Expedition: An Open Source Network Monitoring Tool for Software Defined Networks

Capstone Research Project

April 24, 2015

Ashwin Joshi Darshan Maiya Gaurav Chheda Vamsikrishna Nethi
Interdisciplinary Telecommunications Program
University of Colorado Boulder

Advisor : Mark Dehus

Abstract— **Network virtualization is likely to be the future of the telecommunications industry. Software Defined Networking (SDN) is a key technology in the network virtualization space. In SDN, the control plane of networking devices is handled by a central entity called the controller. The following paper proposes development of a universal fault and performance management tool '*Expedition*' to monitor SDN controllers and topologies managed by these controllers. The research focuses on three main objectives. First, identifying different methods to extract and store the data from the selected SDN controllers. Second, creating a centralized custom layer to interpret the raw data and formulate Key Performance Indicators (KPIs) based on it. Third, reporting the data in forms easily understood by the end user. To achieve these objectives, we designed an abstraction layer to monitor selected controllers and had KPIs reported and visualized appropriately on Splunk. In this paper we discuss the Proof of Concept (POC) developed. In this POC, we monitor KPIs such as throughput, latency, availability for the four selected SDN controllers, viz. Open Daylight, Floodlight, Ryu and Pox. *Expedition* allows for proactive service assurance, which is an important function of the telecommunication Operation Support System (OSS) stack. This proactive monitoring will be crucial for providers to keep Service Level Agreements (SLA) intact with their customers. This will help the organization increase its profitability, as well as gain credibility in the industry.**

I. Introduction

Today, network operators have to use a set of low level commands to configure and manage the network from the command line interface (CLI) [1]. Software Defined Networking (SDN) is a technology in which the control plane and the data plane are separated, and there is a centralized controller which can be programmed to dictate the way the network should behave. Network virtualization allows network operators exercise greater control over their network and also provides enhanced management capabilities providing better flexibility and agility in managing networks. The enhanced management capabilities make network virtualization the future of the telecommunications industry. Software Defined Networking and Network Function Virtualization (NFV) are two of the key network virtualization concepts [2].

In a Software Defined Network, the controller is a single point of contact for control information, and hence has the potential to be a single point of failure for the functioning of the network [3]. Thus, it is critical to monitor the performance of the controller in addition to the performance of the network. The controllers talk with the underlying infrastructure and expose northbound Application Programming Interfaces (APIs) to program its network using a set of web services. Northbound APIs act as an abstraction interface to the application as well as the management systems that lie on top of the SDN stack. Along with controller health, the underlying network must also be monitored to maintain Service Level Agreements (SLAs). Performance and Fault Monitoring are the two major areas from the Fault Monitoring, Configuration, Accounting Performance and Security (FCAPS) model of network management, and were the focus of this research.

Traditional network management techniques such as Simple Network Management Protocol (SNMP), Netconf, RMON, etc. have limited functionality with software-defined networks. Thus, we require a different approach for efficient network management in case of SDN. Automation is a major challenge in managing SDN networks. The applications on top of SDN will constantly send requests to the controller to access controller as well as network information. Understanding the impact of these requests asking for resources from an SDN network impacting the controller performance. Thus, consideration of these impacts is vital when automating network management practices. In SDN networks, it becomes necessary to replicate traditional functions of capacity planning, monitoring, troubleshooting, security, and other critical management capabilities. Hence, automation of network management is vital for SDN to proliferate and succeed.

The key aspect in managing multi-services SDN networks is autonomy in management. Attributes such as network awareness, configuration, protection from faults, identification of errors, and optimization of network resources will require autonomy in deployment for efficient network management of SDN networks. The introduction of these aspects into SDN will decrease the complexity of network control and management to a greater extent, and reduce the probability of errors that occur with manual operation. Further, the job of the network engineer and planner will become simplified, allowing them to utilize the advantages of SDN without having to program the controllers themselves.

i. Statement of the Problem

Although Software-Defined Networking is at a relatively early stage of adoption, major networking vendors are increasingly starting to define their strategy around this technology. While there has been a hesitancy to completely migrate networks to an SDN-based model, networking operators, vendors and providers are all increasingly adopting SDN methodologies on their networks [17]. Considering its utility and ease of use, SDN has the potential to be one of the most disruptive developments in the networking industry. Further, this innovation has been driven by the need for data-center network virtualization, and the migration of service providers and enterprise networks to an SDN-based model. As this expected adoption gains momentum, it is important that service providers have a *single tool that can monitor and interface with multiple SDN controllers.*

It is also important for SDN code developers to be able to measure running time of their code. That will help them evaluate the efficiency of the code being run, and hence, its effectiveness as a practically deployed SDN solution. Along with execution time of code, scalability versus response time behavior has also to be measured so as to accurately predict optimum network performance metrics.

In the current setting, there are several licensed and open source tools for monitoring faults and performance in traditional networks [12]. However, in case of SDN networks, there is no single universal tool capable of monitoring the network irrespective of the SDN controller deployed. Currently, the latent SDN-based service provider industry faces the challenge of integrating physical and software defined network monitoring tools to avoid network management silos. This research aims to provide a solution to this challenge.

The solution proposed through this project is '*Expedition'* - a tool for monitoring SDN controllers, creating intelligent reports and logging errors based on the data stored in the controller, as well as the performance of the controller itself. The tool monitors a defined set of Key Performance Indicators (KPIs) for the network based on the thresholds associated with them. Fault management is integrated with performance management so that faults can be generated in case of threshold violations, and users can take corrective actions specific to each KPI. In addition to this integrated functionality, fault management and performance management features directly monitor the controller to detect any abnormalities in network performance.

ii. Research Question and Sub-problems

This capstone project aims to obtain the answer to the following research question:

"Is it possible to develop a unified fault and performance monitoring tool that can interface with multiple SDN controllers?"

As a solution for the above research question, we have developed a software based abstraction layer which is a universal monitoring platform for four controllers as far as this POC is concerned. The data collected by this layer is visualized in a systematic way for our end users.

The above research question was divided into three main sub-problems:

1. **How to collect the data** - This sub-problem has the following aspects:

A. **Universal Monitoring Tool**: There are many operators in the SDN controllers industry today, such as Cisco, Brocade, OpenDaylight Project, Project FloodLight, Juniper, Big Switch Networks, Ryu, NOX/POX, etc. Almost all of them have their own network monitoring tools, the drawback being that these tools work only with their respective controllers. To develop a universal system/tool capable of working across the different vendor controllers, there is a need to identify the conversion metrics for these controllers. The suggested solution to this problem is a common translation layer above the monitoring engines of each SDN controller, developed through this project. The translation layer can handle input from the various controllers and transform this information into a fixed output format that the monitoring engine can understand. Further, it is important to collect data to evaluate the performance of the controller as explained previously.

B. **Data Exchange technology**: An abstraction layer between the network and our software is required. This layer consists of API calls and calls to web services for fetching data from the extraction translation load (ETL) engine. Along with this, various data exchange formats have been evaluated and chosen. In some controllers, such an API was already present however some controllers need construction of custom web services, allowing for changes required to exchange the specific data that is required for this project. We have chosen JavaScript Object Notation (JSON) data exchange format as it is light, simple to understand, widely used and is application independent [13]. The abstraction layer further depends upon the type of application - a web application or a standalone application. Through the implementation of this project, the application has been hosted on a web server, and is available to be run as a web application.

2. **How to interpret the data** - This sub-problem has the following aspects:

   A. **Backend design for storing raw data from SDN controllers**: The data extracted from the data exchange layer needed to be stored in persistent data storage structures, so that the data could be readily accessed by the application.

   B. **Conditioning and correlating raw data**: Conditioning of data had to be done as part of performance monitoring. This includes calculating KPIs from the available data. Violation of KPIs raises a fault in the fault management system. Upon evaluating the data available and the KPIs that could be represented, the KPIs identified and implemented through this project include interface bandwidth utilization, errors on ports, controller availability, controller throughput and latency.

3. **How to report the data** - This sub-problem consists of the following:

A. **Reporting**: The main use of *Expedition* tool is to generate various fault logs so that network analysts know exactly what part of the network is malfunctioning. In addition, the KPIs mapped are represented graphically so they are accessible to the user. Splunk is chosen as the visualization tool for this project for the same. The graphs generated by Splunk have been included in the results section below. Further, reporting is customized so that users at different organizational levels have specific access to the tool.

II. Literature Review:

In the SDN architecture, the data plane is separated from the control and management planes. Network devices are simple forwarding elements, which depend on the controller to decide the forwarding rules for each flow in the network. Flows installed on the controller could include rules to forward packets out of specific interfaces, drop packets, send to the controller for handling, etc. Controllers bind northbound and southbound systems in the network management infrastructure. Northbound flows talk to the higher application layer, while southbound flows talk to the network elements themselves. These terms are used to describe the APIs in the SDN architecture [15]. These APIs on the controllers are used to perform all the communication between the network devices and the applications that interact with the controller. In SDN, Openflow is the most widely used southbound interface. This protocol is used for communication between the centralized controller and the end network devices (both virtual and physical) so that the controller can discover network topology, define network flows and implement requests relayed via northbound APIs [15]. The northbound APIs are used for communication between the controller and applications or higher layer control programs. Several open source SDN controllers use the REST API as a northbound API.

Greenberg and Hjalmtysson propose four dimensions of network management & control. These are decision, dissemination, discovery & data. The same fundamental approach to look at the network management has been used in our project [4].

Kuklinkski and Chemoui in their paper have analyzed the network management capabilities of the Open Flow protocol and have identified the main challenges for SDN management. They have analyzed the current issues with SDN and have suggested the solution based on different research directions that can be implemented to enhance the network management capabilities of the SDN [5].

Gelberger et al. in their paper discuss performance of SDN for various workloads. They also account for other factors in addition to SDN that account for performance penalty [6].

Kempf et al. discuss fault management for detecting the link node failures for southbound systems. The paper discusses the fast recovery algorithm configured on switches. Controllers can use Link Layer Discovery Protocol (LLDP) messages to discover link and node failures. This paper proposes to develop a tool to monitor the performance and faults that may occur in the SDN enabled network, which has been the focus of this paper [7].

Slater describes various parameters which can be monitored in a SDN network. The paper also predicts the need for monitoring tools as the technology in moving towards SDN. This research aims to achieve the same by exposing controllers APIs and develop a global adapter for SDN Controllers [8].

Wangdong et al. in their paper have proposed an automatic QoS mechanism. The mechanism they have proposed, combines the advantages of the autonomic network management and the SDN technologies. This novel QoS model is also presented based on the AQSDN (Autonomous QoS Managed SDN) architecture. At the end, the self-configuration feature of AQSDN and the enhancement of video quality of the PCaQoS (Packet Context Aware QoS) model are verified [9].

Li et al. in their paper have given a brief architecture of the way a network management tool for SDN can be built up, using NOX as their controller. Aspects of this architecture have been utilized in this paper to deploy the network management tool for the various SDN controllers [10].

Jain et al. in their paper describe Google's practically deployed B4 private WAN network architecture that connects Google's datacenters. The paper further describes the design challenges faced in deploying the same, and some of the considerations that were made for the same. Further, the paper describes how utilizing SDN techniques for the same allowed for a near-hundred per cent bandwidth utilization and reduced the overhead in resources caused by over-provisioning. The section on 'Impact of Failures' provides scope on some of the problems that have been addressed through this research project [11].

Tootoonchian et al. in their paper titled “On Controller Performance in Software-Defined Networks” describe the flow-based benchmarking tool, CBench to extract performance of the NOX, NOX-MT, Beacon and Maestro SDN controllers. Through this project, the CBench tool has been used to monitor and integrate performance KPIs for the OpenDaylight, Floodlight, Ryu and Pox controllers [16], a critical part of ensuring performance of SDN networks.

III. Research Methodology:

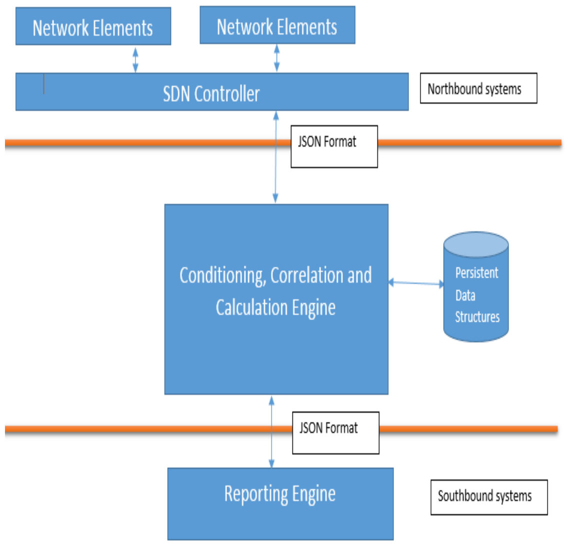

Figure 1: Architecture for the *Expedition* tool

As a proof of concept, the OpenDaylight, FloodLight Ryu and Pox SDN controllers were chosen. These controllers have varied architectures and showing the functionality of the tool with all four architectures validates the tool’s working. The information required to monitor the network accurately is extracted from the controllers as required. The decision on how frequently this information is pulled is decided based on the requirement and importance of the parameter. Further, this decision is based on the frequency and performance of Splunk to visualize the data that is being extracted. In addition, to ensure proactive performance management, the tool takes actions to log faults in case of violations in set thresholds, without involvement of the user. For example, in case the bandwidth utilization on a particular link increases beyond the threshold, the tool creates a log entry detailing the violation of the KPI on that link.

The common network topology used is created using the Mininet network emulator. The sample topology that has been utilized in the execution of this project is composed of two switches, each of which has four hosts connected to it, with the two switches connected to each to each other. In addition, each switch in the network has a socket connection to the SDN controller, which allows for the controller to set flow rules on the switches, via the southbound openflow controller.

For the implementation of the proof of concept, Python is chosen as the programming language. Since Expedition is a universal monitoring tool, it must support all leading SDN controllers. For this purpose, an Extraction and Translation Layer (ETL) is present at the downstream part of the architecture. This layer is responsible to query controllers and network devices to extract the raw data. These raw data is stored in persistent data storage structures, in order to be able to access the data as required. These data is also used to compare network and controller performance under different conditions. This will help network administrators to take preventive actions, and give them time to augment the network capacity to handle increased loads in future.

To obtain performance of the network, the controller queries the network switches to return counter values that are stored in it. The network KPIs that are being monitored include interface errors and interface utilization. The tool converts the raw interface statistics obtained from the switches into a form that can be presented as the required KPIs.

To obtain performance of the controller, the CBench performance benchmarking and evaluation tool is used. While the tool was primarily created for the Floodlight controller, it can be used to work with other open source controllers as well. Tests are run on the tool to simulate the controller with varied number of switches, as well as the number of tests and durations at which CBench is run. The KPIs that are extracted from the controller include the latency and throughput of the controller. This is done by running tests to install flows and testing the average throughput and latency for a specific number of tests and delay in forwarding traffic.

Further, an additional KPI being monitored is the controller availability. This is monitored by evaluating the periods for which the controller remains available to the tool to being monitored. This feature is implemented as a heartbeat monitoring system, which probes each of the controllers periodically to ensure they are reachable from the tool.

To test for performance of an SDN developer’s controller code being developed, appropriate breakpoints are placed on the code, and the time between the breakpoints is measured. This helps developers compare the performance of their code on multiple SDN controllers, to find one that suits their need. Further, such an analysis helps them understand the impact of scale on their code, i.e, the running time of their code with increasing size of the topology. This methodology was followed on previously developed code that calculates shortest path between hosts using Dijkstra’s algorithm for the Pox controller. The code was developed for topologies ranging from four to twenty switches, to compute performance. The results for the same are attached in the research results section of the paper.

In order to relay these data to the correlation engine, JSONs (lightweight APIs used for transmitting data) will be created which can be understood by the monitoring engines. These JSONs have to be similar irrespective of controller and network size being monitored for the data. The correlation engine parses through the JSONs to extract and stores the data in appropriate persistent data structures. As mentioned above, the focus of this project is on configuring SDN based networks, developing a tool to extract the status of the networks and the controllers through APIs and unifying the data by storing in a data structure. Pox, Ryu, Floodlight and OpenDaylight support the northbound Representational State Transfer (REST) API. The next step in the research methodology is an abstraction interface which will extract, transform and load the data.

The backend is very important for the software solution developed. Data extracted is stored in appropriate persistent data structures, which can then be accessed by the data conditioning engine. As the tool scales up to store large amounts of data from larger topologies, database selection plays a crucial part.

The next step in the architecture is the data conditioning engine that can translate the obtained raw data into KPIs. As described above, the KPIs monitored are interface errors, bandwidth utilization, controller latency, controller throughput and controller uptime. Thresholds are set on these values for proactive performance management. As proposed, the fault management module of the solution integrates directly with the controller as well as with the performance management system.

The final aspect of Expedition is reporting the data obtained. The tool would be incomplete if it cannot convey the results clearly and elaborately to its users. Also, it is essential that a user gets the information in a comprehensible form, and hence, data is represented with this in mind. The existing visualization tool, Splunk, has been used primarily for this purpose. Various graphs and charts are generated for the user to access the information accurately. These are attached in the results section of the paper. In addition, Splunk allows users to obtain the research data and graphs in various file types, such as CSV, PDF, etc. Another feature of the Expedition tool is giving users customized access, depending on their organizational level. To demonstrate this, a specific user is created such that the user can only view certain aspects of the tool, and without administrator access to the tool.

Lastly, detailed documentation is created as individual modules are completed. This ensures that there is a clear record of our progress through the course of the research, and for users to understand the functioning of the tool, for future development. As the project has primarily been developed in Python, Sphinx has been used as the preferred tool for documentation, because of the uniformity it offers.

IV. Research Results

After research on choosing the most appropriate SDN controllers on which to test for the validity of the proof-of concept developed, OpenDaylight, Floodlight, Ryu and Pox were chosen. This was based on factors such as programming language, utility, industry adoption and variety of other choices. Developing the tool for the above controllers has given detailed results that validate the working of these controllers.

The results for the time convergence of the Dijkstra's algorithm calculated for the Pox controller have been shown in figure 2 below. The graph plotted has time on the Y-axis and increasing number of switches in the looped topology on the X-axis. The number of switches for which the topology has been created has been incremented from four through to twenty, in steps of two. Additional switches have been added on each branch of the topology described in figure 3.

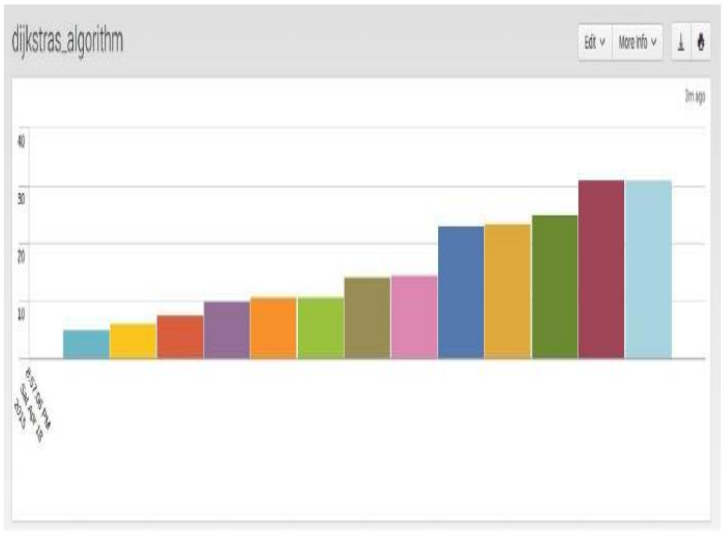

Figure 2: Time convergence of Dijkstra's algorithm

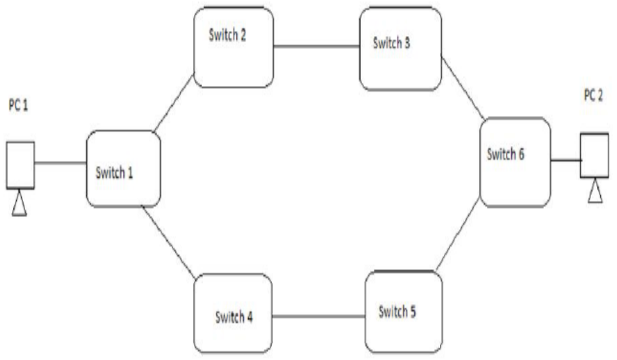

Figure 3: Topology used for Dijkstra's algorithm calculation

The universal abstraction layer outputs the results of KPIs into persistent storage. This store is then configured as a data source for the visualization tool. The important feature of the Splunk visualization tool is that it allows for real time data visualization. Further, we have made use of the Splunk feature to save this dashboard and assign it to certain users for access based dashboard views. As stated previously, we have monitored interface utilization statistics of the 4 selected controllers. This is presented on the dashboard. A change in topology statistics such as traffic leads to a real-time change in the dashboard, thus, making it easy for network administrators to check the performance of the network.

Figures 6.1 and 6.2 in pages 7 and 8 of this document demonstrate the outputs of interface utilization statistics, as obtained for the four controllers on the *Expedition* tool, mapped with interface utilization on the Y-axis and time on the X-axis. Each time instance gives the utilization statistics for each interface of the network. These graphs are real-time and changes according to fluctuations in traffic.

The topology chosen for the measurements of interface utilization and interface errors is shown in the figure below:

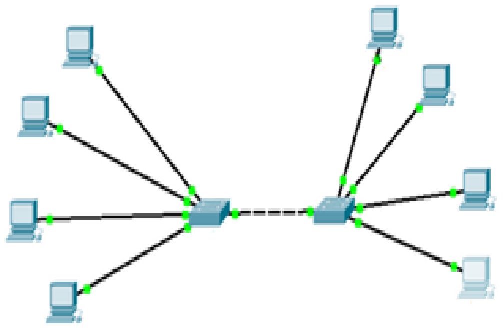

Figure 4: Topology used for interface utilization calculation

The throughput and latency for the Ryu Controller is displayed below. The graph maps the throughput and latency, in units of flows per second, on the Y-axis, to time, on the X-axis:

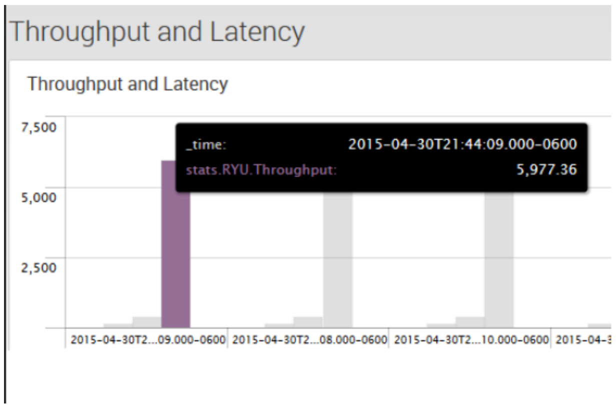

Figure 5: Throughput and Latency of Ryu controller

V. Discussion of Results

The results of the analysis of Dijkstra's algorithm in Figure 2 helps the SDN developer measure how the SDN code scales up with increasing size of topology. As per the graph results obtained, it is clear that the convergence time growth of the algorithm is mostly linear. Such an observation would help developers understand the impact of their code on a real-world network. It must be noted, however, that this calculation was made for a specific sparse topology, with switches incremented along the two branches of the topology in Figure 3. With a different topology, the results obtained may be different, and can be easily calculated in the same manner as above. A further caveat is that this calculation was done statically, with topology changes being done after the controller was taken down. This is one of the drawbacks of using the Mininet network simulator, as it does not allow for real-time changes in the network topology, as would happen in a real-world scenario.

The fault logging feature, as well as KPI for availability is presented in the form of log entries in Splunk. Through this research, a comparison of the four SDN controllers shows that OpenDaylight and Floodlight require a higher CPU processing power to run and compute, as compared to the Pox and Ryu controllers. This is further evidenced by the fact that CBench can be run on a single core virtual machine for Pox and Ryu, whereas multi-core environments are required to compute the same for OpenDaylight and Floodlight.

The computation of throughput and latency for each of the SDN controllers has been done using the CBench tool. The CBench tool installs flows for a customized number of switches and computes the throughput and latency of the controller, in units of flows per second. The graph for the two parameters for the Ryu controller has been displayed in figure 5, and gives a clear indication regarding its health. If there are abnormal spikes in that graph, it indicates that the controller is being over-utilized, and registers an alert to the user.

The comparison of interface utilization, both received and transmitted bytes, as done in Figure 6, and can be accessed in real time through the tool, help network engineers get a real time view of the network, and how it is performing. The dashboard clearly depicts per interface utilization of each of the SDN controllers, with the data being pulled from the controllers at intervals of fifteen seconds. The unified dashboard depicts the utility of the *Expedition* tool, and demonstrates that the research question has been answered positively.

During the course of this research, we have observed that a comparison of the four SDN controllers shows that OpenDaylight and Floodlight require a higher CPU processing power to run and compute, as compared to the Pox and Ryu controllers. This is evident from the running time for each of these controllers in performing equivalent tasks. This is further evidenced by the fact that CBench can be run on a single core virtual machine for Pox and Ryu, whereas require multi-core environments to compute the same for OpenDaylight and Floodlight.

A close observation of results obtained shows that the Pox and OpenDaylight controllers display an anomaly, in that there is a very slight increase in interface utilization statistics even though no traffic is being generated at that point of time. In addition, it allows network engineers to monitor multiple SDN controllers simultaneously, and compare the performance of one controller with another. Such a dynamic, real-time monitoring of the network is vital for network engineers to be able to execute their tasks correctly. Further, it clearly validates the working of the *Expedition* tool, and provides a positive answer to the research question upon which this paper was based.

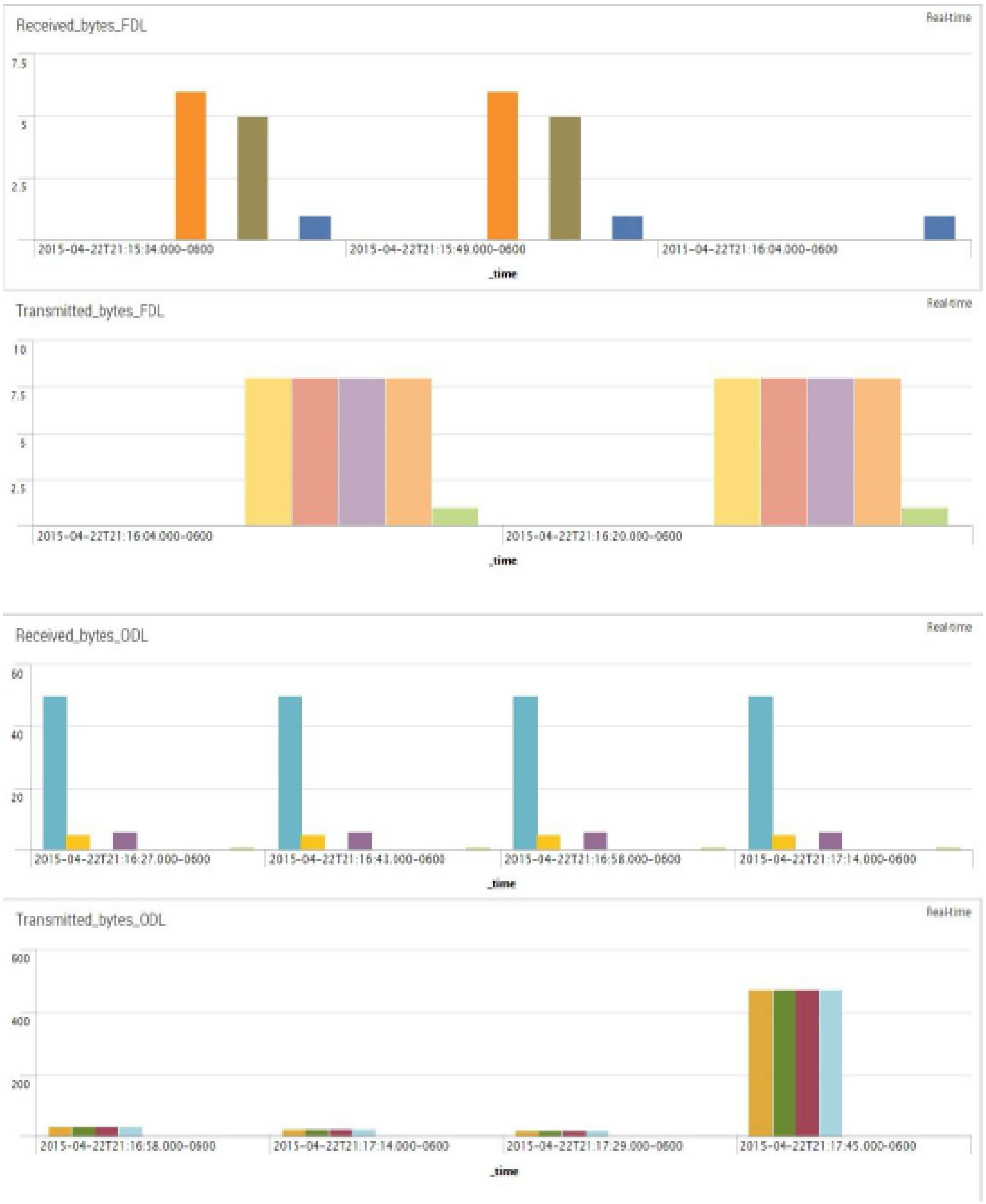

Figure 6.1: Interface utilization dashboard for Floodlight and OpenDaylight controllers

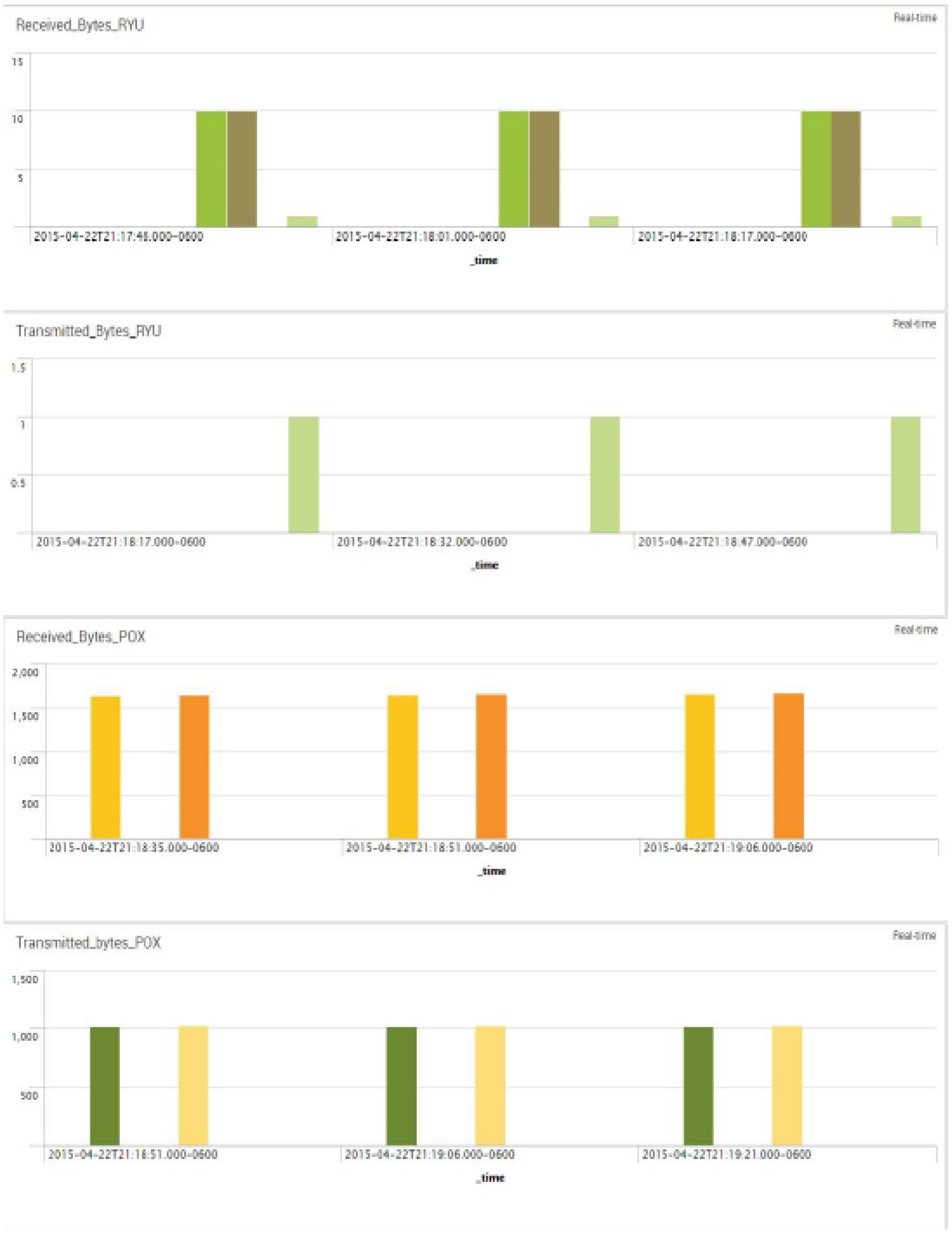

Figure 6.2: Interface utilization dashboard for RYU and POX controllers

VI. Conclusions and Future Research

The project has been successfully implemented as a proof-of-concept to provide an answer to the research question upon which this paper was based. As concluded from the discussion of results section above, it is clearly possible to develop a unified fault and performance monitoring tool that can interface with multiple SDN controllers, as exhibited above.

It is possible to develop this tool further to incorporate several features that would make it more accessible to network engineers. This includes adding additional KPIs for monitoring, such as minimum and maximum response times, packet loss, CPU utilization, memory utilization, etc. These would provide network engineers with a more complete view of the network.

Another issue that is to be handled is the issue of scale. As the number of SDN controllers increases, the tool will have to access each controller be accessed with individual computing resources by implementing a multi-threading mechanism. In addition, the current implementation of the tool requires users to add each additional controller manually to a log file, for the tool to start monitoring. This becomes unfeasible beyond a certain point, and the next step of development would be for the tool to dynamically identify each additional controller.

Another issue of scale that can be worked on is the deployment of large Hadoop clusters to store the data being generated. As a proof-of-concept, the current solution is able to handle raw data using persistent data structures that have been used in this project. However, as a practically deployed tool, a scalable multi-master database such as Cassandra or distributed database HBase can be used to store data in NoSql format on Ext3 file systems. The data can be subjected to map reduce using the Hadoop distributed processing technology to maintain redundancy and speed up queries to the database.

As mentioned in the discussion of results section, the Pox and OpenDaylight controllers display a very small traffic flow across the network topology even though no traffic is being generated at that point of time. This anomaly has to be investigated further using a packet capture mechanism, so as to identify the root cause.

Another huge scope for future research is in implementing a fault correction mechanism for certain KPIs. This can include failover of traffic in case of interface overutilization, a clearing of error counters in case of stagnant interface errors, etc. This would automate a lot of the tasks that a network engineer would have to perform manually.

There is also further scope for research and development of the front-end user interfaces, to make them as intuitive to the network engineer as possible. As the customer-facing part of the solution, the GUI plays a large part in the adoption of the tool. HTML5 with twitter bootstrap is the current trend observed for developing web applications. Certain Javascript frameworks such as AngularJS and Javascript can be used to improve the efficiency of overall module.

VII. Appendix

Appendix 1: List of Resources used

Controllers	Open Daylight FloodLight Ryu Pox
Database	Persistent Data Storage Systems (Flat Files)
Reporting	Splunk
Programming	Python 2.7
Performance benchmarking	CBench
Documentation	Sphinx

Appendix 2: List of Figures

Figure	Title
Figure 1	Architecture for the *Expedition* tool
Figure 2	Time convergence of Dijkstra's algorithm
Figure 3	Topology used for Dijkstra's algorithm calculation
Figure 4	Topology used for interface utilization calculation
Figure 5	Throughput and Latency of Ryu controller
Figure 6.1	Interface utilization dashboard for Floodlight and Opendaylight controllers
Figure 6.2	Interface utilization dashboard for RYU and POX controllers

VIII. References

[1] Kim, A. and Feamster, N., "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, Feb. 2013.

[2] Ennis, S., "Service Assurance for Network Virtualization is about Future Proofing",04/16/14.Retrieved October 12, 2014. Available:http://www.monolith-software.com/blog/2014/04/service-assurance-for-network-virtualization-is-about-future-proofing

[3] ONF, "Software-Defined Networking: The New Norm for Networks". White paper. Open Networking Foundation. April 13, 2012. Retrieved October 12, 2013.

[4] Greenberg, A., Hjalmtysson, G., Maltz, D. A., Myers, A., Rexford, J., Xie, G., and Zhang, H., "A clean slate 4D approach to network control and management," ACM SIGCOMM Computer Communication Review, vol. 35, no. 5, pp. 41-54, 2005.

[5] Kuklinksi, S. and Chemouil, P., "Network Management Challenges in the Software-Defined Networks", IEICE, vol E97-B, No. 1 pp 2-9 2014.

[6] Gelberger, A; Yemini, N.; Giladi, R., "Performance Analysis of Software-Defined Networking(SDN)," Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS),2013 IEEE 21st International Symposiumon , vol., no., pp.389,393, 14-16 Aug. 2013 doi: 10.1109/MASCOTS.2013.58

[7] Kempf, J.; Bellagamba, E.; Kern, A; Jocha, D.; Takacs, A; Skoldstrom, P., "Scalable fault management for OpenFlow," Communications (ICC), 2012 IEEE International Conference on , vol., no., pp.6606,6610, 10-15 June 2012 doi: 10.1109/ICC.2012.6364688

[8] Slattery. T, "Monitoring a Software Defined Network, Part 1". Retrieved September,2014 Available: http://www.nojitter.com/post/240164599/monitoring-asoftware defined-network-part-1

[9] Wangdong, W., Qinglei, Q., Xiangyang, G., Yannan, H., Xirong, Q., "Autonomic QoS Management Mechanism in Software Defined Network", Communications,China, vol 11, No. 7, pp 13-23, 2014

[10] Li, H., Que, X., Hu, Y., Xiangyang, G., and Wendong, W., "An autonomic management architecture for SDN-based multi-service network," in *2013 IEEE Globecom Workshops (GC Wkshps)*, 2013, pp. 830–835.

[11] Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., Zolla, J., Hölzle, U., Stuart, S., and Vahdat, A., "B4: Experience with a Globally-Deployed Software Defined WAN", SIGGCOMM.

[12] Dowling, B.,"10 Free Server & Network Monitoring Tools that Kick Ass",11/04/2009.Retrieved October 12,2014 Available: http://sixrevisions.com/tools/10- free-server-network-monitoring-tools-that-kick-ass/

[13] Unknown, "JSON: The Fat-Free Alternative to XML", Retrieved October 12,2014, Available: http://www.json.org/xml.html

[14] Nickolaisen, N.,"Agile project management helps overcome IT obstacles" .Retrieved October 12,2014 Available : http://searchcio.techtarget.com/tip/Agile-project- management-helps-overcome-IT-obstacles

[15] Rousse, M., Gibilisco, S.," Northbound interface / Southbound interface", Available: http://whatis.techtarget.com/definition/northbound-interface-southbound-interface

[16] Tootoonchian, A., Gorbunov, S., Ganjali, Y., Cacado, M., and Sherwood, R., 2nd USENIX workshop on Hot topics in management of internet, cloud, and enterprise networks and services. Available: https://www.usenix.org/system/files/conference/hot-ice12/hotice12-final33_0.pdf

[17] Feamster, N., Rexford, J., Zegura, E., "The Road to SDN: An Intellectual History of Programmable Networks", ACM SIGCOMM Computer Communication Review, vol 44 issue 2, pp 87-98, 2014