Distributed Mininet Placement Algorithm for Fat-Tree Topologies

Philippos Isaia, Lin Guan
Department of Computer Science
Loughborough University
Loughborough
LE11 3TU, UK
Email: {p.isaia, l.guan}@lboro.ac.uk

***Abstract*—Distributed Mininet implementations have been extensively used in order to overcome Mininet's scalability issues. Even though they have achieved a high level of success, they still have problems and can face bottlenecks due to the insufficient placement techniques. This paper proposes a new placement algorithm for distributed Mininet emulations with optimisation for Fat-Tree topologies. The proposed algorithm overcomes possible bottlenecks that can appear in emulations due to uneven distribution of computing resources or physical links. In order to distribute the emulation experiment evenly, the proposed algorithm assigns weights to each available machine as well as the communication links depending on their capabilities. Also, it performs a code analysis and assigns weights to the emulated topology and then places them accordingly. Some noticeable results of the proposed algorithm are the decrease in packet losses and jitter by up to 86% and 68% respectively. Finally, it has achieved up to 87% reduction in the standard deviation between CPU usage readings of experimental workers.**

I. Introduction

Software-defined Networking (SDN) [1] is a computer networking framework based on the decoupling of the control and data planes, allowing direct programmability. The control plane is abstracted and placed into a central entity called the SDN controller whereas the data plane stays within the networking device. A controller-to-switch link is created that is used for communication between the two entities. Several communication protocols have been developed to allow the exchange of information between the switch and the controller, but the most popular and well maintained is OpenFlow [2]. Other known communication protocols include OpFlex [3], NETCONF [4] and OpenDaylight Lisp [5]. Once communication is achieved, the role of the controller is to inspect data packets and create rules that are then added to the network device flow table. SDN research provided numerous new ideas that solve many networking problems and allow networks to evolve and meet modern demands.

Research proposals, as well as production network configurations and algorithms, need to be validated with the use of simulators, emulators as well as actual testbeds. Simulators include NS-3 [6] which has restrictions due to not utilising the controller-to-switch communication, as well as fs-sdn [7] which promises scalability advantages over its competitors. Finally, there is OMNeT++ [8] which has added OpenFlow support [9], through the use of the INET framework [10]. Testbeds vary from production equipment to custom solutions such as the Pantou [11] project that uses the open source firmware OpenWrt [12] to convert low profile home routers into OpenFlow-enabled switches. NetFPGA [13], is an open source platform that has a number of community developed libraries [14], [15] with added support for OpenFlow experimentation.

Finally, there are emulators, that includes EstiNet 9.0 [16] which acts both as an emulator and a simulator, as well as Mininet [17], the dominant force in the area of SDN experimentation. Being the dominant force, Mininet comes with a huge list of capabilities, which are actively maintained and extended by the community. The most important drawback of Mininet is the fact that being an emulator, is hungry for resources such as processing power. This poses some serious problems when it comes to scalability [18], [19], [20], [21]. To overcome scalability issues, virtual-time enabled Mininet (VT-Mininet) [22], as well as distributed solutions, have been proposed. Distributed solutions, split the load of Mininet into several physical or virtual machines to harness more computing resources. Such solutions include Distributed OpenFlow Testbed (DOT) [23], [24], Mininet Cluster [25] as well as MaxiNet [26]. The problem with these type of solutions is that they use existing placement algorithms to spread the load across many machines. Such methodology can cause bottlenecks in an experimental scenario due to reasons such as link capacity limitations, adding delays that should not be present or even distributing resource hungry parts of the experiment into not powerful machines.

This paper proposes a new placement algorithm for distributed Mininet implementations. The proposed algorithm dynamically analyses the experimental topology as well as the devices used by the distributed Mininet implementation. After the analysis, it distributes the topology components in such a way that it limits the chances of a bottleneck or reductions in the experimental performance.

The remainder of this paper is organised as follows; Section II introduces existing distributed Mininet implementations as well as an analysis of existing placement algorithms. Section III presents the proposed placement algorithm whereas

978-1-5090-6501-1/17/$31.00 © 2017 IEEE

Section IV describes the setup and the way the tests have been carried out. Section V presents a thorough analysis of the experimental results whereas Section VI provides the conclusions.

II. Background

Mininet is an open-source emulator designed to emulate networks for research and experimentation. It uses Linux network namespaces to quickly create lightweight virtual nodes allowing the users to execute real code as well as standard Linux network applications on each of the virtual nodes created. This allows the use of the exact same semantics as hardware OpenFlow switches. In addition, the exact same Application Programming Interface (API) present in hardware switches is available. In theory, any scenario tested in Mininet can be deployed in a production network without any further modification. On the other hand, the fact that each virtual node is an actual Linux namespace, it means that Mininet has serious scalability issues. In a simulation, having limited resources will cause the experiment to take a greater amount of time until it returns the results. But in emulation, limited resources not only delay the experiment but also affect the results. To enhance the performance issues caused by the Linux lightweight virtualization methodology, Mininet-HiFi [27] has been introduced. Mininet-HiFi enforces performance isolation and resource provisioning, but it still lacks large scale emulation performance and scalability.

To extend Mininet's scalability, several distributed solutions have been proposed and implemented. DOT introduced several features that solve some of Mininet's issues but is not actively maintained and it does not support the latest Mininet implementations. The developers on Mininet have added Mininet Cluster prototype which tackles some of the issues, but the development stayed in the prototype phase, with no timeline for the final release. Mininet CE [28] acts as a control layer that combines instances of Mininet forming a larger cluster that can be used for large scale emulation. A slightly different approach has been taken by SDN Cloud DC [29] which introduces a number of new modules that enhance both Mininet as well as POX [30] in order to create an SDN based Data Center that can enable large scale experimentation. The best solution to several Mininet problems came in the MaxiNet project. MaxiNet is an actively maintained project which introduces several new features missing from Mininet, such as the resource monitoring, Docker container support, as well as time dilation [31]. With time dilation, MaxiNet solves another resource related Mininet limitation.

The only drawback of MaxiNet is the fact that even though it can spread the virtual nodes evenly across virtual or physical machines (workers), it has no notion of the workers performance as well as the network performance. Therefore, a resource hungry part of the network can end up in a worker with limited resources and link capacity, thus becoming the bottleneck of the network. With the use of METIS [32] graph partitioning library, MaxiNet splits the emulated network into x equal weight partitions where x is the number of workers available. The weight of each partition comes from two factors. The emulated link bandwidth, as well as the number of links each emulated node has. To optimise the network, it uses minimal edge cut and avoids limited physical links. MaxiNet allows users to specify where each node should be placed, but in a large automatically created topology it is very hard for the user to assign each node to a worker manually.

Mininet Cluster, on the other hand, comes with a variety of placement algorithms such as *RandomPlacer* which places nodes randomly or *RoundRobinPlacer* which it just places nodes equally around all workers. Also, it has *SwitchBinPlacer* which places switches into evenly-sized bins around each worker and *HostSwitchBinPlacer* which places switches and hosts into evenly-sized bins and places them around to each worker.

All these placement algorithms used both in MaxiNet and Mininet Cluster have not specifically designed to examine the network components and the resources they need. Thus a lot of them end up having several cross-server links that force physical links to become bottlenecks in an experimentation scenario. Also, they do not examine the worker capabilities; thus they may place a high demanding bin into a limited resource worker, causing inaccuracies in the final results. Today's data center networks follow fat-tree topologies [33], [34], [35] which in the case of the placement algorithms available in both MaxiNet and Mininet Cluster, a lot of cross-server links will be created, and the risk of bottlenecks will be increased.

III. Proposed Placement Algorithm

Designing a placement algorithm for OpenFlow emulations has to fulfil several requirements in order to avoid negative effects on the emulation results as shown in Section II. In the proposed algorithm, the following requirements were taken into consideration:

- Workers and links capabilities awareness
- Worker resources should be close to emulation resources
- Decrease the number of cross-server links
- Notify the user in case of insufficient resources or possible problems with workers
- Fat-Tree topology optimisation

For the algorithm to achieve the requirements listed above, a set of operations that perform individual tasks has been created. The initial operation is a static code analysis module (as shown in Figure 1) which examines the emulation code written by the user to get a view of the experiment that is about to be implemented. The module examines the code for known Mininet API "keywords" related to the topology components. Using the components found as well as their characteristics a weights table is created, and each network component is assigned a weight relative to its characteristics. These characteristics include the traffic that each component has to handle, the number of nodes connected to each component (if the component is a switch) as well as the number of links connected to the component. Finally, it gives weight to each emulated link using the maximum amount of traffic the link

might face during the emulation. To find the volume of traffic, the module searches for common traffic generation tools code such as iPerf [36] and Ostinato [37]. If the emulation uses traffic generation techniques that are not known to the module, then it uses the link capacity specified in code by the user as a weight indicator. If none of the two is provided, then it finds the number of nodes connected to the link and give weights accordingly.

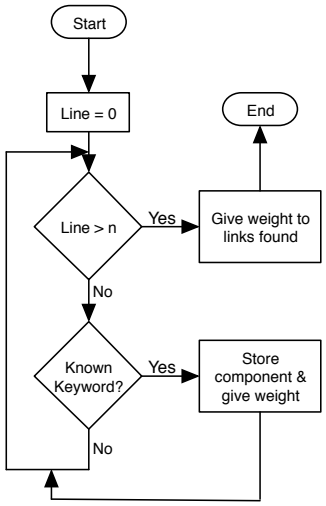

Fig. 1: Module 1 - Static Code Analysis

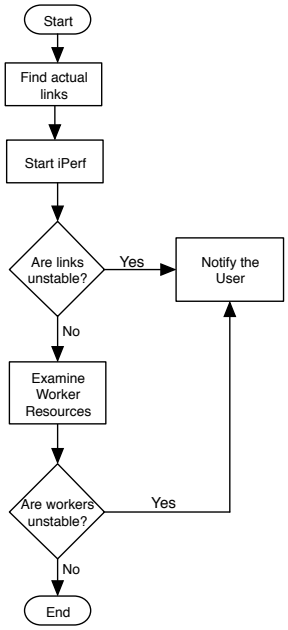

Fig. 2: Module 2 - Link Capacity

The next operation measures the capacity of the actual links connecting the workers using iPerf (as shown in Figure 2), and assigns relative weights to each of those links. If the link capacity shows signs of instability, which can be caused by traffic travelling through the link prior to the experiment or due to hardware malfunction, then the user gets notified. The same happens if the link has a high percentage of packet losses. Such small details can affect Mininet experiments, that is why users have to get notified and either change the experimental platform or agree to proceed with the existing one. In addition to link capacity, the operation examines the resources available to each worker. These resources include CPU power as well as RAM availability. In order for Mininet to run smoothly and not cause any effects on the final results, the CPU usage had to be in a "calm" state, and the RAM has to be sufficient to accommodate its memory needs. The algorithm can spot possible limitations that the workers might have, and it will notify the user. In such case, it is again up to the user to make alterations or agree to proceed.

The final operation of the algorithm creates bins that match the individual workers weight, with a limited number of cross-server links. Due to the fact that a lot of production networks use the Fat-Tree or variation of Fat-Tree topology, the algorithm is optimised for use in such topologies. Therefore, one of the goals that always tries to fulfil is to never split a pod apart if possible. The resulting bins are not meant to be of equal weight, but they should match the weight of each worker. If the algorithm suspects that the topology might face problems due to limited resources, it will notify the user before running the emulation. Having said that, the algorithm will work with any topology, but it has been tested and specifically optimised for Fat-Tree topologies.

IV. Performance Test

To examine the proposed algorithm, a fat-tree experimental topology (Figure 3) was used for both scenarios as well as three physical machine workers (Figure 4). In each of the scenarios, the resources available by the workers or the amount of traffic present in the experimental topology was changed. This allowed testing the functionality of some of the individual modules of the proposed placement algorithm. All of the scenarios were tested in MaxiNet, first with MaxiNet's default placement algorithm (SwitchBinPlacer) as a control experiment and a basis to compare with, and then with the proposed algorithm. All of the experiments were repeated 30 times to eliminate the experimental error. Furthermore, throughout the duration of each experiment, the workers performance was monitored for non-experiment related issues. This is due to the fact that in emulations, non-experiment related processes can use resources which can affect the experimental results.

A. Scenario 1

Scenario 1 purpose was to examine if the proposed algorithm gives the correct weights to both components and links, and assigns them to the appropriate physical workers. In order to achieve that a variety of traffic bandwidth was used in the experimental topology as well as different resource capabilities in physical workers and links (summarised in Table I). In the experimental topology, *Pod 1* had 2Gbps of network traffic travelling within the Pod (intra-pod). No traffic was travelling from *Pod 1* to any other pod. The rest of the pods had no

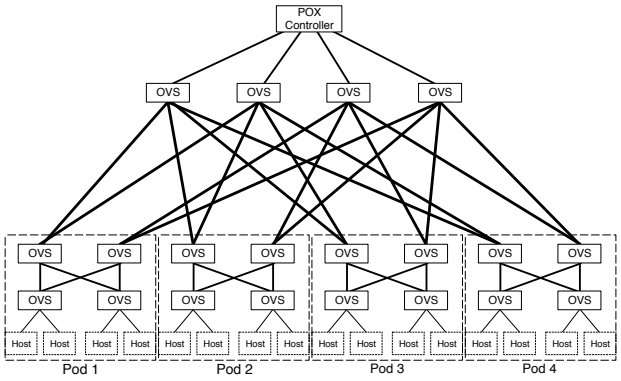

Fig. 3: Experimental Topology

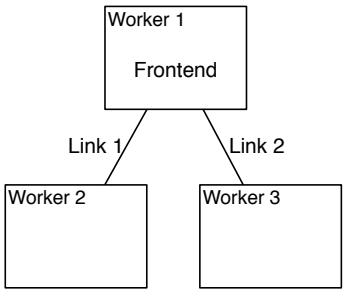

Fig. 4: Workers

intra-pod traffic, but *Pod 2* was exchanging 500Mbps traffic with *Pod 3* and *Pod 4* was sending 250Mbps traffic to *Pod 2* and 250Mbps to *Pod 3*. Physical *Link 1* had a 1Gbps capacity and *Link 2* only 600Mbps. Finally, *Workers 1* and *3* had 8GB of RAM, 4 cores at 2.4GHz and 32GB of hard disk whereas *Worker 2* had 16GB RAM, 8 cores at 2.4Ghz and 50Gb of hard disk.

Component	Characteristics
Pod 1	2Gbps intra-pod traffic
Pod 2	500Mbps to Pod 3
Pod 3	500Mbps to Pod 2
Pod 4	250Mbps to Pods 2 & 3
Link 1	1Gbps Capacity
Link 2	600Mbps Capacity
Worker 1	4 Cores at 2.4GHz, 8GB RAM, 32GB HDD
Worker 2	8 Cores at 2.4GHz, 16GB RAM, 50GB HDD
Worker 3	4 Cores at 2.4GHz, 8GB RAM, 32GB HDD

TABLE I: Scenario 1 Characteristics

B. Scenario 2

Scenario 2 purpose was to examine if the proposed algorithm will assign the components of the emulated scenario accordingly in order to avoid having the physical links becoming the bottleneck of the experiment. That is why the characteristics of both the emulated and physical components (summarised in Table II) were very closely chosen in order to cause a bottleneck if any component is misplaced. In the experimental topology, *Pod 1* was exchanging 2Gbps of traffic with *Pod 2*. *Pod 3* was exchanging 1Gbps of traffic with *Pod 4*, and it also had 1Gbps of intra-pod traffic. Similarly, *Pod 4* had 1Gbps of intra-pod traffic. Physical links *Link 1* & *Link 2* had a 1Gbps capacity. Finally, *Worker 1* had 16GB RAM, 8 cores at 2.4Ghz and 50Gb of hard disk, whereas *Workers 2* & *3* had 8GB of RAM, 4 cores at 2.4GHz and 32GB of hard disk.

Component	Characteristics
Pod 1	2Gbps to Pod 2
Pod 2	2Gbps to Pod 1
Pod 3	1Gbps to Pod 4 & 1Gbps intra-pod traffic
Pod 4	1Gbps to Pod 3 & 1Gbps intra-pod traffic
Link 1	1Gbps Capacity
Link 2	1Gbps Capacity
Worker 1	8 Cores at 2.4GHz, 16GB RAM, 50GB HDD
Worker 2	4 Cores at 2.4GHz, 8GB RAM, 32GB HDD
Worker 3	4 Cores at 2.4GHz, 8GB RAM, 32GB HDD

TABLE II: Scenario 2 Characteristics

V. Analysis

The experimental scenarios gave some very important results which indicate how well the proposed algorithm performs compared to the default MaxiNet placement algorithm (Results are summarised in Table III). First of all, the setup time for both scenarios when using the proposed algorithm is increased. This is an expected result since the proposed algorithm does not immediately start placing components around. It goes through the operations discussed in Section III in order to perform its weight calculations, then find the most appropriate workers and links and then start placing the topology components to the relevant workers. On the other hand, the teardown time is increased, which is an unexpected result since in the proposed algorithm experiments, the teardown technique used in the MiniNet's default. The most probable cause for this increase is the fact that MaxiNet stops hosts and switches in order. Since the proposed algorithm does not not spread the topology in order, but by weight, that means, during teardown MaxiNet has to go around the workers several times until all the components are stopped. Except from these two parameters, the rest of the parameters favour the proposed algorithm. In Scenario 1, the proposed algorithm decreased the packet loss by 81.8%[1] (from 31.61% to 5.73%) as well as the jitter by 64.3% (from 13.74% to 4.91%). Similarly, in Scenario 2, packet losses were reduced by 86.4% (from 53.12% to 7.25%) and jitter by 68.3% (from 19.54% to 6.19%).

The best indication of the work done by the proposed algorithm comes from both the CPU as well as the RAM usage readings. Even though the average value for CPU usage is very close for both algorithms, the standard deviation of the reading indicates that the proposed algorithm has utilised all of the workers CPUs almost equally. Since the average reading consists of three CPU readings, a high value of standard deviation indicates that some of the CPUs are underutilised and some are overutilised. Therefore, the emulated components are

[1]Percentage Change Calculated by $((V_2 - V_1)/V_1) * 100$

not assigned the best possible way. The same outcome happens with RAM where in some workers is underutilised or in some cases overutilised reaching up to 100%. Reaching 100% of RAM or CPU usage in emulation means that the workers do not have enough resources to run smoothly the experimental scenario, which will increase the number of packet losses or the delays in packet travel times. This is true in both scenarios since both jitter and packet loss percentages are significantly higher in the default MaxiNet placement algorithm compared to the proposed algorithm.

On close inspection of network traffic, Maxinet's default "SwitchBinPlacer" algorithm placed both scenarios the exact same way, ignoring the differences in traffic and link capacities. In both scenarios, *Pod 1* was assigned to *Worker 1*, *Pod 2* to *Worker 2* and *Pod 3* & *Pod 4* to *Worker 3*. This is the main reason that both jitter and packet loss reach high values. The default algorithm placed some of the high demanding pods into low resource workers.

Parameter	**Maxinet Default**		**Proposed Algorithm**	
	Sc. 1	**Sc. 2**	**Sc.1**	**Sc.2**
Packet Loss (%)	31.61	53.12	5.73	7.25
Jitter (ms)	13.74	19.54	4.91	6.19
Setup Time (s)	35.30	37.05	47.42	55.11
Teardown Time (s)	12.56	14.54	13.42	17.34
CPU Usage (%)	86.67 σ=23.09	88.33 σ=11.55	88.33 σ=2.89	91.42 σ=2.47
RAM Usage (%)	68.11 σ=24.43	81.53 σ=13.86	83.67 σ=7.51	85.63 σ=1.53

TABLE III: Experimental Results

VI. Conclusion

In this paper, we proposed a new placement algorithm for distributed Mininet network emulators. The proposed algorithm assigns weights to various components present in an emulated scenario such as hosts, switches, links as well as the traffic that will be present in the emulation. It then assigns weights to the available workers depending on their resources (CPU, RAM) as well as the links that connect them together. Finally, matching the emulated components weights with workers and links weights, it assigns each emulated component to the most appropriate worker. The algorithm is optimised for fat-tree topologies, in such a way that it does its best not to break apart pods especially if there is an indication of increased intra-pod traffic. The proposed algorithm compared to MaxiNet's default placement algorithm manages to decrease packet losses by up to 86% and jitter by up to 68%. Also, it has managed to perform a better workers utilisation, indicated by the CPU usage standard deviation which was decreased by up to 87%.

References

[1] O. N. Fundation, "Software-defined networking: The new norm for networks," *ONF White Paper*, 2012.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[3] "Opflex: An open policy protocol white paper." http://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-731302.html. Accessed: 26-06-2017.

[4] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network Configuration Protocol (NETCONF)." RFC 6241 (Proposed Standard), June 2011. Updated by RFC 7803.

[5] "Opendaylight lisp flow mapping." https://wiki.opendaylight.org/view/OpenDaylight_Lisp_Flow_Mapping. Accessed: 26-06-2017.

[6] T. Henderson, M. Lacage, G. Riley, M. Watrous, G. Carneiro, T. Pecorella, and others., "Network Simulator 3." https://www.nsnam.org. Accessed: 07-10-2015.

[7] M. Gupta, J. Sommers, and P. Barford, "Fast, accurate simulation for SDN prototyping," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 31–36, ACM, 2013.

[8] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, p. 60, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[9] D. Klein and M. Jarschel, "An OpenFlow extension for the OMNeT++ INET framework," in *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, pp. 322–329, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2013.

[10] A. Varga, R. Hornig, B. Seregi, L. Meszaros, and Z. Bojthe, "INET Framework." https://inet.omnetpp.org. Accessed: 26-06-2017.

[11] "Pantou." http://archive.openflow.org/wk/index.php/Pantou_:_OpenFlow_1.0_for_OpenWRT. Accessed: 07-10-2015.

[12] "OpenWrt." https://openwrt.org. Accessed: 07-10-2015.

[13] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo, "Netfpga–an open platform for gigabit-rate network switching and routing," in *Microelectronic Systems Education, 2007. MSE'07. IEEE International Conference on*, pp. 160–161, IEEE, 2007.

[14] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown, "Implementing an openflow switch on the netfpga platform," in *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pp. 1–9, ACM, 2008.

[15] S. H. Min, Y. Choi, N. Kim, W. Kim, O. Kwon, B. Kim, J. Lee, D. Kim, J. Kim, and H. Song, "Implementation of a programmable service composition network using netfpga-based openflow switches," in *1st ASIA NetFPGA Developers workshop, Korea*, 2010.

[16] EstiNet Technologies Inc, "EstiNet." http://www.estinet.com. Accessed: 07-10-2015.

[17] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, p. 19, ACM, 2010.

[18] K. Kaur, J. Singh, and N. S. Ghumman, "Mininet as software defined networking testing platform," in *International Conference on Communication, Computing & Systems (ICCCS)*, pp. 139–42, 2014.

[19] R. R. Fontes, S. Afzal, S. H. Brito, M. A. Santos, and C. E. Rothenberg, "Mininet-wifi: Emulating software-defined wireless networks," in *Network and Service Management (CNSM), 2015 11th International Conference on*, pp. 384–389, IEEE, 2015.

[20] F. Keti and S. Askar, "Emulation of software defined networks using mininet in different simulation environments," in *Intelligent Systems, Modelling and Simulation (ISMS), 2015 6th International Conference on*, pp. 205–210, IEEE, 2015.

[21] R. L. S. De Oliveira, A. A. Shinoda, C. M. Schweitzer, and L. R. Prete, "Using mininet for emulation and prototyping software-defined networks," in *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*, pp. 1–6, IEEE, 2014.

[22] J. Yan and D. Jin, "Vt-mininet: Virtual-time-enabled mininet for scalable and accurate software-define network emulation," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, p. 27, ACM, 2015.

[23] A. R. Roy, M. F. Bari, M. F. Zhani, R. Ahmed, and R. Boutaba, "Design and management of dot: A distributed openflow testbed," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pp. 1–9, IEEE, 2014.

[24] A. R. Roy, M. F. Bari, M. F. Zhani, R. Ahmed, and R. Boutaba, "Design and management of dot: A distributed openflow testbed," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, pp. 1–9, IEEE, 2014.

[25] B. Lantz and B. O'Connor, "A mininet-based virtual testbed for distributed sdn development," in *ACM SIGCOMM Computer Communication Review*, vol. 45, pp. 365–366, ACM, 2015.

[26] P. Wette, M. Draxler, A. Schwabe, F. Wallaschek, M. H. Zahraee, and H. Karl, "Maxinet: Distributed emulation of software-defined networks," in *Networking Conference, 2014 IFIP*, pp. 1–9, IEEE, 2014.

[27] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pp. 253–264, ACM, 2012.

[28] V. Antonenko and R. Smelyanskiy, "Global network modelling based on mininet approach.," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 145–146, ACM, 2013.

[29] J. Teixeira, G. Antichi, D. Adami, A. Del Chiaro, S. Giordano, and A. Santos, "Datacenter in a box: Test your sdn cloud-datacenter controller at home," in *Software Defined Networks (EWSDN), 2013 Second European Workshop on*, pp. 99–104, IEEE, 2013.

[30] M. McCauley, "POX," 2012. Accessed: 26-06-2017.

[31] D. Gupta, K. Yocum, M. McNett, A. C. Snoeren, A. Vahdat, and G. M. Voelker, "To infinity and beyond: time warped network emulation," in *Proceedings of the twentieth ACM symposium on Operating systems principles*, pp. 1–2, ACM, 2005.

[32] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[33] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 63–74, ACM, 2008.

[34] K. C. Webb, A. C. Snoeren, and K. Yocum, "Topology switching for data center networks.," *Hot-ICE*, vol. 11, 2011.

[35] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: a scalable and flexible data center network," in *ACM SIGCOMM computer communication review*, vol. 39, pp. 51–62, ACM, 2009.

[36] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs, "NLANR/DAST : Iperf - the TCP/UDP bandwidth measurement tool." http://sourceforge.net/projects/iperf/. Accessed: 26-06-2017.

[37] "Ostinato network traffic generator and analyzer." http://ostinato.org. Accessed: 26-06-2017.