International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

Software Defined Networking: Advanced Software Engineering to Computer Networks

Ankush V. Ajmire[1], Prof. Amit M. Sahu[2]

[1]Student of Master of Engineering (Computer Science and Engineering), G.H. Raisoni college of Engineering and Management Amravati (M.S.), India

[2]Assistant Professor Department of (Computer Science and Engineering), G.H. Raisoni college of Engineering and Management Amravati (M.S.), India

Abstract: ***The paradigm defined networking software is becoming increasingly important and frequently used in the fields of computer networks. It allows us to run the software that manages the entire network. This software becomes more complicated to provide new features that were impossible to imagine before and it requires better performance, better reliability, security and better use of resources will only be possible by advanced software engineering techniques (high availability and distributed systems, optimized Linux kernel, synchronization, validation techniques).***

Keywords: Software Defined Networking, Computer Networks, Openflow controller, Advanced Software Engineering.

1. Introduction

Software-defined networking (SDN) is the "hot" network technology in recent years [2]. It brings many new features and solves many difficult problems of existing networks. The approach proposed by the SDN paradigm is to move the network intelligence from the packet switching devices and put it in the central logic controller. Forwarding decisions are made first in the controller then moves down to the supervised switches that simply running these decisions. This gives us many advantages such as the control and overall viewing the entire network at a time that useful for automating network operations, better use server / network utilization.

A controller (well known as network operating system) is a dedicated server that is running special control software, framework, which interacts with the switching devices and provides an interface for management applications written by the user observe and control the entire network. With same as, the controller is the heart of SDN, and its characteristics determine the performance of the network itself [1].

Author describes the basic architecture of contemporary controller. For each part of a controller author show the software engineering techniques are already being used and could be used in the future to improve performance characteristics. This paper shows the result of our last experimental evaluation of SDN / OpenFlow controllers. On this basis, author explains that the performance of one controller is still not enough to manage the data centers and large scale networks. Finally, this paper presents the approach to high performance and reliable distributed controller for next generation, and discuss possible ways to mention organized and software engineering techniques in high demand.

2. Background

2.1 History

Since the early 2000th many researchers at Stanford University and Berkeley University have begun to rethink the design and architecture of the network and the Internet. The modern Internet and enterprise companies have a very complex architecture and that are built using an old design paradigm. This paradigm includes the demand for a decentralized and autonomous control mechanism that means that each instrument network devices both the forwarding features and the control plane (congestion control, routing algorithms). Furthermore, any additional functionality in modern networking (for example, the traffic engineering, access control, load balancing) is provided by the set of complex protocols and special devices like getaway.

Corporate networks and backbone, infrastructure of data centers, networks for research organizations and educational, home and public networks for both wired and wireless are built on a variety of hardware and proprietary software that are high cost and difficult to manage and maintain. This leads to inefficient use of physical infrastructure, high on cost for security risks, management tasks and other problems.

Enterprise networks are often large, perform a wide variety of protocols and applications, and typically operate under constraints of reliability and high safety; thus, they represent challenging environment for network management. The stakes are high; the company's productivity can be severely hampered by network or burglaries misconfigurations. However, current solutions are low, which makes enterprise networking both costly and error-prone. In fact, most of today's networks are requiring substantial manual configuration by trained operators to achieve even moderate security [2], [4].

The architecture of the Internet is closed for innovations [5]. Reducing the impact of the real world in any given network innovation is because the huge installed base of equipment

and protocols, and reluctance to experiment with production traffic, that created an excessively high barrier to entry for new idea. Today, there are almost no practical way to experiment with new network protocols (for example, new routing protocols, or alternative to IP) in sufficiently realistic environments (for example, the ladder carrying live traffic) to gain the trust necessary for their large-scale deployment. The result is that most of the new ideas of the networking research community are accused and untested.

The design of the modern system often uses virtualization to decouple the system service model of its physical realization. Two common examples are the virtualization disks with logical volumes that the storage interfaces and virtualization of computing resources through the use of virtual machines. The insertion of these abstraction layers allows operator flexibility to achieve the operational objectives divorced from underlying physical infrastructure. Today, the workload can be instantiated dynamically expanded at runtime, moved between physical servers (or geographical locations), and suspended if necessary. Data and computing can be replicated in real time across multiple physical hosts for high availability in a single site or disaster recovery across multiple sites. Unfortunately, computing and storage have successfully leveraged the virtualization paradigm, the network remains largely stuck in the physical world [7], [8], [9]. As clearly stated in [6], networking has become a significant operational bottleneck.

Although the simple task of the routing can be implemented on arbitrary topologies, and the implementations of almost all other network services (for example, routes of politics, ACLs, QoS, isolation areas) are based on the state of topology dependent configuration. The management of this configuration state is cumbersome and error prone adding or replacing equipment, topology changes, move the physical location or handling hardware failures often require significant manual reconfiguration.

Virtualization is no stranger to networks, networking has long supported virtualized primitives such as virtual link (tunnels) and broadcast domains (VLANs). However, these primitive have not significantly changed the business model of networks and operators to continue to configure multiple physical devices to achieve a limited degree of virtualization and automation. So while storage and computing have both been greatly improved by the paradigm of virtualization, networking has yet to break free of the physical infrastructure. In addition, network virtualization functionality implemented via protocols under L2-L4 layers increase the cost and complexity of the network equipment and the difficulty of setting up this type of hardware material.

2.2 SDN

In addition, to solve all the above problems with network management and mentioned configuration, reduce the complexity of network software and hardware, and make them more open to innovation networks the great community of academic and industrial researcher Open Networking Foundation [10] propose a new paradigm for the networking of Software Defined Networking .The approach proposed by SDN paradigm is to separate the control plane (i.e. the policy to the management network traffic) data path plane (i.e. the actual packet transfer mechanisms) (in Figure 1).

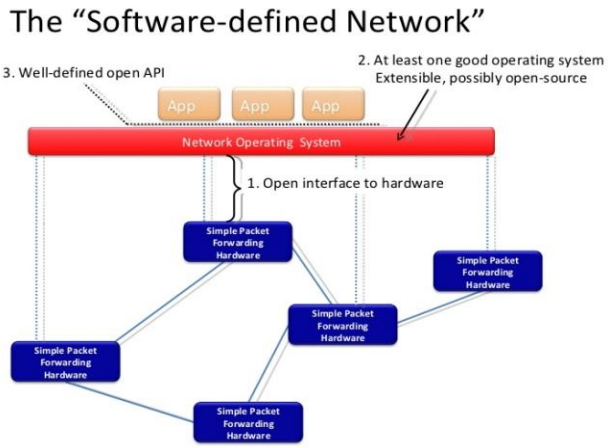

Figure 1: Organization of Software Defined Network.

Traditionally, hardware implementations embodied the logic required for the transmission of packets. That is, the hardware was to capture all the complexity inherent in packet transmission decision. According to the new paradigm [2], [3], [5] all the orders for reference are done first in the software (remote controller), and the hardware simply mimics those decisions for subsequent packets on which that decision applies (for example, all Packets flows networks). Thus, the hardware does not need to understand the packet forwarding logic; it caches only the results of previous transfer decisions (taken by the software) and applies to packets with the same header.

The essential task is to match incoming packets to previous decisions. Packet transmission is treated as a matching process, all packets matching an earlier decision handled by the hardware, and with all non-matching packets handled by the remote control software. It is important to mention that only the packet headers are used in the matching process.

A network switching equipment must now implement a simple set of primitives to manipulate packet headers (to match against the matching rules and modify if necessary) and forward packets [2]. The basic function of such base SDN-switching software is a flow table that stores the matching rules (as packet header patterns must match incoming packet headers) and a set of actions to be applied to the packet matched with success.

Switching hardware must also provide common interface agnostic provider for remote controller. To unify the interface between the remote switching and controller hardware Special OpenFlow protocol [11] was introduced. This protocol provide the controller with a way to discover the OpenFlow-enabled switches, set the matching rules for the switching equipment and to collect statistical switching devices. Figure 2 illustrates an interaction between the controller based OpenFlow and switching hardware based OpenFlow, it controls the switch provides a set of transmission rules.

The paradigm in SDN control functionality is implemented by a dedicated remote host running special control software. At present, there are a number of controllers. The best known are POX [14], Beacon [15], MUL [17], Floodlight [16],

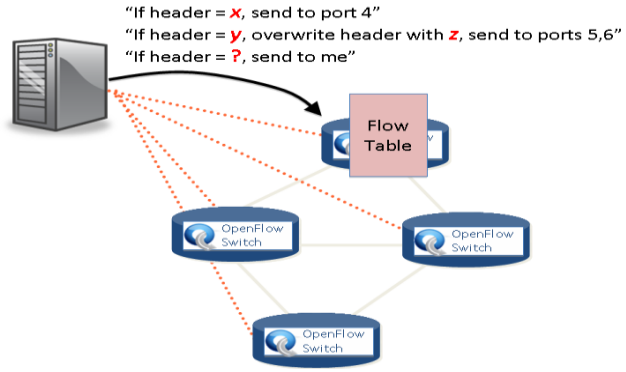

Figure 2: Paradigm of Software Defined Network

NOX [13], Ryu, [20] and Maestro [19]. Again, a controller is a framework that works with OpenFlow-enabled switching devices and provides an interface for management applications written by the user to observe and control the entire network. A controller does not manage the network by then self; it merely provides a programming interface. The Applications implemented on top of the network operating system perform the actual management tasks.

A controller represents two major conceptual departures from the status quo. First, the network operating system provides programs with a centralized programming model; programs are written as if whole network were present on one machine (i.e., routing algorithms use Dijkstra to calculate shortest paths, not Bellman-Ford). Second, the programs are written in terms of high-level abstractions (for example, user and host names), no low-level configuration parameters (for example, MAC and IP addresses). This allows management directives are applied independent of the underlying network topology, but it is requires that the network operating system maintain the fixings carefully (i.e. mappings) between these abstractions and low-level configurations.

2.3 OpenFlow

The OpenFlow protocol is used to manage switching devices: adding new flow, removal of flow, collect statistics. It supports three types of messages as follows:

- **Controller-to-switch** messages are initiated by the controller and used to inspect or manage the state of the switch directly.
- **Asynchronous** messages are initiated by the switch and used to update the controller of network events and the changes in the switching state.
- **Symmetrical** messages are initiated by either the controller or switch and sent unsolicited. All the messages and the detailed specifications of the OpenFlow protocol could be found in [12].

3. Controller

Based on the analysis of materials available about almost twenty four SDN/OpenFlow controllers, paper proposed the reference architecture controller SDN/ OpenFlow shown in Figure 3. The main components are as follows:

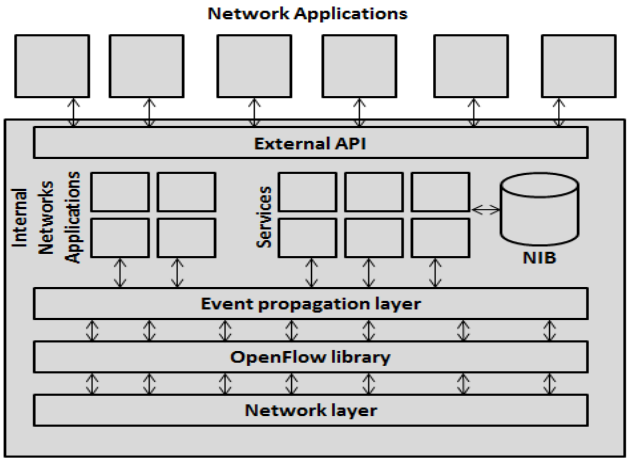

Figure 3: The basic architecture of OpenFlow/SDN controller.

1) **Network layer:** is responsible for the communication with the switching devices. This is the base layer of each controller which determines its performance. There are two main tasks.
- Reading incoming messages from the OpenFlow channel. Usually, this layer is based on the execution time chosen programming language. For faster communication with NIC we can also use the fast packet processing framework as Intel DPDK [22]and netmap [21].
- Inbound OpenFlow messages. The common approach is to use multithreading. A thread listens to the new socket switching connection requests and distributes new connections on other worker threads. A discussion thread communicates with appropriate switches receives flow configuration requests from them and returns the feed configuration rule. There is couple of advanced techniques. For example, Maestro distributes incoming packets using the round-robin algorithm, so this approach is expected to show better results with unbalanced load.

2) **Library OpenFlow:** The main functionality is OpenFlow message processing, verifying the accuracy and according to a type of packet produce new event as "packetin", "portstatus" and etc. The most interesting part here is not in modern controllers still is resistance to malformed message.

3) **Event layer:** The layer is responsible for the propagation of the event between the basic applications, services, and the internal network application. The network application subscribed on the basis of events produces in which other applications can subscribe. This is usually done by publishing / subscribing mechanism, either by writing your own implementation or using the standard as lib-event for C / C ++, RabbitMQ for Erlang..

4) **Services:** This is the most commonly used network functionality as switches discovery, topology creation, routing, and firewall.

5) **Internal network applications:** It is your own application as a learning L2 switch. "Internal" means that it is compiled with the controller in order to get better performances.

6) **External API:** The main idea behind the layer is to provide independent language to communicate with the controller. This common example is based on RESTful Web API.

7) **External network application:** Lever applications in all language services via External API exposed by the internal applications and controller services. These applications are not necessary in a low latency communication and good performance with the controller. The common example is application monitoring.

8) **Web user interface layer:** It provides a user interface based on the Web to manage the controller by setting up different parameters

Therefore, the most important general question before choosing the controller or the creation of a new programming language is to be used. There is a tradeoff between performance and ease of use. For example, POX controller writes about Python is good for rapid prototyping, but it is too slow for the production.

4. Experimental Controller Evaluation

Author conducted an experimental evaluation of the controllers. Authors test bed consisted of two servers connected via a 10 GB link. The first server was used to launch the controllers and the second server has been used for the traffic generation according to a test case.

Paper chose the seven controllers SDN/OpenFlow following:

- NOX [13] is a multi-threaded C++ based controller written above the Boost library.
- POX [14] is a controller based on single-threaded Python. It is widely used for rapid prototyping of network application in research.
- Beacon [15] is a controller based on Java multi-thread which is based on OSGi and spring frameworks.
- Floodlight [16] is a Java-based controller that uses multi-thread Netty framework.
- MUL [17] is a multi-threaded C-based controller writes on top of libevent and glib.
- Maestro [19] is a Java-based controller that uses multi-thread library java.nio.
- Ryu [20] is a Python wrapper regulator that uses gevent of libevent.

Each controller runs the application from L2 switching learning provided by the controller. There are several reasons for this. It is representative and at the same time very simple. It fully utilizes the internal mechanisms of the controller, and it also shows the effectiveness of the programming language is selected by implementing simple hash lookup.

This paper used the latest available sources of all controllers and run all controllers with the recommended settings for the performance and latency test, if available.

As traffic generators, author used freely available cbench [18] and author's hcprobe framework for controllers tests. Cbench and hcrpobe emulates any number of OpenFlow switches and hosts. Cbench is intended to measure various aspects of controller performance, including response time of the minimum and maximum control device, maximum throughput. Hcprobe used to investigate various characteristics of the controllers in a more flexible way by specifying templates to generate OpenFlow messages, the change in the number of reconnection attempts in case the controller closes accidentally the connection, choosing the profile of traffic. It is written in Haskell, which is the high level programming language and allows users to easily create their own scenarios for controller tests.

Authors testing methodology includes measures of performance and scalability, and advanced functional analysis such as security and reliability. The objective measures of performance / scalability is to get maximum throughput (number of outstanding packets, flows/sec) and the minimum latency (response time, ms) for each controller. For reliability, author measured the number of failures in the long term test under a given workload profile. And as for safety, author gives how the controllers work with OpenFlow malformed messages.

Figure 4 mounts the maximum throughput for a different number of available cores by a controller. The single threaded controllers (Pox and Ryu) showed no scalability in processor cores. The performance of multithreaded controller increases constantly in line for 1-6 rings and much slower for 7-12 cores for use with Hyper Threading Technology (for the benefit of the maximum performance of the technology is 40%). Beacon shows the best availability, reaching the speed of 7 billion streams per second. This is because the use of shared queues for incoming messages and prizes for outgoing messages.

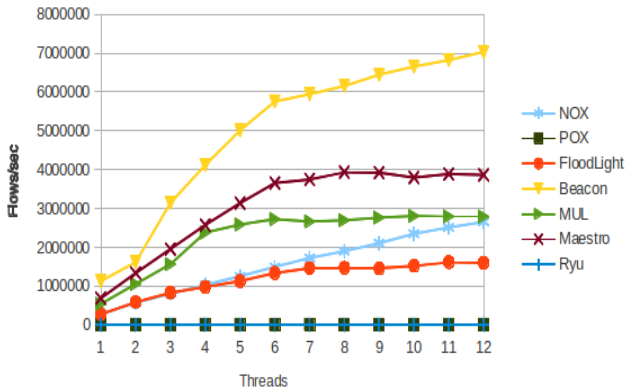

Figure 4: The different number of threads achieves average throughput.

The average response times of all the controllers are in between 80-100ms. Long term tests show that most controllers when running for a long time begin to loos connections with switches and lose packages in messages. The average number of errors is 100 for 24 hours. And almost all accidents controllers or lose the connection with a switch when they received malformed messages.

Returning to the flow values and sufficiently understand if the current performance. In the new flow demand data centers happens in every 10us maximum 300µs and 2ms on average [23]. Assuming small data center with 100K guests and 32 hosts / rack, the maximum rate of flow of arrival can be up to 300M with the rate between 1.5M and 10M. Assuming 2M flow/s rate for a controller, it requires only 1-5 controllers to handle the median charge, but 150 for the peak load. In large scale networks the situation can be extremely worse.

Solving the problem should go two ways. The first way is one controller improves itself by making advanced multi-threaded optimizations. The second way is by using multiple controllers bodies jointly managing the network. This approach is called distributed controller.

5. Moving to Distributed Controller

As seen in the previous section one controller is not enough to manage the entire network. There are two problems here:

- **Scalability:** Because networks are developing rapidly, the controller's resources are not enough to maintain the status of all network devices. In addition, the flow setup latency in a larger network is also increasing.
- **Reliability:** Controller is a single point of failure. If the controller fails, the network system stops.

To solve the above mention problems, we need physically distributed control plan with centralized view of the entire network. The diagram of the solution is shown in Figure 5.

Networks are divided into segments controlled by dedicated example of the control unit. Network segments can overlap to ensure network resilience in the event of failure of any controller. In this case, the switches will be redistributed to the appropriate bodies of the controller. Each controller is connected to a distributed data store that provides a consistent view of the entire network. It stores all the switching and application specific information. Application state is stored in the data store distributed to facilitate migration of the switch and the controller of the failure recovery.

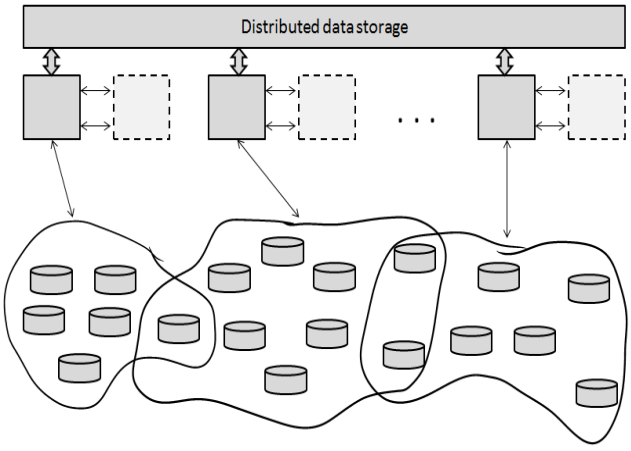

Figure 5: The organization scheme of distributed controller.

In addition, each controller fails over in case of controller failure. It might be cold or hot. The cold failover is disabled by default and will only start when the master controller is crashes. The hot failover receives the same message as the master controller, but read only access. This provides the smallest recovery time.

There are many open research questions such as how to organize the coherence controllers in the right way, how to reduce overhead on the use of distributed data store, and how to make the switch migration, how to run applications on distributed controllers, what is the better controllers placement.

6. Conclusion

Software-defined networking (SDN) has been developed rapidly and is now used by early adopters such as data center. It provides immediate savings in capital costs by replacing owner's routers with switches and controllers raw materials, abstractions of computer science in network management offering operational cost savings with functionality and performance improvements too. However there is much research to be done especially in the area of SDN software. Controllers are not yet ready for use in production because of poor performance to work with data center and large scale network load. Distributed controller is the next step in the development of SDN/OpenFlow controllers. It is to solve the reliability and scalability problems of controllers.

References

[1] Ruslan Smeliansky, Alexander Shalimov,"Applied Research Center for Computer Networks," Moscow State University,Moscow, 2013.
[2] T. Koponen, M. Casado, S. Shenker, D. Moon, "Rethinking Packet Forwarding," Hardware. In Proc.of HotNets, Nov. 2008.
[3] Scott Shenker, Michael J. Freedman, Justin Pettit, Jianying Luo, Natasha Gude, Nick McKeown, Martin Casado,"Rethinking enterprise network control," IEEE/ACM Transactions on Networking (TON), v.17 n.4, p.1270-1283, August 2009.
[4] Michael J. Freedman, Jianying Luo, Nick McKeown, Justin Pettit, Scott Shenker, Martin Casado,"Ethane: Taking Control of the Enterprise," Kyoto, Japan, August 2007.
[5] Tom Anderson, Hari Balakrishnan, Nick McKeown, Guru Parulkar, Jennifer Rexford , Scott Shenker, Jonathan Turner, Larry Peterson, "OpenFlow: enabling innovation campus networks," Computer Communication Review, v.38 n.2, April 2008.
[6] J. Hamilton, "Data work center networks are my way," at Stanford Clean Slate Summit, 2009.

[7] R. Ramanthan,T. Koponen, S. Shenker S. , M. Casado,"Virtualizing the Network Forwarding Plane," In Proc. PRESTO November 2010.
[8] J. Pettit, T. Koponen, M. Casado, S. Shenker, B. Pfaff,"Extending Networking into Virtualization Layer," HotNets-VIII, Oct. 22-23, 2009.
[9] J. Gross, B. Pfaff, M. Casado, S. Crosby, J. Pettit, "Virtual Switching in Era of Advanced Edges," 2nd Workshop on Data Center Converged and Virtual Ethernet Switching , 22, Sep. 6, 2010 .
[10] "Documentation: Open Networking Foundation," https://www.opennetworking.org.
[11] "Openflow," http://www.openflow.org.
[12] "Openflow documentation and specification," http://www.openflow.org/wp/documents.
[13] Koponen, T., Pfaff, B., Pettit, J., Casado, M., McKeown, N., Shenker, Gude, N.,"NOX: towards an operating system for networks," Computer Communication Review 38, 3 (2008), 105-110.
[14] "Pox specification and documentation," http://www.noxrepo.org/pox/about-pox/.
[15] "Beacon specification and documentation," https://openflow.stanford.edu/display/Beacon/Home.
[16] "Floodlight documentation on ," http://floodlight.openflowhub.org/
[17] "Mul specification and documentation," http://sourceforge.net/p/mul/wiki/Home/
[18] "Cbench specification and documentation," http://www.openflow.org/wk/index.php/Oflops
[19] Zheng Cai, "Maestro: Achieving Coordination and Scalability in Centralized Network Control Plane," Thesis, Rice University, 2011.
[20] "Ryu documentation on," http://osrg.github.com/ryu/
[21] Luigi Rizzo, "netmap: anovel framework for fast packet I/O," Usenix'12, June 2012.
[22] "Packet Processing and Enhanced with Software from Intel DPDK," http://intel.com/go/dpdk.
[23] Aditya Akella, David A. Maltz, Theophilus Benson, "Network traffic characteristics and data centers in the wild," IMC, 2010.