Conducting experiments using a platform for evaluation and assessment of SDN performance

Diyana D. Kyuchukova, Georgi V. Hristov

Software-Defined Networks is a brand new network concept that changes the way networks operate. In order to satisfy the network performance there are some issues that needs to be considered. Finding the best controller location in the control plane of a Software-Defined network is an important task that is related to the whole network performance. In order to find methods for improving the network performance it is necessary to perform thorough analyses of the metrics or criteria that affects the controller placement. The analyses can be performed as surveys, but in order to achieve better results simulation experiments or practical studies are required. Based on the results of such research activities, the correlation between the criteria that affect the network performance can be detected. Different simulation and emulation programs can be used for the implementation of such studies. In this paper the authors propose a platform capable of conducting emulations and further analysis and estimation of SDN performance.

Провеждане на експерименти с платформа за изследване и оценка производителността на SDN мрежите (Дияна Д. Кючукова, Георги В. Христов). *Определянето на местоположение за контролерите в управляващата равнина на SDN мрежите е важна задача, която засяга производителността на мрежата. За да се намерят методи за подобряване на работоспособността на мрежа е необходимо да се извършат задълбочени анализи на факторите, които влияят на избора къде да бъде поставен контролера. Тези анализи може да се направят на база проучвания, но за да се постигнат по-добри резултати е необходимо да се пристъпи към провеждане на симулационни и / или практически изследвания, чрез които да се открие корелационната зависимост на факторите, които влияят върху производителността и да се търси метод за нейното подобряване. За провеждането на такива изследвания биха били полезни различни софтуерни симулационни и емулационни продукти. В настоящата статия се предлага платформа за изследване и оценка производителността на Софтуерно дефинираните мрежи.*

Introduction

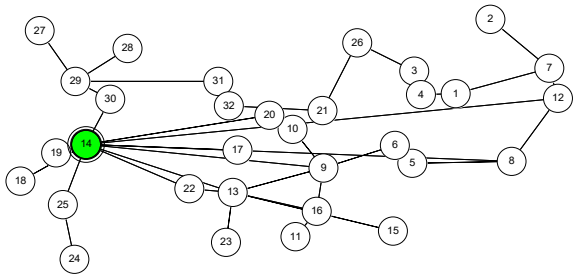
The basic idea behind the SDN network architecture is the separation of the control from the forwarding plane. While this separation significantly simplifies the network operations, it opens questions about the network performance. In the SDNs, the control of the whole network is implemented by a separate device called controller, which plays a vital role in the overall performance of the network. Based on an analysis from several sources [1, 2, 3] it can be concluded that the placement of the controller in the SDN control plane has very huge impact on the whole network performance. Network latency, fault tolerance and controller imbalance are among the criteria that are taken into account when determining the location of the controller. In most cases these criteria are correlated and thus it can be seen that some trade-offs will be necessary. Such trade-offs can be discovered by carrying out simulation or practical experiments. This paper focuses on the process of conducting simulation experiments by using a platform capable of evaluation, assessment and further analysis of the control plane performance. The platform represents a set of tools, which are necessary to run a predefined network topology with a certain controller placement.

The environment needs to be prepared before starting any experiments. It is clear that testing a new product (protocol, technology, etc.) could not be performed in a real working environment. Before starting the implementation of innovations in real environments they need to be explored in a prototype environment. The prototype environment can be copy of the original one, but in the context of SDN networks making a copy of the network itself is

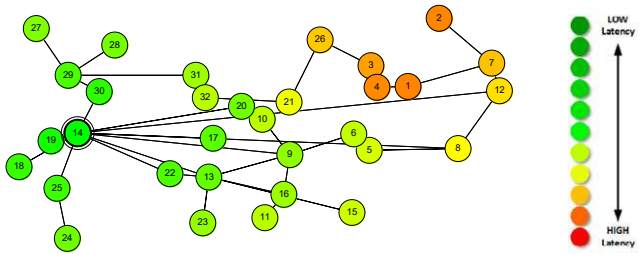
pointless and unprofitable. There are many software products which offer great possibilities that allow users to create prototype network topologies of real networks. The network topologies could be defined software-wise rather than on a hardware level, which would reduce costs during the setup and development time. But there is one condition that needs to be considered. Network designs created on the software level should be deployable on actual hardware. In addition, the software implementation of a network topology can be scaled in order to meet the limitation of the hardware running the emulation platforms.

Running a network topology in a software-based environment

To the best of our knowledge, there are several software platforms that allow user to emulate network topologies. The proposed platform uses the Mininet emulation product that can create a realistic virtual network, running real kernel, switch and application code, on a single machine (either a host or virtual machine) in seconds [4]. Mininet is Linux-based emulation software that make possible splitting a single system into multiple Linux subsystem called containers. The Linux containers are interconnected via virtual links that can be parametrized with delay, jitter, packet loss and bitrate. Mininet can run multiple predefined network topologies, but supports also user defined ones. To run user defined networks, Mininet requires Python scripts that describe the topologies.



a) "Evolink" topology with the placement of one controller



b) "Evolink" topology with the placement of one controller and latency diagram

Fig.1. Example network topology.

In order to show how the proposed platform is organized a basic measurement of performance metrics will be conducted on a test environment. The topology that is used for network experimentation is shown in Figure 1. This topology is taken from *Internet Topology Zoo (ITZ)* - a web based data base that houses more than two hundred and fifty network topologies from around the world [5]. The ITZ platform provides topologies in graphical notation, containing the geographical location of the network devices and the links between them. The file formats of the topologies (.GML and .GraphML) cannot be directly run in the Mininet environment, because it currently supports only Python scripts. This is the reason why the proposed platform uses a parser that will extract the information from .GraphML file and will create a Python script. The last will be used to generate the topology in Mininet.

In addition, the geographic location of the network devices is relevant information, which is used from the parser to determine the latency of the link between the network devices. The latency, in seconds, between two connected nodes A and B is calculated by using the following formula:

$$d = \frac{dist(A,B)}{\upsilon} \quad (1)$$

The distance $dist(A,B)$ between nodes A and B is calculated by the spherical law of cosines that uses the longitude and the latitude of the nodes from the .GraphML topology file. The value υ is a constant and represents the speed of the signal, which is approximated with the speed of light and the reflective factor of 1.52 for optical fiber [6].

The next step after generating the topology is to find the controller location in the control plane. The controller can be placed in the topology randomly, but this would be meaningless since the main goal is to find the best controller location. In order to find a placement of the controller in the topology the proposed platform uses a Matlab based instrument, called POCO (from Pareto Optimal COntroller placement) [2]. To determine the best controller location according to different metrics the POCO-toolset uses an exhaustive evaluation of all possible placements in a given topology.

Regarding the metrics that can be used to calculate the best controller location, they can be grouped in performance-only related metrics and reliability aware metrics [7]. The first group refers to failure free scenarios. A metric related to this group is latency. To find the best controller location, the POCO-toolset should calculate either the average or the maximum

latency between a node and the controller. The corresponding best controller location is when the average or maximum latency has minimal value. For each scenario the POCO-toolset calculates either:

$$L_{avg}(C_i) = \frac{1}{n}\sum min\, SP\,(n_i, c) \quad (2)$$

or

$$L_{max} = \max \text{minSP}(n_i, \text{c}) \quad (3)$$

The shortest path between node n_i and the controller c is depicted as $SP(n_i, \text{c})$, the average latency for the current controller placement C_i is $L_{avg}(C_i)$ and the worst case latency is L_{max}.

The average latency is taken into consideration when determining the controller location in the current evaluation of the performance metric over the Evolink network topology. As shown in Figure 1-b the controller is located at the placement of the node with ID 14. In addition, the POCO-toolset has generated a coloured diagram of the latency between all nodes and the controller. The greener the node colouring is, the smaller the latency between this node and the controller is. After exhaustive evaluation, the POCO-toolset has determined that the average latency has minimum value when the controller is placed at the place of node 14.

After all of these comes the final step, which is related to running the topology in Mininet. The topology is run easily by entering the following command into a Linux terminal:

```
~$sudo ./FileName.py
```

where *FileName.py* is the Python script that defines the topology. The controller placement is a problem that affects only the in-band control plane implementation. In order to emulate in-band control plane an Open vSwitch instance should be used. Open vSwitch, sometimes abbreviated as OVS, is multilayer software, running under Apache 2.0 license. The OVS is well suited to function as a virtual switch in virtual machine environments [8]. To achieve in-band control, the OpenFlow switch should run into Linux kernel otherwise the in-band communication might not work correctly. OVS is an instance of OpenFlow switch that can run directly on a Linux kernel. This is the reason why Open vSwitch is the preferable instance for the proposed platform.

The control plane functionality is implemented by Floodlight controller [9]. The controller is software that can be installed on every commodity server. At the present moment of time there are more than thirty SDN controllers created by different vendors, universities or research groups. They differ from each other in the program languages that are used for their implementation and also they use different runtime multi-threading techniques. Some previous works [10, 11] argue that the type of the SDN controller also affects the performance of the network. This is why the controller in the proposed platform is an interchangeable component.

For the evaluation of the performance metrics for a predefined network topology run in the Mininet environment, the Distributed Internet Traffic Generator (D-ITG) is used. D-ITG is a platform capable to produce traffic that accurately adheres to patterns defined by inter departure time between packets (IDT) and the packet size (PS) stochastic processes [12]. The architecture of the D-ITG traffic generator is shown in Figure 2. Every module is connected through several communication channels to other modules. The most important modules for the following measurements of the Evolink network performance are ITGSend, ITGRecv and ITGDec. ITGSend module is responsible for traffic generation over network under test and for sending process, while ITGRecv modules control reception of the traffic.

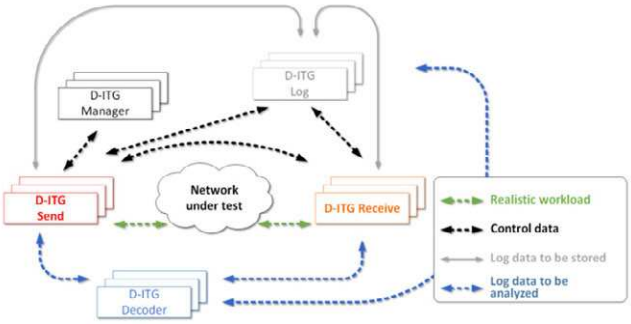


Fig.2. D-ITG traffic generator.

To collect logging information both modules responsible for sending and reception of traffic communicate with ITGLog module via Log Channel. In addition D-ITG decoder is used to provide data files that can be analysed.

Measurements and results

The sender module of the traffic generator can work in a single-flow mode or multi-flow mode [13]. For the purpose of the current measurement, the authors of this paper use ITG sender working in multi-flow mode. In this mode the sender can simultaneously generate several flows addressed to one or more recipients. The flows should be predefined in the script file. The script is made of a line for each traffic flow, so that for n flows the script file has to contain n lines, each of which is used to

specify the characteristics of one flow. The script file generated for the measurements contains the following lines:

```
-a 10.0.0.3 -rp 1003 -C 10000  -t 60000
-a 10.0.0.3 -rp 1004 -C 100000 -t 60000
-a 10.0.0.3 -rp 1005 -C 150000 -t 60000
```

Different receiver ports have been used in order to distinguish individual flows. All flows have been sent to a node with address 10.0.0.3 for the duration of one minute. The duration of the flows are defined in milliseconds with `-t 60000`. All flows traverse the network simultaneously. The difference between the three flows is the bitrates in terms of a packets per second. The first flow transmits data with constant bitrate 10×10^3 packets per second, the second flow transmits 100×10^3 packets and the third flow with bitrate of 150×10^3 packets per second. By default, if not specified, the size of each packet is 512 bytes.

```
------------------------------------------------
Flow number: 1
From 10.0.0.28:44706
To    10.0.0.3:1003
------------------------------------------------
Total time                =     59.960952 s
Total packets             =        598938
Minimum delay             =      0.000000 s
Maximum delay             =      0.000000 s
Average delay             =      0.000000 s
Average jitter            =      0.000000 s
Delay standard deviation  =      0.000000 s
Bytes received            =     306656256
Average bitrate           =  40914.127714 Kbit/s
Average packet rate       =   9988.800712 pkt/s
Packets dropped           =              0 (0.00%)
Average loss-burst size   =      0.000000 pkt
------------------------------------------------
```

Fig.3. Summary results of flow with constant transmission rate of 10000 packets per second at the sender side.

As shown in Figure 3 - the summary results of the flow with constant transmission rate 10000 packets per second, the average bitrate is around 40 Mbps. This is obvious, because 512 Bytes are equal to 4096 bits and since the packets, which have been transmitted are 10000 per second the average bitrate achieved is 40 Mbps. The results can also be plotted using the appropriate software products. For the purpose of this paper the authors use Matlab [14].

The Figure 4 shows the throughput of the first flow. The data is obtained from the sender log file. Obviously the transmission rate is four times bigger than the capacity of the links. The topology under test is built by 10 Mbps links, which is a prerequisite for bigger delays or even packet loss. This can be noticed by the throughput diagram on the receiver side, shown in Figure 5.

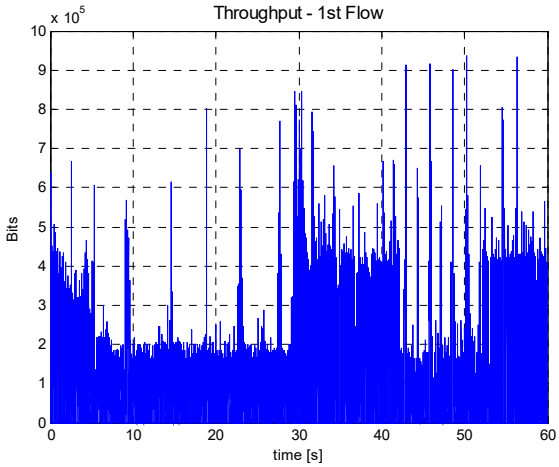


Fig.4. Throughput of the first flow of the sender side.

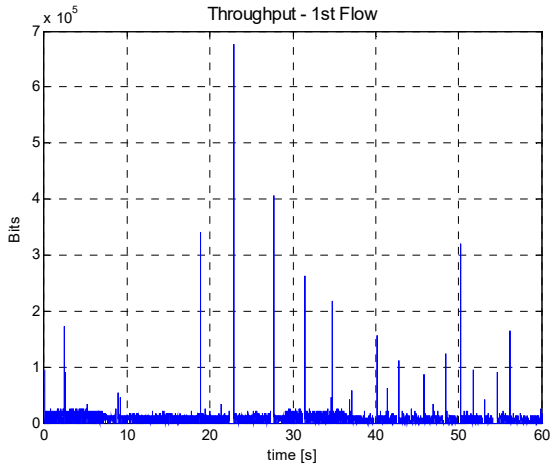


Fig.5. The throughput of the receiver side.

```
------------------------------------------------
Flow number: 1
From 10.0.0.28:44706
To    10.0.0.3:1003
------------------------------------------------
Total time                =     60.644229 s
Total packets             =         62730
Minimum delay             =      0.053149 s
Maximum delay             =      8.015104 s
Average delay             =      0.539471 s
Average jitter            =      0.003727 s
Delay standard deviation  =      0.193957 s
Bytes received            =      32117760
Average bitrate           =   4236.876027 Kbit/s
Average packet rate       =   1034.393561 pkt/s
Packets dropped           =        536169 (89.53%)
Average loss-burst size   =     26.118911 pkt
------------------------------------------------
```

Fig.6. Summary results of flow with constant receiver rate 10000 packets per second at the receiver side.

The receiver log file can also be obtained. The summary results of the first flow with constant receiver rate of 10000 packets per second are shown in Figure 6. It can be easily noticed that the dropped packets are about 90%. The minimum, maximum and average delays are also shown. Figure 7 shows the delay of the first flow.

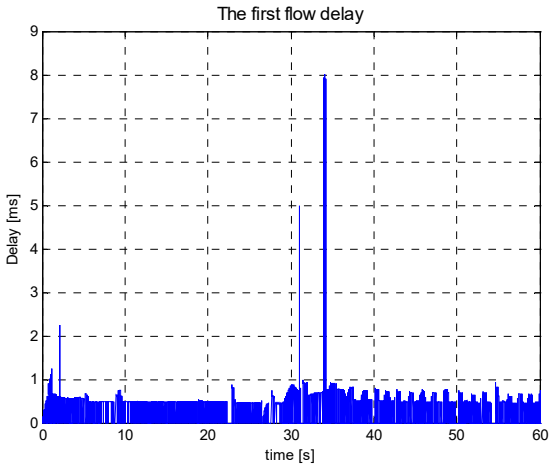


Fig.7. The delay of the first flow.

As shown in the summary log and also in the plotted delay diagram (Figure 7), the delay has a pick value during the 33th second of the measurement. And the value of this delay is about 8 ms. The average delay is less than 1 ms – it is about 0,5 ms. The delays of other flows are shown in the next two figures (Figure 8 and 9). The delays of the second and third flows have bigger maximum values. The average values are also bigger than the values for the first flow. All delays are obtained in the receiver side since the sender transmits flows without adding delay. At the sender side only the latency can be found.

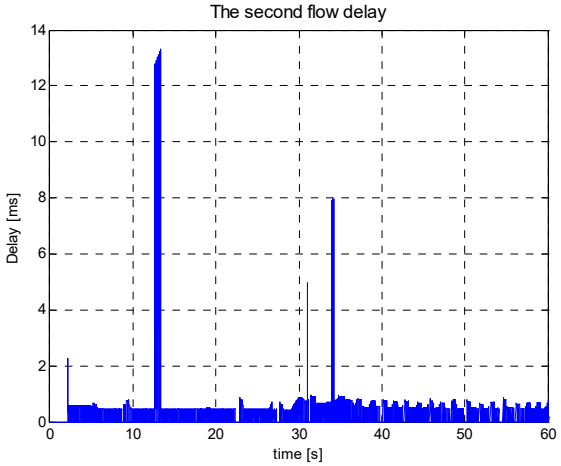


Fig.8. The delay of the second flow.

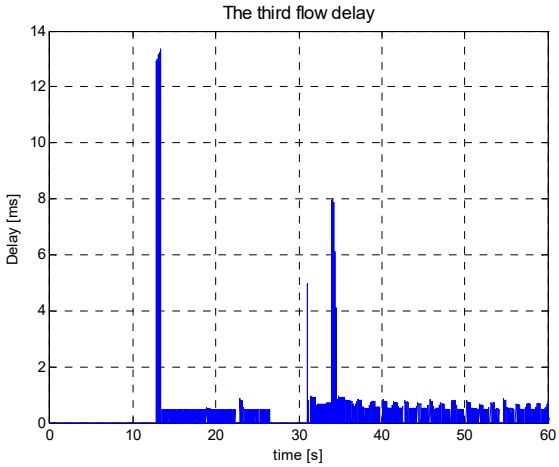


Fig.9. The delay of the third flow.

Summary

In summary the proposed platform is graphically represented in Figure 10. By using this platform user can evaluate, assess and analyse different performance metrics like delays, jitter, packet loss, sending and receiving bitrates of real-world network topologies. The simplicity to parse nearly every network to a commodity computer reveals new possibilities to test several aspects of SDN behaviour.

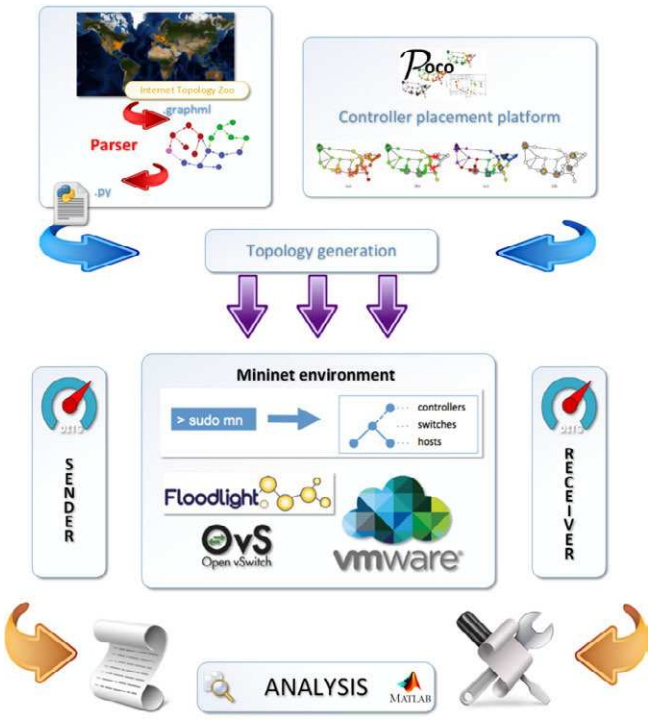


Fig.10. Block diagram of the proposed platform for evaluation, assessment and analysis of SDN control plane performance.

In conclusion, this platform for evaluation of the network performance can give accurate assessment of how the controller placement affects the network performance. The authors are determined to continue their work and to carry out more thorough experiments, which will be presented in future publications. There are some factors which influence the controller placement process. These factors can relate to failure tolerance or controller imbalance. For feature work, the authors plan to carry out experiments with scenarios, where the controller failure occurs in a network with more than one controller.

ACKNOWLEDGEMENTS

The work presented in this paper is completed as partial fulfilment of Project FNI-16-FEEA-04 "Study on the impact of the controller location in the management plane on the performance of the software-defined networks. financed under the Scientific Fund of the University of Ruse.

REFERENCES

[1] Heller, B., R. Sherwood, and N. McKeown. The controller placement problem" in Proceedings of the first workshop on Hot topics in software defined networks, ser. HotSDN.

[2] Hock, D., S. Gebert, M. Hartmann, T. Zinner, P. Tran-Gia. POCO-Framework for Pareto-Optimal Resilient Controller Placement in SDN-based Core Networks. Network Operations and Management Symposium (NOMS), 2014 IEEE, May 5-9, 2014, Krakow, Poland

[3] Y. Hu, W. Wang, X. Gong, X Que, S. Cheng. On the placement of controllers in software-defined networks. The Journal of Chine University of Posts and Telecommunications, vol.18, pp 92-97,171, Oct., 2012.

[4] http://mininet.org/

[5] Knight, S., H. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The Internet Topology Zoo. IEEE Journal on Selected Areas in Communications, vol. 29, no. 9, 2011, pp. 1765–1775.

[6] Großmann, M. and S. J. Schuberth. Auto-Mininet: Assessing the Internet Topology Zoo in a Software-Defined Network Emulator. OttoFriedrich University, Tech. Rep., 2013.

[7] Borcoci, Eugen, et al. On multi-controller placement optimization in software defined networking-based wans. ICN 2015 (2015): 273.

[8] Open vSwitch. May 2013. [Online]. Available: http://openvswitch.org

[9] Project floodlight. May 2013. [Online]. Available: http://www.projectfloodlight.org/floodlight

[10] Tootoonchian, Amin, et al. On Controller Performance in Software-Defined Networks. Presented as part of the 2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services. 2012.

[11] Shalimov, A., et al. Advanced study of SDN/OpenFlow controllers. Proceedings of the 9th central & eastern european software engineering conference in russia. ACM, 2013.

[12] Avallone, S. S. Guadagno, D. Emma, A. Pescape, and G. Ventre. DITG distributed Internet traffic generator. in Proceedings of the First International Conference on the Quantitative Evaluation of Systems. QEST 2004, 2004, pp. 316–317.

[13] Botta, A., et al. D-ITG 2.8. 1 Manual. Italia: University of Napoli Federico II, 2013.

[14] Matlab. May 2013. [Online]. Available: http://www.mathworks.de/ products/matlab

Diyana Kyuchukova *- PhD Student at the department of "Telecommunication" at the University of Ruse "Angel Kanchev". Her current scientific interests are in the field of communication networks, new generation network architectures, network virtualization technologies.*

tel.: +359 82 888 817 e-mail: dkyuchukova@uni-ruse.bg

Assoc. Prof. Georgi V. Hristov - *member of the department of "Telecommunication" at the University of Ruse "Angel Kanchev". His current scientific interests include communication networks, new generation network architectures, network virtualization technologies, unmanned aerial vehicles, 3D visualization and printing technologies and robotics.*

tel.: +359 82 888 663 e-mail: ghristov@uni-ruse.bg

Received on: 31.10.2016