# Towards Load Balancing in SDN-Networks During DDoS-attacks

Mikhail Belyaev
Dept. of Computer Systems and Software Engineering
St.Petersburg State Polytechnical University
Applied Research Center for Computer Networks
St.Petersburg, Russia
belyaev@kspt.icc.spbstu.ru

Svetlana Gaivoronski
Computational Mathematics and Cybernetics dept.
Moscow State University
Applied Research Center for Computer Networks
Moscow, Russia
s.gaivoronski@gmail.com

***Abstract*—Software Defined Networks (SDN) are becoming a trending technology in modern Internet. This technology helps to solve a significant number of well-known engineering problems in a effective and elegant way as they provide software-defined centralized network control. An SDN controller can be extended with application that effectively serve for concrete purposes and provide flexible management of network flows. This opens a great number of opportunities for a lot of network security problems such as maintaining of privileges in a proper way, splitting control and data planes, and attacks detection and mitigation. In this work we consider the opportunities of SDN for a "survival" mitigation during DDoS attacks, the load balancing problem. We propose two-level balancing solution in SDN networks, which includes traditional balancing between servers and load balancing between network devices as well. Experiments show that our solution increase "survival" time of a system during DDoS attack in times compared to existing balancing solution in SDN networks.**

***Keywords—load balancing; DDoS mitigation; SDN networks;***

## I. Introduction

Several years ago clouds made a computational revolution in IT world. Clouds also can be considered as logical step of computational evolution, starting from computations on single machines and going through clusters and grids. The idea behind the clouds is to migrate all computational, storage, network, even some specific services requirements to a service-oriented platform using virtual machines at data centers. This idea provides great opportunities for variety of consumers: from independed researchers, small and medium businesses to big organizations. The trend of migrating computations to the clouds continues to grow: according to recent statistics, about 60% of server workloads will be virtualized in 2013 [1], and totally cloud service market is forecast to grow to 18.5% in 2013 [2]. Nowadays, a plethora of big organizations extend their resources for cloud computing: Amazon EC2, Windows Azure, Google Engine, etc.

But every story has two sides. Wide spread of cloud technology leads to a number of interesting research problems, one of which is a load balancing among resources. Load balancing is a problem of resource distribution which guarantees that all available resources are used with maximum utilization. Despite the fact that load balancing problem in cloud considers different types of resources, in current work we focus on traffic balancing and network resources utilization. Network resources utilization typically includes L7 load balancing which is balancing between computing nodes or L4 load balancing which is balancing between network equipment. Existing approaches on load balancing typically describe L7 *or* L4 balancing, but not both.

Over the last few years we also can notice the wide adoption of very new conception in networking - programmable networks, or so-called Software Defined Networks. For example, that technology is already adopted by Google and a number of other significant players. In current work we decided to consider load balancing problem in case of SDN networks. SDN decouples control plane from data plane and gives the functionality of network and resources management to *controller* which can be programmable by user. That leads to such advantages as, for example, flexibility of flows management. In current work we ask ourselves a question: given the opportunities of SDN networks, can we reconsider the problem of load balancing? What can be improved and what can we do better? As a result, we propose load-balancing solution that examines only ip source and destination ip addresses. In common terms, our approach may be considered as L4 solution, but in fact in works at even lower level of OSI model.

One of the interesting application of load balancer that we consider in this work is the balancing in case of DDoS attacks. Speaking of DDoS attacks, load balancing is one of the significant mitigation survival techniques that typically increase maximum capacity of defended system.

The contribution of the paper can be summarized as following:

- We researched the opportunities to apply new concept of networks, SDN, for solving a well-known problem of load balancing during DDoS attacks. We found that ideas that stand behind SDN serve for that purpose natively;
- We proposed an efficient algorithm of two-level load balancing which includes typical load balancing between servers and balancing between network devices for SDN networks. Experiments show that our solution significantly increase survival time of the system under DDoS-attack;
- We implemented our algorithm as the part of DDoS

detection and mitigation system, which is able to detect the presence of attack and start different mitigation solutions automatically.

The paper is organized as following. In section II we provide brief overview of existing load balancing techniques. In section III we describe our load balancing solution for SDN networks, including overview of main differences between traditional networks and SDN networks, detailed description of the proposed algorithms and overview of implementation details. In section IV we provide evaluation results and in section V we summarize our work.

## II. Load Balancing Background

### A. L7 Balancing

L7 load balancing usually decouples on server cluster load balancing and server load balancing[3]. *Server cluster load balancing* is typically produced between computing nodes. The cluster load can be interpreted as client sessions or running applications. In the first case, TCP sessions are evenly distributed between servers and if some server is overloaded it prohibits new incoming connections. The redirection of already established connections are not usually performed due to the TCP session transfer and applications synchronization overheads. Commonly used techniques for distribution of client sessions between servers are using of DNS server or Network Address Translation (NAT). In the case of running applications servers are clustered by the type of their applications (database server, Web applications, etc.) and every client request is divided between several clusters [4]. The case of *server load balancing* occurs when the system tries to decrease the load of particular server. In our work we will not consider that case.

For both NAT and DNS distribution system should choose the most appropriate server for the next session. All balancing techniques typically fall into the classification, which includes static and dynamic approaches.

*Static* load balancing algorithms use a-priori information about the system state such that throughput or computation power, or any other performance features of selected nodes. Static approaches ignore current state of the nodes and their load. The main advantage of static approaches is an easy implementation, but the possibility of inefficient balancing is high. Static load balancing typically presented by following techniques [3], [5]: random selection; hash selection, where hash is generally considered as a function of client ip addresses; and (weighted) round-robin which may or may not consider performance of servers.

*Dynamic* load balancing distributes load between the servers during runtime. Such balancers typically monitor the load of every single server and when imbalance reaches specified throughput they start balancing algorithms [6]. The dynamic algorithms include such simple techniques as selection of server with fastest response time, server with the smallest number of connections, dynamic round-robin techniques and others. More sophisticated algorithms were observed and compared in [7]. It includes description of Honeybee Foraging Angorithm [9] that based on nature algorithm of honeybee self-organization; Biased Random Sampling [8] that uses random sampling of the system domain to achieve self-organization; ACCLB which is load balancing mechanism that base on Ant Colony and Complex network theory [10] and several others.

All described algorithms are suitable for different purposes, typically in Cloud computing environment, but none of them take into assumption the information about load of network devices themselves which is significantly important in case of SDN networks.

### B. L4 Balancing

L4 balancing is the balancing between network devices and equipment. In case of SDN network, algorithms describing L4 balancing focused at load distribution from different switches between controllers . Such problem is very significant in terms of reliability of SDN networks but do not has any connection with traffic load balancing.

R. Wang at al [11] describes the load balancing between servers solution under the OpenFlow protocol. That solution may occasionally affect alternate routes between entry point and selected server, but it is not studied in the paper.

The lack of load balancing between switches and alternate routes in SDN networks studies has driven us to cover this gap.

## III. Load balancing solution for SDN networks

### A. SDN background

Software-Defined Networking (SDN) is a rising approach to networking that allows administrators to manage network services through an abstraction of lower-level functionality. SDN is based on decomposition of the network traffic in two layers: *the control plane* and *the data plane*. The control plane is the distributed system that actually makes decisions on where and how the traffic on the data plane (the usual TCP-IP stack user data) is sent.

SDN networks provide a simple and robust way to access all levels of traffic management in the data layer of the network through a simple interface and using software-based mechanisms exclusively. When it comes to the task of load balancing, this approach does provide some pros and cons, including, but not limited to:

- A single point of failure (the SDN controller) that may become a bottleneck of the whole setup if used in a wrong way;
- There is a number of security concerns regarding the complex interactions between the control plane and the data plane using state-of-the-art SDN implementations;
- The ability to provide means for routing, splitting and controlling traffic streams on all layers of TCP-IP stack using a single software-based solution;
- Multiple ways to provide hardware duplication of used lines and connections without any need to make an account for it on the data layer;
- Ways to balance the traffic on OSI model layers 2 and 3 (as opposed to the usually employed levels 4 and 7).

In this work we are trying to employ the benefits of SDN while also avoiding the possible implications that can be caused by negative effects. The security of SDN networking, which is becoming a big concern the more this type of networking gets widespread and can have negative implications on the way load balancing works, is out of scope for this paper.

### B. Proposed solution

The proposed solution to traffic load balancing generally consists of two parts: the L7 load balancing using standard means (DNS/NAT balancing) of splitting the traffic streams between endpoint servers and the L4 load balancing to enable splitting the packets between different paths in the network. The first part does not consider the network which lies between entry point and servers as it is shown at figure 1. Such balancer operates in terms of entry point and servers. The second part takes into consideration all information about local network which includes the network topology (map), current load of channels between different switches, throughput of the channels and other significant parameters, as it shown at figure 2. This approach assumes that the SDN network between the server-level load balancer and the endpoint servers is structurally excessive in order to have different paths to the same servers to begin with. We do not describe the way to do L7 load balancing in this paper, which can be found at [3], [4], [7], [8]. It should be noted, however, that the approach we describe in this section does assume that the traffic streams are already distributed between endpoint servers as the algorithm does not provide any kind of distribution between those itself.

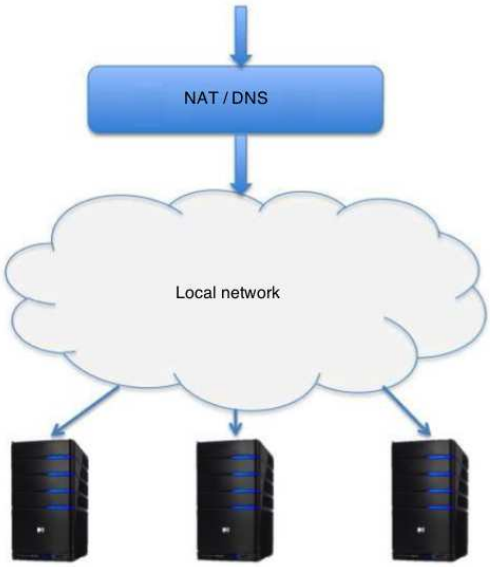


Fig. 1. First level of balancing.

The basic advantage of having these two levels of load balancing is that they can be made sufficiently independent from each other. As the inner balancing algorithm operates on layers 2-4 of the OSI model, it does not care for any of the peculiar properties the outer balancer may introduce, and vice versa. The fact that we use SDN control level to do the job of inner load balancing and do not introduce any additional modifications to the packets themselves, we can be sure that the

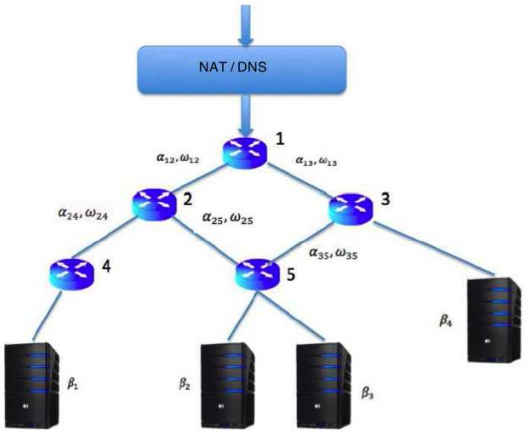


Fig. 2. Second level of balancing. $\alpha_{ij}$ stands for bandwidth of a channel between switches $i$ and $j$, $\omega_{ij}$ stands for current channel load

two levels of load balancing do not interact in any unintended way.

The algorithm itself is based on the fact that we can use the SDN switch-level flows to redirect traffic based on destination and source IP-address information. This allows for dividing the traffic between different routes in the network regardless of the packets' actual contents. The algorithm goes as follows:

1) Acquire the load and topology information for the network;
2) Override the routing for the network with static routing information acquired by using Bellman-Ford pathfinding algorithm;
3) Iteratively keep splitting (and reapplying) traffic paths for routes that are:
   - Overloaded;
   - Have alternate routes available.

   The splitting is done by using source IP address mask as packet distinguishers.

Let the network contain switches $1-N$. A *channel* between switches $i$ and $j$ will be addressed as $(i, j)$. We define the *bandwidth* of this channel as $\alpha_{ij}$ and the *current channel load* as $\omega_{ij}$. We say the the channel is ***overloaded*** if $\omega_{ij} + \epsilon \geq \alpha_{ij}$, where $\epsilon$ is a constant small load value parameter. In current implementation we define $\epsilon$ as an input parameter for the algorithm. The impact of this parameter and the range of its values has not been studied yet and is considered as a direction of futher work.

Let the endpoint servers be defined as $\beta_1 \dots \beta_K$. Bandwidth matrix $M_{maxload}$ is the matrix of size $N \times N$ containing all the bandwidth values $\alpha_{ij}$. Load matrix $M_{load}$ is the matrix of size $N \times N$ containing all the current load values $\omega_{ij}$. Matrix of available resources $M_{free}$ is defined as $M_{maxload} - M_{load}$.

*Phase 1* of the algorithm need to be executed before the need for load balancing arises (e. g. if we apply the approach to mitigate network attacks, we need to run it in a timed loop without other phases for as long as the attack *doesn't* begin to keep the topology and load information updated). Phase 1 introduces and updates the network load mask $M_{load}$, where

element $\omega_{ij}$ corresponds to the number of bytes coming from switch $i$ to switch $j$ during a single update period.

*Phase 2* is applied only once to override the default packet routing mechanisms and use statically defined routes we can later modify using network address masks. This is performed by running the standard Bellman-Ford algorithm on the whole network topology graph in order to acquire shortest paths from the network entry point to the endpoint servers.

*Phase 3* goes as follows. On the first iteration we build the current path table $T_{path}$ based on the path information acquired on Phase 2. $T_{path}$ is essentially a set of triples $\{ips_{src}, ip_{\beta_i}, path\}$ where each triple denotes a path $path$ from all addresses conforming to the address mask $ips_{src}$ to the address $ip_{\beta_i}$, which is the IP address of the server $\beta_i$. On the first iteration of phase 3 value of $ips_{src}$ for all the entries in the table is `0.0.0.0/0`, which is a wildcard accepting all the possible IP addresses. On the second and subsequent iterations, this table gets updated along with the corresponding network flows:

1) Update $M_{load}$ and $M_{free}$ with current load information from SDN switches;
2) Find the first overloaded link in $M_{load}$: the link $(i, j)$ such that $\omega_{ij} + \epsilon \geq \alpha_{ij}$;
3) Find the first path $r_q$ in $T_{path}$ such that it contains link $(i, j)$;
4) For the $ip_{\beta_i}$ part of $r_q$, find a new shortest path from entry to the server $\beta_i$ assuming that link $(i, j)$ is closed in current topology. If there is no such path, we should go back to 3 and find a new path for the same link. If there are no more paths containing this link, we should go back to 2 and select a new link. Let's call the new path $path_q$;
5) Calculate the maximum available additional load for $path_q$. For that, we look up every link in $path_q$ in $M_{free}$: $al = m_{i_{crit},j_{crit}} = min\,(m_{ij} : (i, j) \in path_q)$ and note $al$ and $(i_{crit}, j_{crit})$. If $al < \epsilon$, go back to 4 and find a new path that does not contain $(i_{crit}, j_{crit})$.
6) Try to calculate the new sets of masks $ips_{old}$ and $ips_{new}$ such that they divide all the address space of $ips_{src}$ into parts with coefficient $al/\omega_{ij}$. Remove the corresponding entry from $T_{path}$, insert all the entries $\{ips_{old\_k}, ip_{\beta_i}, path\}$ and $\{ips_{new\_k}, ip_{\beta_i}, path_q\}$ into $T_{path}$.
7) Commit the changes in $T_{path}$ to all the switches across $path$ and $path_q$.
8) Wait for the timeframe and go back to 1.

Of course, it is not generally possible to introduce a set of new network ip/mask pairs for a given one such that it divides all the address space denoted by it to a particular fraction, but it is possible to do it in a discreet manner, up to some number of bits in a network address. For example, if we do the division for 5 significant bits, we can divide the address space into parts that are multiples of $1/(2^5)$. For all practical purposes, this discreetness does not seem to introduce any significant effect on the behavior of the algorithm.

For example, given a ip/mask `9.0.0.0/8` and a factor of $1/3$, we do the following:

1) Introduce the 32 masks dividing the address space given into 32 equal pieces by adding 5 bits to the size of the mask (the ip/mask becoming `9.0.0.0/13`) and enumerating the now valid 5 bits in ip address with values from 0 to 31;
2) Divide the space of 32 address/mask pairs into 2 parts by the ratio: that is, putting first 21 pairs (`9.0.0.0/13`–`9.160.0.0/13`) into first part and last 11 pairs (`9.176.0.0/13`–`9.248.0.0/13`) into the second;
3) Collapse the pairs in each half that can be summarized using a pair with a smaller mask size (e.g. all pairs in (`9.0.0.0/13`–`9.112.0.0/13`) can be collapsed into (`9.0.0.0/9`).

For the given pair `9.0.0.0/8` the result will create 6 address/mask pairs. It can be easily shown that for any $N$ bits used as a discretion factor, any ip/mask pair will produce no more than $N + 1$ new ip/mask pairs in each iteration, thus the growing factor of introduced flows is constant. The ip address space is finite, so this process will always terminate.

It should be noted that we produce flow management that is based at source ip adresses due to the following reasons. First of all, after the phase 1 traffic is already distributed uniformly among available servers. Secondly, we want to distrubute attacking traffic among available routes as uniformly as possible. From that point of view, the worst case to the presented algorithm is the case of DoS attack with one attacking ip address.

### C. Implementation

The proposed approach was implemented as a part of a attack detection and mitigation system called `Callophrys`. The system aims at both identifying, detecting and mitigating DDoS attacks at early phases. It uses the Floodlight Openflow controller[12] for both gathering information from the SDN switches and applying the calculated paths and wildcards by deploying them to the switches. Both tasks are achieved through controller's REST API.

`Callophrys` is a distributed software system employing a number of asynchronous agents (actors[13]) communicating using immutable messages both between processes and machines and inside them. This model of computation allows for greater modularity and scalability of the whole system, but also introduces some difficulties for implementation of non-asynchronous algorithms, like the one introduced in the previous section.

The biggest difference for the procedural description of the algorithm found above is the fact that in an asynchronous system there is no need to wait for the next timeframe to come or for the other part of the system (i. e. the SDN controller) to send in the next portion of data to perform useful work. The asynchronous way of implementing the same algorithm is to identify the important events (in this case, keeping the topology and load information updated can be done independently from the rest of the algorithm) and perform actions when they happen. The load balancing algorithm is incapsulated into a single actor (whether it can be further decomposed to employ capabilities of the asynchronous computations is subject to further research) that handles the following kinds of messages:

- *Timeframe* message signaling that a timeframe is reached (sent by the program scheduler):
  1) Send a query message for the current network topology;
  2) Send a query message for the current load information.
- *Topology* message signaling that the topology information has changed (sent by Floodlight interface):
  1) Update the topology information.
- *Load* message signaling that the periodic load information is received (sent by Floodlight interface):
  1) Validate and apply the current load information;
  2) If the balancer is in active mode, start performing a single phase 3 iteration of the algorithm.
- *Alert message* signaling that the attack has started (sent by one of `Callophrys` detector programs):
  1) Turn on active mode;
  2) Force send a *timeframe* message to self.

An important property of an actor in the actor model is the fact that message handlers are never run concurrently. All the message-handling routines, other actions and receiving/sending can be done in separate threads, but the message handlers themselves are always run in an order and thus we don't need any kind of additional synchronization precautions. Thus this asynchronous implementation is very similar to the procedural one described in the previous section.

## IV. Evaluation

The prototype `Callophrys` system was evaluated using the Mininet[14] network simulator and Floodlight SDN controller. The simulation used a custom Mininet script for the network topology. The test topology configuration is shown at picture 3. The attacker nodes (signed with "A") are generating high traffic towards the target nodes (signed with "T") using the `iperf` tool[15], while the SDN switches (the rest of the nodes) try to balance the load between themselves and data links. For the purpose of this experiment, the links that are always overloaded (the links coming directly to target nodes, all the links coming to and from the entry node) are simulated as having infinite bandwidth, while all the other links in the setup have a thorough simulation with limited bandwidth and latency.

The process of evaluated using Floodlight and its built-in network-tracing mechanism. The preliminary experiments using this setup show that the proposed approach does balance the network load between switches, let the traffic take alternate roots and does not overload the switches with static flows.

In this setup, the time between the start of the attack and the full balance of traffic between available roots was from 10 to 60 seconds. Total number of flow rules generated is around 13000, while every single switch is subject to no more than 3000 different rules. Most of existing SDN switches can easily handle number of rules up to hundreds of thousands. A more thorough simulation using different topologies and testing on a real physical network is a subject of further work, as well as introducing more accurate tools to measure the effect of the algorithm.

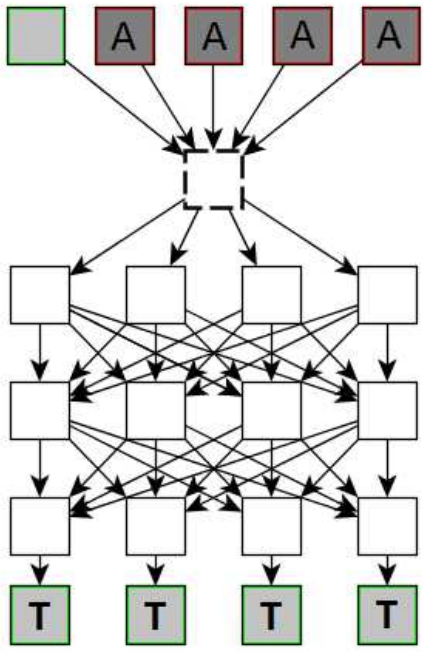


Fig. 3. The test network topology

## V. Conclusion

In this work we were focused on studying of possible impacts SDN networks could bring into traditional network security related problems. As DDoS attacks remain one of the most important security problems for a single hosts and data centers as well, we decided to consider DDoS-mitigation solution for SDN networks. DDoS mitigation solutions are typically divided into two classes: active techniques that include filtering of an attacking traffic and "survival" techniques that include increase of resources under attacks and effective load balancing.

We noticed that all existing load balancing solutions are based on a load balancing between endpoint resources, such as different servers in our case. Despite the fact that those techniques serve well for that purpose in traditional networks, SDN benefits help to increase survival time even more. Typical SDN controllers select alternative route between entry point and end point only when current route is not available. That works well and helps to remain network reliability. Nevertheless, during DDoS attack it does not solve problem at all as all attacking traffic will be forwarded to different route. There is high probability of new route to be overloaded as well. In our work we propose a solution for efficient traffic distribution between all alternative routes in a SDN network.

Experiments show that our solution helps to increase survival time of defended system during DDoS attack, thus it is effective as DDoS mitigation solution in SDN networks, but it can also be used as a general load balancing system.

## REFERENCES


[1] Cloud Hypermarket, *The Cloud Revolution*, http://www.cloudhypermarket.com/whatiscloud/CloudUptake.

[2] Garthner, Inc. *Gartner Says Worldwide Public Cloud Services Market to Total* $131 *Billion.*, https://www.gartner.com/newsroom/id/2352816

[3] Jasobanta Laha, Rabinarayan Satpathy , Kaustuva Dev. *Load Balancing Techniques: Major Challenges in Cloud Computing - A Systematic Review.* IJCSN International Journal of Computer Science and Network, Volume 3, Issue 1, February 2014

[4] Kaushik V. K., Sharma H. K., Gopalani D. *Load Balancing In Cloud-Computing Using High Level Fragmentation Of Dataset*

[5] P. Mohamed Shameem and R.S Shaji.*A Methodological Survey on Load Balancing Techniques in Cloud Computing.* International Journal of Engineering and Technology (IJET)

[6] Malik, S., *Dynamic Load Balancing in a Network of Workstation*, 95.515 Research Report, 19th November, 2000

[7] Rajoriya, Sheetanshu. *Load Balancing Techniques in Cloud Computing: An Overview.* International Journal of Science and Research 3.7 (2014).

[8] Randles, M., Lamb, D. and Taleb-Bendiab, A., *A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing*, 24th International Conference on Advanced Information Networking and Applications Workshops, 551-556, 2010

[9] Padhy, R. P., and Rao, P G. P., thesis entitled *Load balancing in cloud computing system*, Department of Computer Science and Engineering, National Institute of Technology, Rourkela, Orissa, India, May, 2011

[10] Zhang, Z. and Zhang, X., *A Load Balancing Mechanism Based on Ant Colony and Complex Network Theory in Open Cloud Computing Federation*, Proceedings of 2nd International Conference on Industrial Mechatronics and Automation (ICIMA), 240-243, May, 2010

[11] Richard Wang, Dana Butnariu, and Jennifer Rexford. *OpenFlow-Based Server Load Balancing GoneWild.* Hot-ICE11 Proceedings of the 11th USENIX conference on Hot topics. Vol 12

[12] Floodlight Openflow Controller, http://floodlight.openflowhub.org

[13] Hewitt, Carl. *The Actor Model*. MASSACHUSETTS INST OF TECH CAMBRIDGE, 1993.

[14] Lantz, Bob, Brandon Heller, and Nick McKeown. *A network in a laptop: rapid prototyping for software-defined networks*. Proc. of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. ACM, 2010.

[15] Schroder, Carla. *Measure Network Performance with* `iperf`. Enterprise Networking Planet, 2007.