine the effectiveness of the metering system.

4. Results and Discussion

The measurements made support the assertion that the job-based metering system introduced in this report effectively improves HPC efficiency. In Fig. 3, we see that when the matrix multiplication is 1000 rows by 1000 columns, the completion time of the multiplication is significantly more for the unmetered runs compared to the metered, while the metering introduces minimal overhead. This same trend can be seen at different traffic levels, computational complexities, and core counts in Fig. 4.

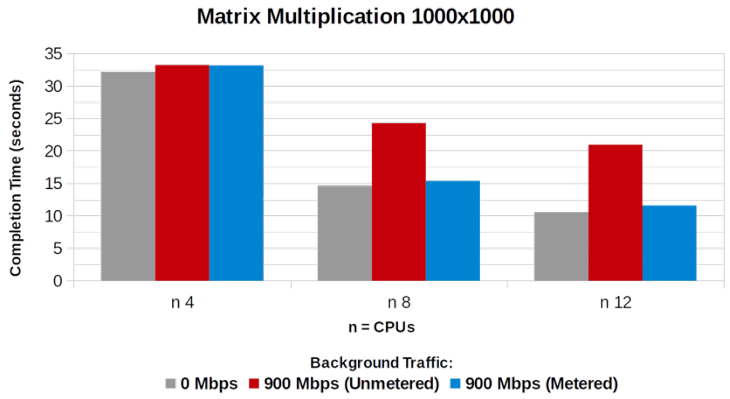

Fig. 3 Computational completion times: metered vs. unmetered

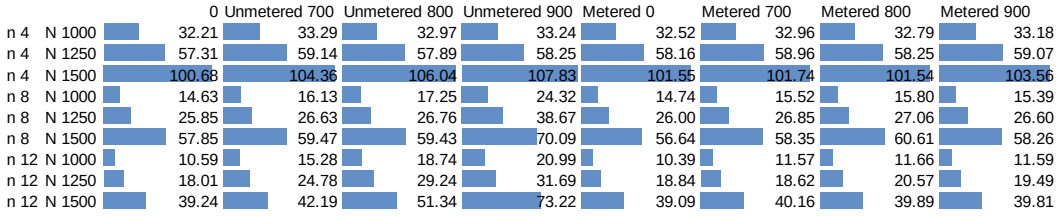

Fig. 4 Computation completion time in various different traffic scenarios

5. Conclusions and Recommendations

We have used OpenFlow tables to successfully demonstrate that smart implementation of network slicing can improve matrix multiplication computation time. This can be extrapolated to a variety of different computation types on a variety of HPC environments. Future research in this area could verify these results using a number of different computations and environments (HPC, remote distributed, etc.). Future research in this area could also be aimed at introducing more sophisticated AI/ML functionality in either job detection, metering level, or network configuration to enable more advanced HPC aiding techniques.

6. References

1. McKeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L, Rexford J, Shenker S, Turner J. OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review. 2008;38:69–74.

2. Suzuki K, Sonoda K, Tomizawa N, Yakuwa Y, Uchida T, Higuchi Y, Tonouchi T, Shimonishi H. A survey on OpenFlow technologies. IEICE Transactions on Communications. 2014;97:375–386.

3. Dasari VR, Humble TS. OpenFlow arbitrated programmable network channels for managing quantum metadata. Journal of Defense Modeling and Simulation: Applications, Methodology, Technology. 2016;13:1–11.

4. Sharma S, Staessens D, Colle D, Pickavet M, Demeester P. Fast failure recovery for in-band OpenFlow networks. 2013 9th International Conference on the Design of Reliable Communication Networks (DRCN); 2013 Mar 4–7; Budapest, Hungary.

5. Monsanto C, Foster N, Harrison R, Walker D. A compiler and run-time system for network programming languages. ACM SIGPLAN Notices. 2012;47(1):217–230.

6. Akyildiz IF, Lee A, Wang P, Luo M, Chou W. A roadmap for traffic engineering in SDN-OpenFlow networks. Comput Netw. 2014;71:1–30.

7. Bianchi G, Bonola M, Capone A, Cascone C. OpenState: programming platform-independent stateful openflow applications inside the switch. ACM SIGCOMM Comput Commun Rev. 2014;44(2):44–51.

8. Watashiba Y, Date S, Abe H, Ichikawa K, Kido Y, Yamanaka H, Kawai E, Shimojo S. Architecture of virtualized computational resource allocation on SDN-enhanced job management system framework. 2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO); 2016 May 30–June 3; Opatija, Croatia.

9. Basnet SR, Chaulagain RS, Pandey S, Shakya S. Distributed high performance computing in OpenStack cloud over SDN infrastructure. 2017 IEEE International Conference on Smart Cloud (SmartCloud); 2017 Nov 3–5; New York, NY.

10. Jamalian S, Rajaei H. Data-intensive HPC tasks scheduling with SDN to enable HPC-as-a-service. 2015 IEEE 8th International Conference on Cloud Computing; 2015 June 27–July 2; New York, NY.

11. Endo A, Jingai R, Date S, Kido Y, Shimojo S. Evaluation of SDN-based conflict avoidance between data staging and inter-process communication. 2017 International Conference on High Performance Computing and Simulation (HPCS); 2017 July 17–21; Genoa, Italy.

12. Benito M, Vallejo E, Beivide R. On the use of commodity ethernet technology in exascale HPC systems. 2015 IEEE 22nd International Conference on High Performance Computing (HiPC); 2015 Dec 16–19; Bangalore, India.

13. Daga A, Powell R, Heath A, Grossman R, Greenway M, Bailey S, Narayan S. OpenFlow enabled hadoop over local and wide area clusters. Proceedings of High Performance Computing, Networking, Storage and Analysis, SC Companion; 2012 Nov 10–16; Salt Lake City, UT. 2012;00:1625–1628.

14. Arap O, Brown G, Himebaugh B, Swany M. Software defined multicasting for MPI collective operation offloading with the NetFPGA. In: Silva F, Dutra I, Costa SV, editors. Proceedings of the Euro-Par 2014 Parallel Processing 20th International Conference; 2014 Aug 25–29; Porto, Portugal. Cham (Switzerland): Springer International Publishing; c2014, p. 632–643.

15. Craig A, Nandy B, Lambadaris I, Koutsakis P. Bloomflow: Openflow extensions for memory efficient, scalable multicast with multi-stage bloom filters. Computer Communications. 2017;110:83–102.

16. Date S, Abe H, Khureltulga D, Takahashi K, Kido Y, Watashiba Y, U-Chupala P, Ichikawa K, Yamanaka H, Kawai E, Shimojo S. An empirical study of SDN-accelerated HPC infrastructure for scientific research. 2015 International Conference on Cloud Computing Research and Innovation (ICCCRI); 2015 Oct 26–27; Singapore, Singapore.

17. Makpaisit P, Ichikawa K, Uthayopas P, Date S, Takahashi K, Khureltulga D. MPI_Reduce algorithm for OpenFlow-enabled network. 2015 15th International Symposium on Communications and Information Technologies (ISCIT); 2015 Oct 7–9; Nara, Japan.

18. Pfaff B, Lantz B, Heller B, Barker C, Beckmann C, Cohn D, Talayco D, Erickson D, McDysan D, Ward D, et al. OpenFlow switch specification. Version 1.3.0 (wire protocol 0x04). Menlo Park (CA): Open Networking Foundation; 2012 June 25 [accessed 2018 May 23]. https://www.opennetworking.org/images/stories/downloads/sdn-resources /onf-specifications/openflow/openflow-spec-v1.3.0.pdf.

19. Huang L, Gaolei L, Wu J, Li L, Li J, Morello R. Software-defined QoS provisioning for fog computing advanced wireless sensor networks. IET Cyber-Physical Systems: Theory & Applications. 2016;1–3. doi:10.1109/ICSENS.2016.7808814.

20. Huang NF, Li CC, Li CH, Chen CC, Chen CH, Hsu IH. Application identification system for SDN QoS based on machine learning and DNS responses. 2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS); 2017 Sep 27–29; Seoul, South Korea.

21. Jmal R, Chaari L. An OpenFlow architecture for managing content-centric-network (OFAM-CCN) based on popularity caching strategy. Computer Standards & Interfaces. 2017;51(C):22–29.

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
2 (PDF)	DIR ARL IMAL HRA RECORDS MGMT RDRL DCL TECH LIB
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
2 (PDF)	ARL RDRL CIH S V DASARI N SNOW