Communications
2015; 3(5): 128-136
Published online August 17, 2015 (http://www.sciencepublishinggroup.com/j/com)
doi: 10.11648/j.com.20150305.18
ISSN: 2328-5966 (Print); ISSN: 2328-5923 (Online)

Effects of Linux Scheduling Algorithms on Mininet Network Performance

Mohammed Basheer Al-Somaidai, Estabrak Bassam Yahya

Dept. of Electrical Engineering, Mosul University, Mosul, Iraq

Email address:
mohammedbasheerabdullah@uomosul.edu.iq (M. B. Al-Somaidai), eng_est_1990@yahoo.com (E. B. Yahya)

To cite this article:
Mohammed Basheer Al-Somaidai, Estabrak Bassam Yahya. Effects of Linux Scheduling Algorithms on Mininet Network Performance. *Communications*. Vol. 3, No. 5, 2015, pp. 128-136. doi: 10.11648/j.com.20150305.18

Abstract: Software Defined Network (SDN) is considered a burgeoning technology in the field of computer networks particularly and in communication technologies in general. A promising architecture of SDN mainly depends on decoupling mechanisms of the control and management plane from the data forwarding plane in network device. Besides, the use of programmable interfaces between network layers. In another word, SDN uses open, flexible, and dynamic architecture that is defined through the use of different software's programming languages. Simulation and emulation network platforms play an important role in studying and evaluating different networks design and performance. Mininet is the most popular SDN platform. This research is concerned with the impacts of operating system scheduling algorithms used by Mininet emulator on network performance with different controllers and topologies types and sizes. It has been noted that network performance under PROC scheduling algorithm was better and more stable than those under CFS or RT scheduling algorithms. As well, when network topology be more complicated, i.e. contains a large number of switches and presence of loops, the network performance is worse than that with simple topology especially, this case is more worse with RT scheduling algorithm.

Keywords: SDN, Mininet, RT, CFS, PROC

1. Software Defined Networks (SDN)

Computer networks technologies are constantly evolving and during the last decade, they changed our lifestyles in many aspects. People in the world use the unlimited services that are provided by Internet networks more and more every day. They use Internet for research, education, businesses, online banking, online gaming, shopping, and the new social networking applications. Thus, the Internet networks based on current architecture have become too complicated and gigantic. In addition, current network devices have specific complex structure, which combines control and management operations along with data forwarding operation in the same physical device. A solution that is able to meet the requirements of continuous network development and provide dynamic base environment is needed. Software Defined Networking (SDN) technology may play an important role to solve computer networks challenges [1,2]. SDN technology is getting a lot of interest around the world. Furthermore, SDN architecture and concept can be integrated in smart ways with other network technologies such as Network Functions Virtualization (NFV), cloud computing, Internet of Things (IoT) and Data Center (DC)/WAN technologies [3-5]. Therefore, observers are talking about a great and prosperous future for the software defined networking (SDN) in particular along with information technology and electronic services in general in various aspects of life.

Based on SDN technology, the control and management plane is physically segregated from the data forwarding plane, where the data forwarding plane resides on network devices and the logical mechanism function to control and manage network performance is put in centralized unit represented by the controller [1,6,7]. The controller runs special software that is written using one of the programming languages such as C/ C++, Java, and Python to take decisions about the forwarding method for new incoming packets in each network device, while the forwarding plane in physical devices performs packet forwarding based on these decisions. SDN designers use open Application Programming Interfaces (APIs) to communicate between software controller and other network layers (physical/application), where network engineers are able to use this APIs to configure network behavior in such a way and provide backward compatibility with new applications [7]. Fig. 1 shows SDN architecture model. The separation of control and management function from network forwarding device and put

it in a centralized unit besides the use of open APIs provides flexible, programmable, cost effect, vendor agnostic and active network architecture [8]. The OpenFlow wire protocol represents the most prominent communication southbound APIs that are used in SDN architecture to connect the control and management unit (i.e. the controller) with the data forwarding unit (i.e. OpenFlow switches) over a secure channel using Transport Layer Security (TLS) or over a Transmission Control Protocol (TCP) channel. It has the capability to establish control session with controller, modify matching entries that are used by the switch to forward a packet, specify actions that will be executed against each matched packet, and defining a structure of different messages types such as flow modification messages, statistical description messages, error messages, switch status messages, and several other messages [9, 10].

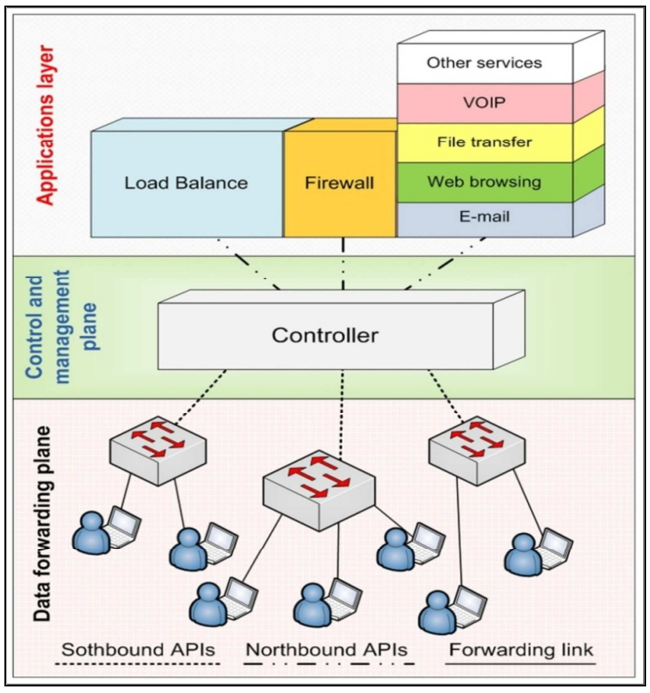

Fig. 1. *SDN architecture model.*

2. Related Works

B. Lantz, et. al. [11] analyzed the performance of Mininet emulator; which is a virtual environment developed by Stanford University that can be used to emulate a number of nodes in virtual network within single test machine, in order to develop, interact with, and customize the SDN concept with OpenFlow protocol. This study showed Mininet ease of use, scalability, and limitations.

S. Yeganeh, et.al. [12] studied the concept of decupling control and management plane from data forwarding plane in SDN and discussed scalability trade-offs in SDN design space and challenges in this field. The overload on controller plays the pivotal role in network scalability based on SDN paradigm but SDN architecture is considered very promising to solve these challenges in the future.

M. GroBmann, and S. Schuberth [13] developed a method to initialize automatically virtual network topologies built in Mininet through the use of Internet Topology Zoo (ITZ); which can be defined as a store for data of network topologies in graphical formats, to build up real-world test-suites at small scale. Also, they developed a distributed test suite for evaluation purposes that uses a Distributed Internet Traffic Generator (D-ITG). The method of building network topologies proposed in this work provides an efficient and easy way to construct virtual network topology in Mininet without the need to use Python script.

B. Astuto, et.al. [14] provided historic review about programmable network idea from its beginning time down to the SDN revolution. The study presented the architecture of SDN and discussed OpenFlow features, application and related software to deploy and develop SDN networks based on OpenFlow standard, which includes emulation/simulation platforms, software controllers and virtual/current implementation of programmable switches.

P. Wette, et.al. [15] proposed MaxiNet platform to extend Mininet emulator to span an emulated network over several physical test machines in order to solve scalability challenge that faced performance of Mininet and constrained its use to build networks that have only several hundred nodes as a result of resources limitation. This approach of distributed emulation of SDN enables researchers to emulate large network such as data center network. Therefore, they introduced a traffic generator for data center traffic and used it in emulated network example about data center consisting of 3200 hosts on a cluster of only 12 physical test machines.

3. Mininet SDN Platform

There is a number of simulation/emulation platforms that could be used to study and evaluate SDN concept in virtual environment. Mininet emulator is considered the most popular and efficient SDN based OpenFlow protocol platform due to a number of features it supports including integrity, flexibility, availability, and simplicity in building SDN network. Such networks contain OpenFlow switches, controllers, hosts, and connecting them either through Ethernet interfaces, or by secure interfaces based upon Secure Shell (SSH) protocol in a single Linux kernel [11]. Mininet uses Ubuntu Linux distribution with a different version as an operating system. In addition to the capability to use Fedora Linux operating system but with some limitations in supporting a number of features. Furthermore, It uses lightweight virtualization methods to provide efficient test environment in a single test machine which gives the developer or the researcher complete vision about real test bed network performance.

In Mininet both the controllers and the switches can be installed and run either on the user namespace or in the kernel namespace to accelerate its performance. Besides, each host in Mininet represents a standalone shell process that behaves like an actual real machine which can be used to represent terminal edge in network or to represent a server program such as HTTP server and FTP server, etc. A host in Mininet connects directly

with main Mininet module (mn). Fig. 2 shows an example of a small network in Mininet [11,5]. The network elements can interact with each other and debugging is done using Command Line Interface (CLI). This method of virtualization provides best sharing mechanism for resources of the operating system, fast network boot up, and easy to setup network elements.

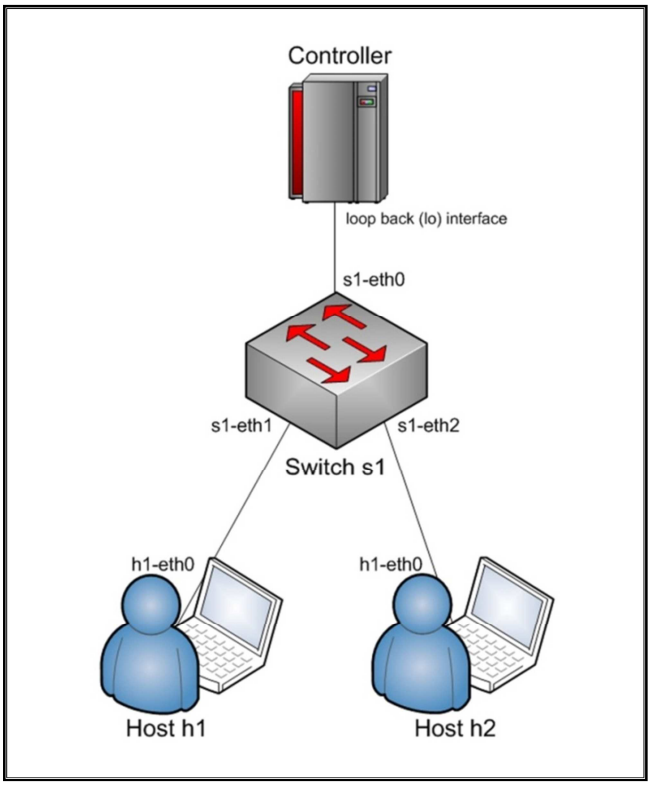

Fig. 2a. *Example of a small network in Mininet.*

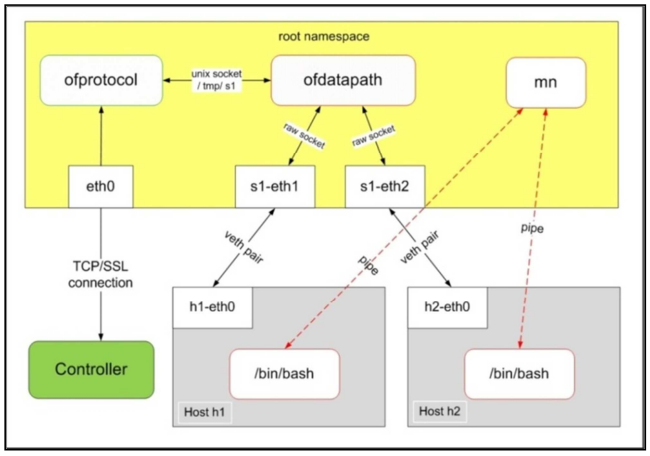

Fig. 2b. *Network implementation in Mininet.*

On the other hand, Mininet has limitations in scalability for a very large network where it basically depends on the emulating machine specifications which include CPU type, CPU frequency, system memory size, and RAM memory size [11]. The resources of the laptop that was used in this research are an Intel core™ i5-3320MB CPU@2.60GHzx4 with RAM:8GB, 128GB SSD, 500GB hard disk with Ubuntu 12.04 LTS-32-bit, and a kernel Linux 3.11.0-15-generic as operating system.

4. Linux Scheduling Algorithms

As mentioned above Mininet emulator basically uses Ubuntu Linux distribution to run its platform models and it is integrated with kernel programs to build, configure, analyse, and interact with designed networks. Linux is considered one of the most important open source, powerful, flexible, and a permanently up to date operating systems [16]. It provides a number of features to the user that cannot be found in other operating systems such as long running time without shutdown or reduction in system performance, multiple users who have access to the system and perform multiple tasks at the same time.

One of the major goals of operating system designers is to minimize overhead and delay of operating system services. Therefore, they choose a certain scheduling algorithm to improve operating system performance. In Linux, a number of scheduling algorithms can be used and that depends on its performance on the division of the CPU cycle in the form of time slices that are used to run several tasks. Time slices are allocated to each task according to task priority and policies used by this task. This is obtained because system timer performs periodic interrupts so that the scheduler can choose the task that will be executed in the next CPU time slice. The tasks that are run on Linux can explicitly be classified into four classes. They are Real Time (RT) scheduling class, Fair scheduling class, and two special purpose scheduling classes which include idle scheduling class, and stop scheduling class [17]. Mininet emulator can use RT, or CFS scheduling algorithm to schedule its processes and share the system resources.

4.1. Rt Scheduling Algorithms

Real time processes have very strict timing processing and scheduling conditions. Therefore, the process by the CPU units to prevent loss of incoming real data or sequential processing operations should never be delayed [18]. Typical real time programs are video and audio programs, controller programs, and measurement and monitoring programs etc. There are two RT scheduling modes, the first one is called SCHED_FIFO (First in-First out) mode where the current higher priority task doesn't have a limited time slice and should be executed until be terminated. The other mode is called SCHED_RR (Round Robin) mode where every task has a specific number of time slices to execute and when this time is over the task is added to the end of queue and its time slices are assigned to the next real time task [17].

4.2. Fair Scheduling Algorithms

The delay problem in earlier Linux kernels scheduling algorithms such as O(1) scheduling algorithm has been resolved in Linux kernel 2.6.23 through the Completely Fair Scheduling (CFS) algorithm that is designed to ensure fairness between tasks by dividing CPU time among runnable tasks in approximately ideal case using high resolution nanosecond-accurate time slices. For example, if two tasks would be runnable, they would apparently get 50% processing

power for each. Therefore, assuming the two tasks are to be started at the same time, the execution time or runtime of each task at any moment is exactly the same, therefore completely fair. That means, each task runs for an infinitesimal small amount of time but with full processing power, then the full processing power is switched to the other task.

Since it is physically not possible to drive current processors in that way and it is highly inefficient to run for very short times due to switching cost, CFS tries to approximate this behavior as closely as possible. It keeps track of the runtime of each runnable task. It is called virtual runtime, and tries to maintain an overall balance between all runnable tasks at all times [17].This means that CFS uses dynamic time slices with value which depends on task priority, and current tasks load in the system [19].

Besides the possibility to use RT and CFS, Mininet has another scheduling algorithm called "PROC". In this algorithm file system, user ID space, process ID space, kernel, shared libraries, device drivers and other common node are shared between processes that arrange as a control group (cgroup) [11]. CPU time is fairly shared among all cgroups which have not been used up their time slice for the given period. This feature has similar goals to the Linux real time (RT) scheduler, which also limits process time execution, but differs in that when no limits are set; it acts in a work-conserving mode identical to the default Linux completely fair scheduler (CFS).

5. Research Methodology

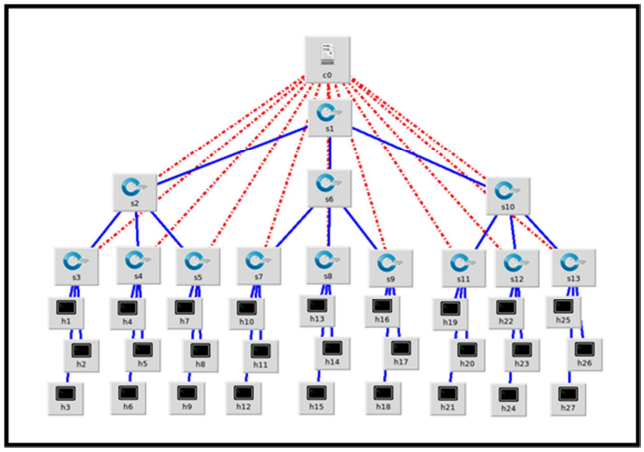

Fig. 3. *Tree (3,3) network topology.*

Tests in this research include the effect of CFS, RT, and PROC scheduling algorithms with varying CPU time values. Mininet package has a number of Python script examples that are used to define Mininet capability and the user can modify them to create new functionality. In order to implement an easy method of the designed test in Mininet, we modified cpu.py example script to implement tree topology (depth=3, fan out=3) as shown in Fig. 3, where the network has 13 switches and 27 hosts in three cases. The first case, we built an OpenFlow network that uses Open vSwitch which supports version 1.3 and are controlled either by OpenFlow remote controller or by control and administration tool (ovs-ofctl) to add proactive flow entries. The second case, we used an Open vSwitch in standalone mode (non-OpenFlow) to represent conventional network. Finally, in the third case, the network has both OpenFlow and non-OpenFlow switches to represent a hybrid network. For each network case, the amount of end-to-end throughput is calculated (i.e. the test is done between first host in network h1 and the last host in the network h27) under the effect of using different scheduling algorithms RT,CFS, and PROC with varying CPU time values from (1%-100%).

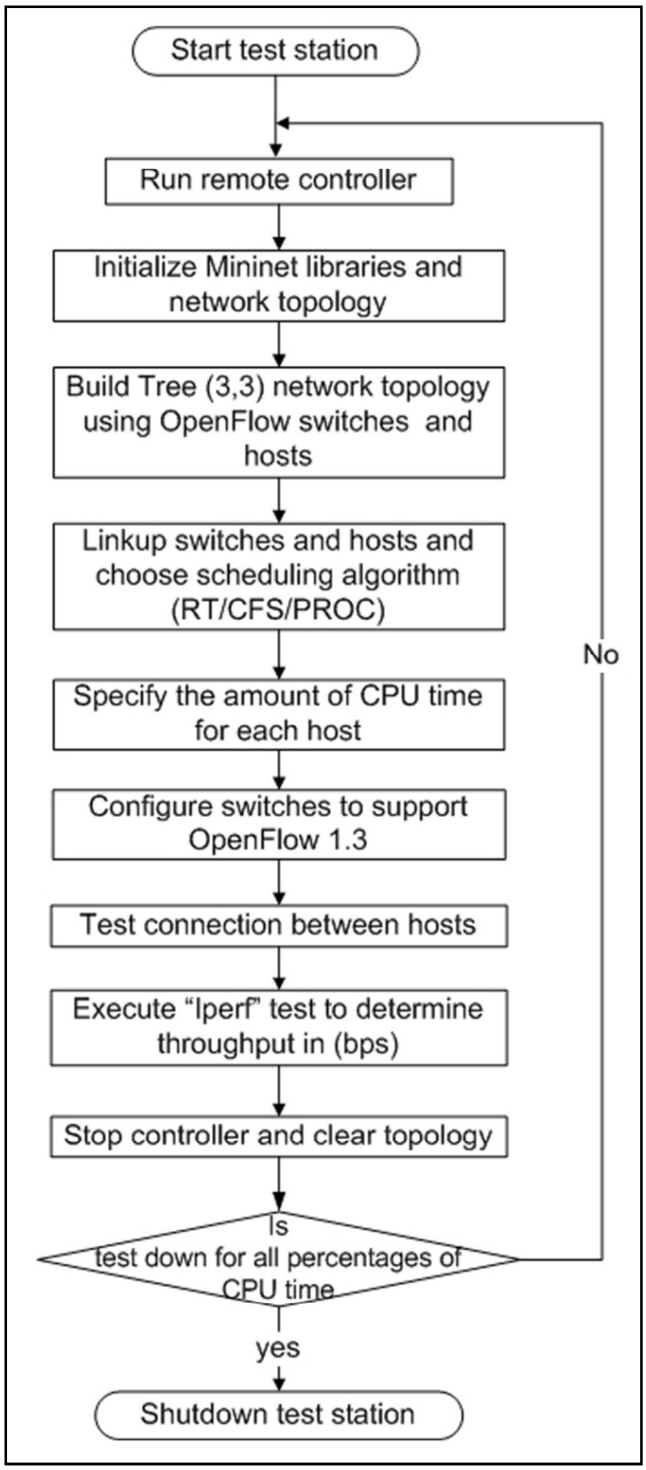

Fig. 4a. *Methodology of scheduling algorithms tests using remote controller.*

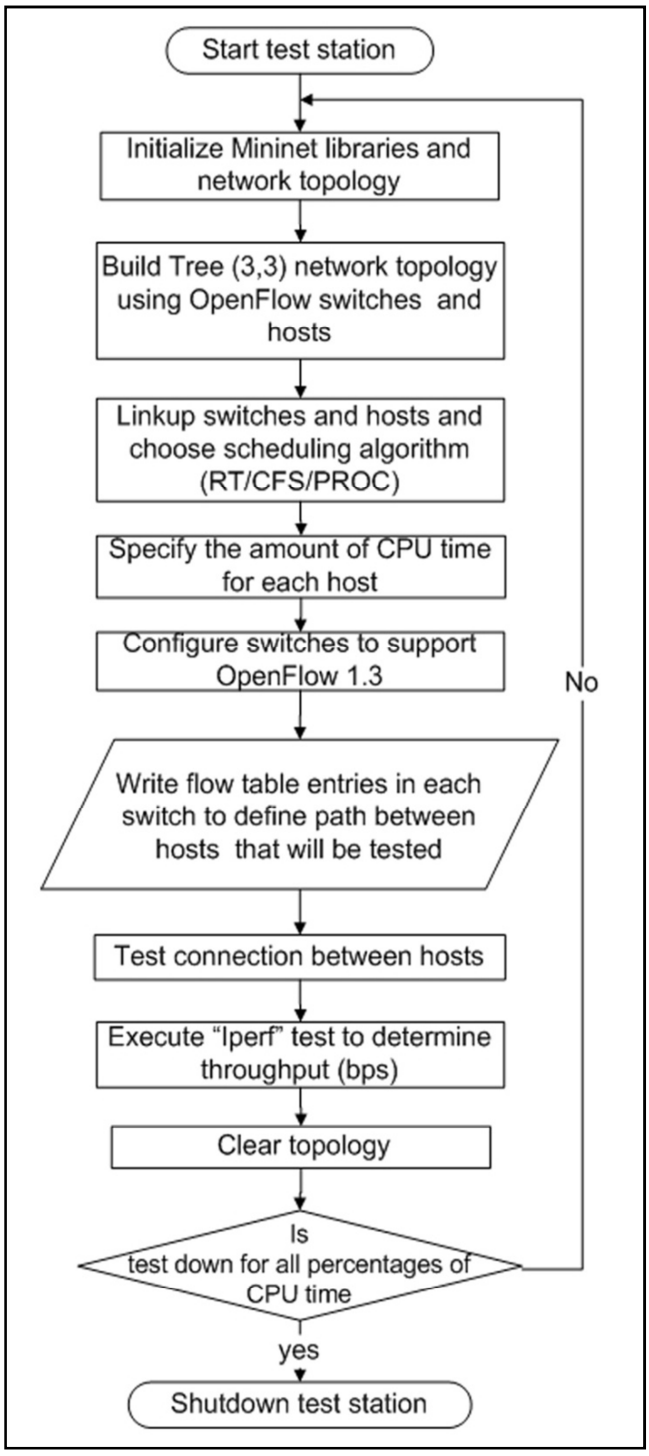

Fig. 4b. *Methodology of scheduling algorithms tests using ovs-ofctl tool.*

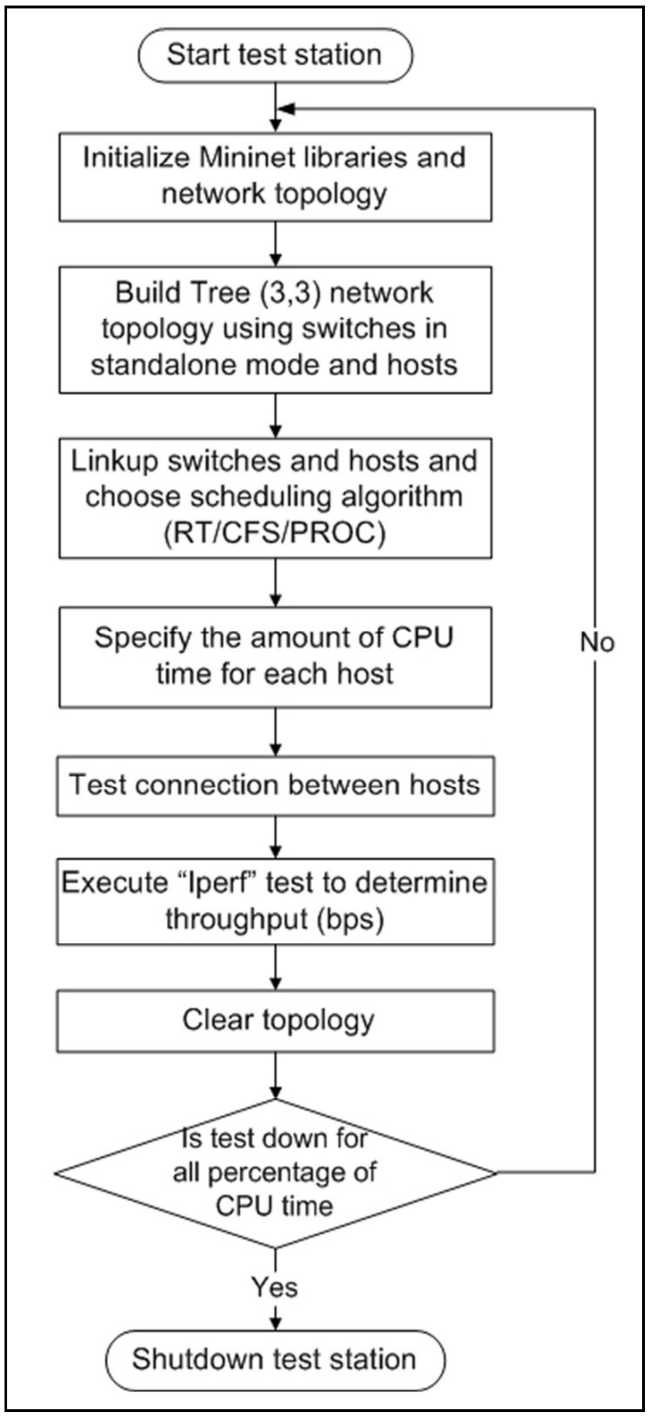

Fig. 4c. *Methodology of scheduling algorithms tests for conventional network.*

Fig. 4 illustrates methodologies used in each case. After initializing Mininet libraries which are represented by classes, and functions used to build and configure network nodes (switches, controller, and hosts) and building network topology, the scheduling algorithm will be chosen and the duration of the time slices for each host in the network is specified. When network starts, each switch in the network is configured to support OpenFlow 1.3 using ovs-vsctl configuration tool. Besides, in case of using an ovs-ofctl to control switches behavior, the flow entries that define the path between hosts will be stored in switches (proactive flow entries) without the need to use any controller. Two simple benchmarks (Ping, and Iperf) were used to validate network connection and calculate end-to-end network throughput respectively.

The amount of throughput is also affected by network topology as a result to complex processes needed to perform topology discovery mechanisms by the controller and finding the path between source and destination for transmitted packets especially when the network has loops. Therefore, the above methodology is repeated for different topologies.

6. Results and Discussion

Tests are carried out for three cases of networks to study the effects of different scheduling algorithms, which include RT, CFS, and PROC as illustrate in the previous section. The first case is OpenFlow based network using tree (3,3) topology controlled by remote controller (Ryu, OVS-controller) or controlled by proactive entries added by ovs-ofctl administration tool. Fig. (5), (6), and (7) show these tests results. The results show the throughput as a function of CPU time for the three scheduling algorithms using log-log scale, where the PROC scheduler, which is designed to provide high level of virtualization and resource sharing in Mininet environment, has the best performance while the other two schedulers have weak performance for short CPU time (i.e. less than 10%). Furthermore, from the results, it can be noted that the OpenFlow network controlled by OVS controller has the worst change and less amount of throughput on the contrast of the results of using Ryu and ovs-ofctl tool.

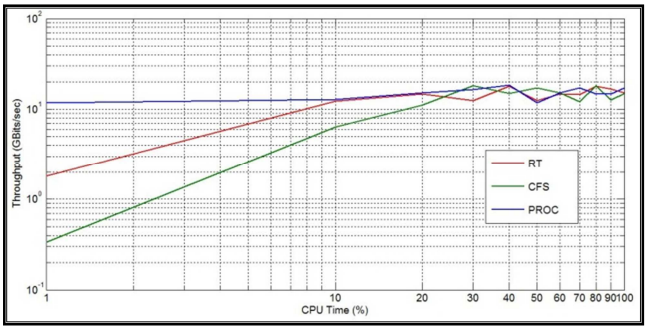

Fig. 5. *End-to-End throughput in OpenFlow network controlled by OVS controller.*

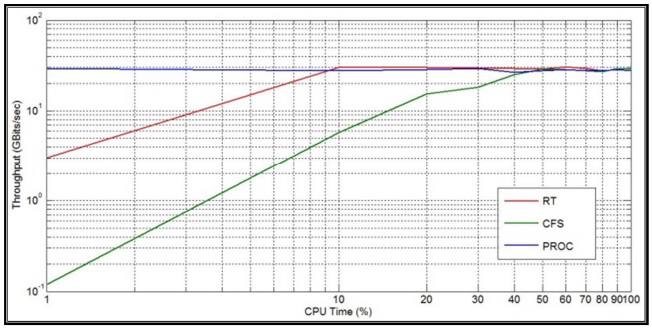

Fig. 6. *End-to-End throughput in OpenFlow network controlled by Ryu controller.*

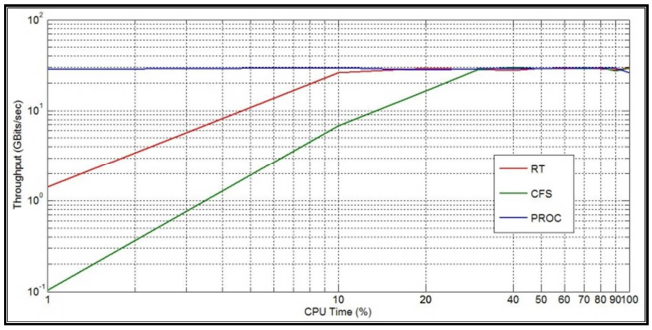

Fig. 7. *End-to-End throughput in OpenFlow network controlled by ovs-ofctl.*

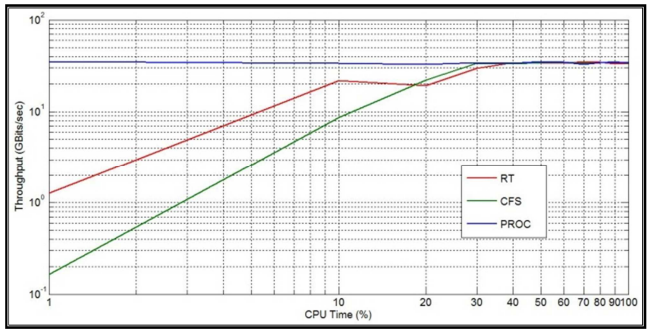

Fig 8. *End-to-End throughput in conventional network.*

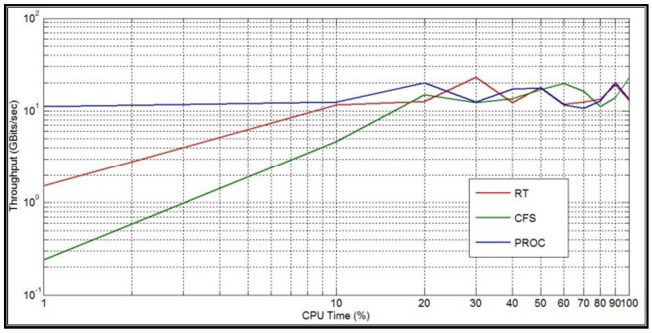

Fig. 9. *End-to-End throughput in hybrid network controlled by OVS controller.*

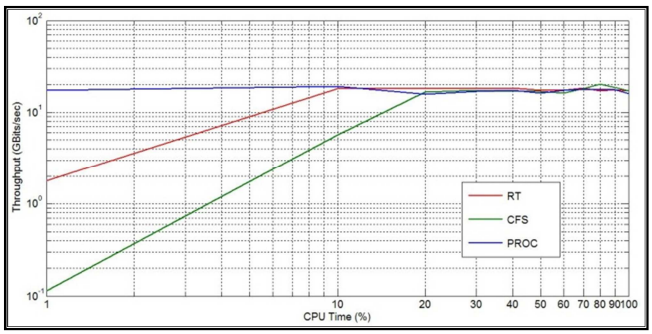

Fig. 10. *End-to-End throughput in hybrid network controlled by RYU controller.*

The second case is a conventional network (non-OpenFlow network). Fig. 8 shows the throughput in conventional network for different scheduling algorithms. When compare between the results of OpenFlow network and conventional network, it has been noted that the segregation of control and management logical functions from data forwarding units does not affect the network performance. Fig. 9, and 10 show the results of the third case for a hybrid network which runs both OpenFlow switches and non-OpenFlow switches (conventional switches). It can be seen that the amount of network throughput is less than pure OpenFlow network or pure conventional network as a result to the complexity of the method to control the network, and the need to additional processing to achieve compatibility between OpenFlow switches and conventional switches.

The amount of end-to-end throughput with different scheduling algorithms is affected when we change the network topology as a result of additional processing to discover network topology with increase in network size, increase in the number of paths between nodes, and presence of loops in the

network and this leads to change in routing roles required to direct packets through the network. A number of topology types and sizes were tested with different scheduling algorithms and the results are shown in Fig. 11, 12, 13, and 14.

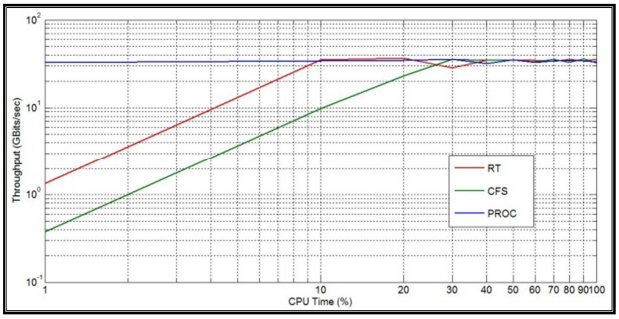

Fig. 11a. *End-to-End throughput in single (2) OpenFlow network controlled by Ryu controller.*

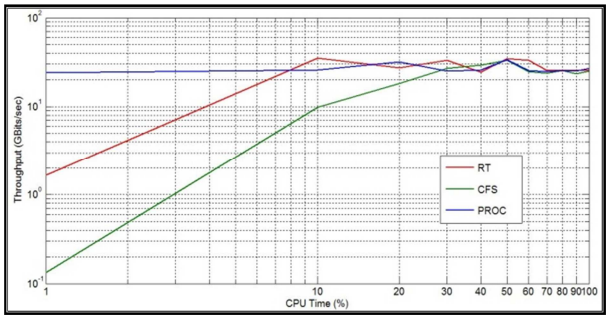

Fig. 11b. *End-to-End throughput in single (25) OpenFlow network controlled by Ryu controller.*

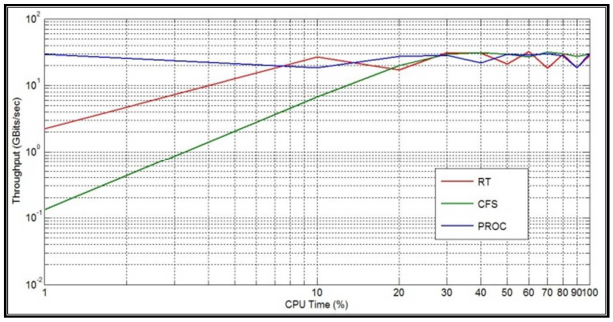

Fig. 12a. *End-to-End throughput in linear (3) OpenFlow network controlled by Ryu controller.*

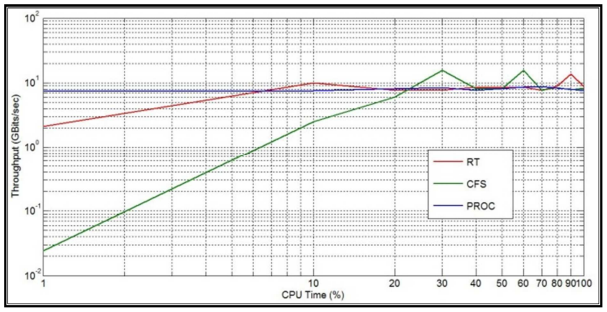

Fig. 12b. *End-to-End throughput in linear (25) OpenFlow network controlled by Ryu controller.*

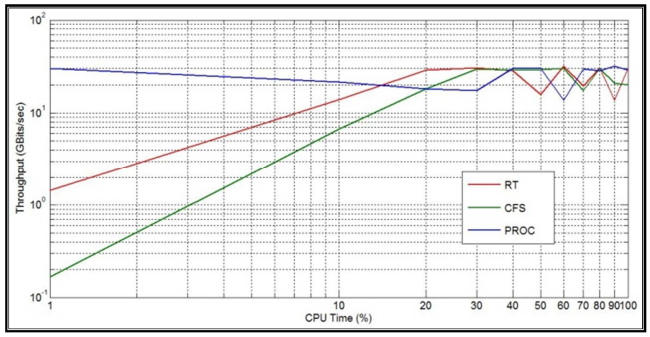

Fig. 13a. *End-to-End throughput in tree (2,2) OpenFlow network controlled by Ryu controller.*

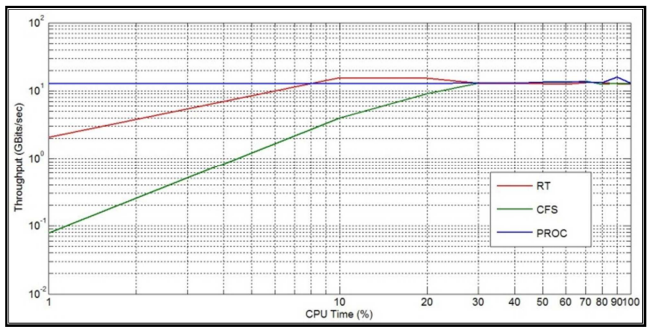

Fig. 13b. *End-to-End throughput in tree (5, 2) OpenFlow network controlled by Ryu controller.*

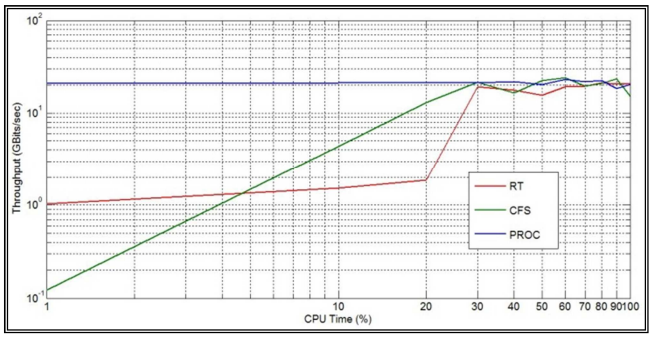

Fig. 14a. *End-to-End throughput in mesh (2, 2) OpenFlow network controlled by Ryu controller.*

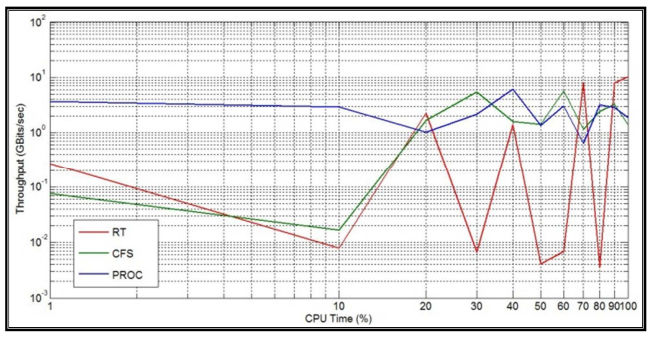

Fig. 14b. *End-to-End throughput in mesh (5, 5) OpenFlow network controlled by Ryu controller.*

From the results, the single topologies have the best throughput and performance with different scheduling algorithms because of having a single switch and direct path for each host, which will cause low processing operations. Linear, tree, and mesh topologies have many switches connected with

each other in a certain way leading to the presence of different paths between hosts. Therefore, they need a lot of processing to detect path between hosts. This is shown clearly in mesh (5, 5) network, where there are several paths to achieve communication between any two points, which requires the use of spanning tree protocol (STP) to find a path. In this case, the RT scheduling algorithms fails to achieve a stable performance of the network, while both of CFS and PROC scheduling algorithms have a better performance, especially for large CPU time (Fig. 14 b). This bad performance can be considered one of the limitations faced the work on virtual tests environment and currently there are many attempts to address these limitations. One of the important ways of these attempts through the use of virtualization technologies such as multi-core processor virtualization and network virtualization (i.e. network slices).

7. Conclusions

Through the study in this research we can note that the network performance is affected by the amount of available resources and properties in the operating systems to run different network elements including software switches and controllers. This is considered one of the main determinants facing the expansion in construction of large networks either in the virtual environment or in testbed. In Mininet, The type of scheduling algorithms used by the operating system to manage processes execution and the amount of CPU time for each process affects network throughput. When the percentage of CPU time is smaller than 10% and we used RT or CFS scheduling the end-to-end throughput was smaller than the amount of throughput when we used PROC scheduling algorithm. Therefore, network performance under PROC scheduling algorithm, which is chosen as a default scheduling algorithm for Mininet emulator, was better and more stable than that under CFS or RT scheduling algorithms. Besides, the number of switches and how to shape with each other to construct network topology have an axial effects on network performance as a result to the need of a lot of processing to find path between any two ends. RT scheduling algorithm has been showed the worse performance along with large processing time followed by CFS and PROC scheduling algorithm. We aim through this work to shed some light over the limitations facing the construction of SDN networks either in virtual test and evaluation environment or in testbed. In another study, we will look forward to overcome some of these limitations including scalability restriction through using virtualization technologies to simplify network topology and best system resources sharing among network components.

References

[1] Open Network Foundation (ONF),"Software-defined networking: the new norm for networks", April 2012, Available at: https:// www.opennetworking.org/images/stories/downloads/openflow/ wp-sdn-newnorm.pdf, accessed on 25/10/2014.

[2] L. MacVittie, "The Programmable Network", white paper, F5 Networks, Inc., 2013.

[3] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Holzle, S. Stuart and A. Vahdat, "B4: experience with a globally-deployed software defined WAN", SIGCOMM'13 Hong Kong, China, pp. 3-14, August 2013.

[4] Migration Working Group, "Migration use cases and methods", Open Networking Foundation (ONF), February 2014, available at : https:// www.opennetworking.org/images/stories/downloads/sdn-resources/use-cases/Migration-WG-Use-Cases.pdf, accessed on 25/10/2014.

[5] Open Network Foundation (ONF), "OpenFlow-enabled SDN and network functions virtualization", February 2014, available at :https://www.opennetworking.org/images/stories/downloads/ sdn-resources/solution-briefs/sb-sdn-nvf-solution.pdf, accessed on 25/10/2014.

[6] T. Nadeau and K. Gray, "SDN: Software Defined Networks", first ed., O'Reilly Media, Inc., August 2013.

[7] Open Network Foundation (ONF), "SDN architecture", June 2014, available at: https:// www.opennetworking.org/images/stories/downloads/sdnresources/technical reports/TR_SDN_ARCH_1.0 _06062014.pdf, accessed on 25/10/2014.

[8] S. Azodolmolky, "Software Defined Networking with OpenFlow", Packt Publishing, 1st ed., October 2013.

[9] Open Network Foundation (ONF), "OpenFlow switch specification, version 1.4.0 (wire protocol 0x05)", October 2013. Available at: https://www.opennetworking.org/images/stories/downloads/sdn resources/onfspecifications/openflow/openflow-spec-v1.4.0.pdf, accessed on 25/10/2014.

[10] A. Clemm, and R. Wolter, "network-embedded management and applications understanding programmable networking infrastructure", Springer New York, 2013.

[11] B. Lantz, B. Heller, and N. McKeown, "A network in a lap-top: rapid prototyping for software-defined networks", In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks New York, 2010.

[12] S. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking", IEEE Communications Magazine, February 2013.

[13] M. Großmann, S. Schuberth, "Auto-Mininet: assessing the Internet topology zoo in a software-defined network emulator", 7ter Workshop Leitungs-, (MMBnet 2013), Hamburg, Germany, 2013.

[14] B. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: past, present, and future of programmable networks", IEEE Communications Surveys & Tutorials, in press, January 2014.

[15] P. Wette, M. Draxler, A. Schwabe, F. Wallaschek, M. Zahraee, and H. Karl, "MaxiNet: distributed emulation of software-defined networks", IFIP networking conference, copyright © ISBN 978-3-901882-58-6, 2014.

[16] G. Glass, K. Ables, "Linux for programmers and users", Prentice Hall, copyright © Pearson Education, Inc., February 2006.

[17] C. Blue, V. Seeker, "Process scheduling in Linux", University of Edinburgh, December 2013, available at:http://www.criticalblue.com/news/Wpcontent/uploads/2013/12/linux_scheduler_notes_final.pdf on 25/10/2014.

[18] D. Bovet, M. Cesati, "Understanding the Linux kernel", O'Reilly, 1st. ed., October 2000.

[19] G. Cheng, "A comparison of two Linux schedulers", M.Sc. Thesis, Oslo and Akershus University, College of Applied Sciences, 2012.