Leveraging Net2Plan planning tool for network orchestration in OpenDaylight

Jose-Luis Izquierdo-Zaragoza, Angel Fernandez-Gambin, Jose-Juan Pedreno-Manresa, Pablo Pavon-Marino
Telematics Engineering Group, Department of Information Technologies and Communications
Universidad Politécnica de Cartagena
Plaza del Hospital, 1, 30202 Cartagena, Spain
Email: josel.izquierdo@upct.es

***Abstract*—The software-defined networking (SDN) paradigm brings unprecedented agility and flexibility to network operators. The usage of centralized software-based controllers simplifies operation and management and enables innovation in the control plane through network programmability. Besides, a controller may be operated via third-party applications using the so-called northbound application programming interfaces (APIs). In this paper, we report our preliminary work on the interaction between the northbound interface of the open-source industry-supported OpenDaylight controller and our Net2Plan tool. Net2Plan is an open-source network planning tool that is able to execute user-made algorithms. By connecting Net2Plan to OpenDaylight, users would be able to easily orchestrate their networks, e.g. defining flow routing, making use of built-in or user-defined algorithms, implemented in the Net2Plan framework. In this paper, we present some illustrative experiments, and discuss future challenges.**

Keywords—Software-defined networking; network orchestration; OpenFlow; OpenDaylight; Net2Plan; northbound API; open-source

I. INTRODUCTION

Historically, technology advances and novel architectures have supported the exponential growth of the Internet. In fact, the main concern for network operators and service providers has been to scale network capacity to accommodate increments in traffic volume [1].

Today, the major challenge is to provide a variety of services such as video on demand (VoD) or cloud computing. These services require networks being able to adapt dynamically to changes in traffic profiles and automate on-demand provisioning. To address these requirements, software-defined networking (SDN) has emerged as a disruptive technology.

SDN is a revolutionary approach to design and manage communication networks. The basic idea is to separate control and forwarding planes into software-based controllers and simple forwarding devices, respectively, with the control function being programmable. Therefore, operators get rid of vendor lock-in and have the ability to implement flexible networks that can be dynamically provisioned.

SDN is currently attracting a significant attention from academia and industry. A group of network operators, service providers, and vendors has created the Open Network Foundation, an industry-driven organization to promote SDN and to standardize the OpenFlow protocol [2], which enables communication between control and forwarding planes. Another example is the OpenDaylight controller [3], an extensible SDN controller developed under the umbrella of the Linux Foundation and supported by the industry.

Still, even though SDN is growing at a rapid pace, the networking community is focused on centralized control and network programmability, defining languages and protocols [4][5]. However, in our opinion, network orchestration is also an important issue. Our argument is that that networking primitives are already available (i.e. protocols and controllers), so it is time to solve real-world problems to create a positive feedback in the development process.

Actually, the main outcome of SDN is that several network configurations can be automated. Automation allows services to be provisioned quickly as well as in an efficient and scalable manner. In addition, almost any issue related to human intervention may be eliminated [6]. Hence, to effectively bridge the gap between traditional networking and SDN we need to put in value the benefits of this technology: (i) separation of the control plane from the data plane, (ii) a centralized controller and view of the network, (iii) open interfaces between devices in both planes, and (iv) programmability of the network by external applications.

In this work, we introduce Net2Plan [7], an open-source technology-agnostic network planning tool, as a framework to operate SDN networks. Net2Plan allows users to fast-prototype their own algorithms, or use the provided built-in ones, and evaluate their output designs using either automatic report generation or post-analysis simulation tools. We believe that connecting Net2Plan with an SDN controller like OpenDaylight may produce a powerful open-source stack to cover the SDN ecosystem.

The main contribution of the paper is to report a set of experiments we are carrying out to study the integration of Net2Plan with OpenDaylight in order to orchestrate an OpenFlow-based network. In this first approximation, we address how Net2Plan can be used to interact with OpenDaylight to complete the following processes. First, we present the functionality that permits Net2Plan retrieving topology and traffic information from a network managed by

This work was partially supported by the Spanish project grant TEC2010-21405-C02-02/TCM (CALM) and the FPU predoctoral fellowship program of the Spanish Ministry of Education, Culture and Sport (reference no. FPU12/04571). It was also developed in the framework of project "Programa de Ayudas a Grupos de Excelencia de la Región de Murcia" funded by F. Séneca (Plan Regional de Ciencia y Tecnología 2007/2010).

an OpenDaylight controller. Then, we use Net2Plan to estimate the end-to-end traffic matrix from the information obtained in the previous step. With the estimated information, we use a Net2Plan algorithm to compute the routing that optimizes the link bandwidth utilizations. Finally, we illustrate how this optimized routing computed by Net2Plan can be applied into the network via OpenFlow rules, using the Net2Plan-OpenDaylight interface.

The presented procedures are an example of the multiplicity of analysis and network optimization processes that can be implemented in Net2Plan. Thanks to the connection to OpenDaylight northbound interface, the door is open to apply all the strength and flexibility of Net2Plan to orchestrate and optimize communication networks, controlled by OpenDaylight. We believe that the open-source nature of Net2Plan tool perfectly fits with Opendaylight, also an open-source initiative. In particular, the OpenDaylight connection plugin and all the algorithms used in this paper will be included in the next Net2Plan release (under completion at the time of writing, expected for July 2014).

The rest of the paper is organized as follows. Sections 2, 3 and 4 briefly describe OpenFlow, OpenDaylight and Net2Plan, respectively. Section 5 explains the Net2Plan-OpenDaylight interface. Section 6 reports our experiments. Section 7 describes the roadmap of Net2Plan into SDN field. Finally, Section 8 concludes the paper.

II. BACKGROUND

SDN was originally developed to facilitate innovation and enable programmable control of the data plane, so that the network is reduced to "simple" forwarding hardware and decision-making controller(s). Driven by this principle, OpenFlow [2] was developed to standardize information exchange between the two planes.

In the OpenFlow architecture, the forwarding device (or switch) contains a flow table (or several, from OpenFlow 1.1) and an abstraction layer that communicates with a controller via OpenFlow protocol through a secure channel. Using this protocol, controllers populate and manage flow tables. Each flow table contains flow entries, which determine how incoming packets are processed. Flow entries consist of match fields (i.e. ingress port, layer 2-4 header…), counters and a set of actions to apply to matching packets (i.e. forward to an output port).

OpenFlow describes two different management approaches to populate flow tables: proactive and reactive. Reactive mode is applied when incoming packets do not match any flow entry. Packets are sent to the controller, which installs rules back into the switch. Alternatively, the controller may install proactively flow entries into the switch according to user policies, before packets arrive. Reactive approach is a more powerful model because the controller can implement some kind of logic that cannot be represented into flow tables. In contrast, proactive mode yields to better performance because it avoids the extra latency on the first packet of the flow. In practice, applications may implement a combination of both, where some traffic is handled proactively and some is handled reactively. This hybrid mode would provide zero-latency forwarding for particular flows while still preserve fine-grained traffic control for the rest of the traffic.

Flow-based forwarding through OpenFlow consolidates functionalities of layer-specific hardware into a single device, which essentially involve packet forwarding based on a general definition of flow.

III. OPENDAYLIGHT

At this early stage of SDN development, most of the SDN controllers in the market are like "black-boxes", and the lack of interoperability is discouraging SDN adoption by organizations that run equipment from multiple vendors [8].

In this sense, the industry acknowledges the benefits of establishing an open, reference framework for programmability and control through an open-source SDN solution [3]. Also, mitigation of risks of early adoption and co-existence with existing infrastructure are also major concerns. In this context, the OpenDaylight project was created in early 2013.

OpenDaylight is an open-source project with a modular, pluggable, and flexible controller platform at its core. It is implemented in Java, so it can be installed on any platform that supports Java.

The controller exposes open northbound APIs which are used by applications. OpenDaylight supports the development of extensions as Java modules, using the OSGi framework, and REST for a web-based northbound API. The OSGi framework is used for applications that will run inside the controller, while the REST API is used for applications that run outside the controller. The network operation logic and algorithms reside in the applications (i.e. Net2Plan in our case). These applications use the controller to gather network information, run algorithms to perform some optimization or analytics tasks, and then use the controller to spread new rules in the network, if any.

The controller already provides a collection of pluggable modules (known as base network services) working out-of-the-box. Some examples are the topology discovery services or the statistics collector. In addition, platform oriented services and other extensions can also be inserted into the controller platform for enhanced SDN functionality.

On the other hand, the southbound interface provides support for multiple protocols (as separate plugins), e.g. OpenFlow 1.0/1.3, BGP-LS, PCEP… in a vendor-neutral environment.

Before introducing Net2Plan, we briefly describe the architecture under study. Fig. 1 helps us to illustrate it. Switches and hosts define the network topology. Dashed lines represent the control plane that connects every switch to OpenDaylight via the OpenFlow plugin. On the other side, Net2Plan is connected to the controller using the REST API in the northbound. Then, Net2Plan is able to retrieve some information from the controller (i.e. topology and link load statistics) and to take some actions (i.e. programming flow rules).

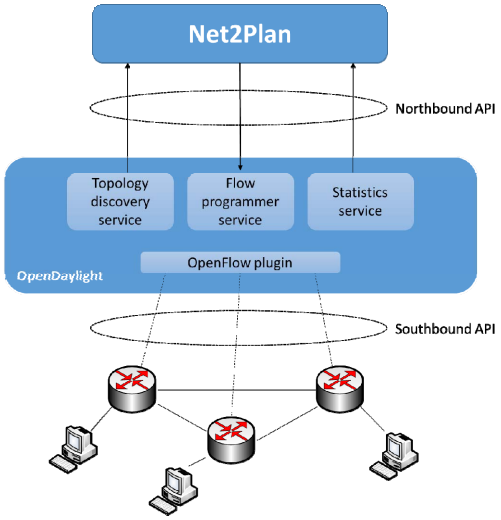

Fig. 1. Network architecture including topology, controller and Net2Plan.

IV. NET2PLAN

Net2Plan is a free and open-source Java tool devoted to the planning, optimization and evaluation of communication networks [7]. It was originally thought as a tool to assist the teaching of communication networks planning courses. Eventually, it has converted into a powerful network planning tool for the academia and industry [7], together with a growing repository of network planning resources.

Net2Plan is built on top of an abstract network representation, so-called network plan, based on seven abstract components: nodes, links, routes, traffic demands, protection segments, shared-risk groups and network layers. The network representation is technology-agnostic, thus Net2Plan can be adapted for planning networks in any technology. Technology-specific information can be introduced in the network representation via user-defined attributes attached to any of the abstract components mentioned above.

Current version of Net2Plan (0.2.3, March 2014, available for download in [9]) provides six different tools: offline network design, traffic matrix design, three post-analysis simulator (network resilience, connection-admission control, and time-varying traffic), and a reporting tool. For this work, we focus on the offline network design tool (see Fig. 2).

This tool is targeted to evaluate the network designs generated by built-in or user-defined offline network design algorithms, deciding on aspects such as the network topology, the traffic routing, link capacities, protection routes and so on. If needed, those algorithms based on constrained optimization formulations (i.e. ILPs) can be fast-prototyped using the open-source Java Optimization Modeler library (JOM), to interface from Java to a number of external solvers such as GPLK, CPLEX or IPOPT.

The word “offline” means that all the variables in the network design are supposed to be known deterministic values. For instance, in our study, this could be a snapshot of the network (topology and end-to-end average traffic), although in reality network conditions fluctuate along time.

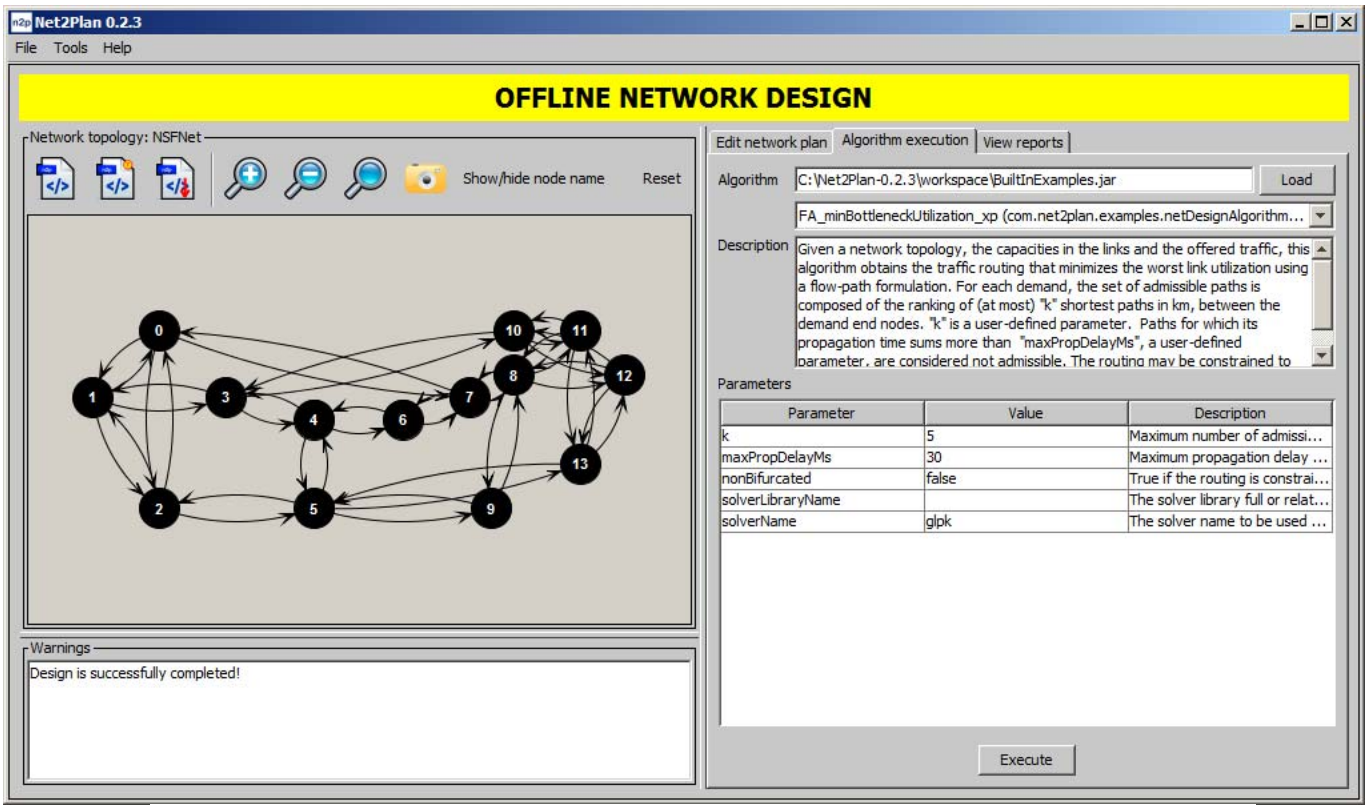

Fig. 2. Screenshot of the offline network design tool.

The workspace of the window is divided into three areas: input data, execution and reporting (right area), plot area (top-left area), and warning area (bottom-left area). In Net2Plan, network designs can be completed incrementally, modifying the current network design, that the user can inspect visually in the "Network Topology" panel, and the "Edit network plan" tab. When the offline design tool starts, the current network design is empty. Then, the user can modify it (i) using manual editions in the "Network Topology" panel or the "Edit network plan" tab, (ii) loading existing designs and/or traffics demands, (iii) applying offline design algorithms (that take the current network design, and return a modified design that becomes the current design).

As we will describe in the following section, this tool is used as baseline to deploy the Net2Plan-OpenDaylight interface.

V. Communicating Net2Plan with OpenDaylight

OpenDaylight supports interaction with third-party applications via OSGi bundles (or modules) as well as via REST API. The former requires the development of a Java package following certain conventions, and should be "installed" into OpenDaylight before using it. In contrast, the REST API provides a simple way to communicate via HTTP messages (as in a web-browser), and it can connect to OpenDaylight at any time during the network operation. While the OSGi bundle option is the most complete, since it allows bidirectional communication among many other functionalities, we prefer to use the REST API at this stage. The rationale behind this choice is that this work is just a proof-of-concept, and we prefer to avoid the complexity of OSGi for novice users. Development of OSGi bundles is left as further work.

As aforementioned, we use as a template the "offline network design" tool to implement our interface with OpenDaylight. This tool already includes almost any of the desired features: a canvas where showing the network topology, a panel for algorithm execution, and a panel for visualizing network information. Therefore, we only need to include the specific functionalities to connect with OpenDaylight.

When started, OpenDaylight runs a web-server in the local machine (default port 8080). On the one hand, if users connect to the controller via web-browser, they will get the web user interface (webUI) of OpenDaylight (see Fig. 3). The webUI shows the network topology (information about physical location is unknown, so users must manually position nodes), flow routing tables, statistics, and so on. On the other hand, if connected via HTTP messages following the REST API, users will receive a JSON/XML response, which contains the requested information.

The new "OpenDaylight northbound manager" tool in Net2Plan (under a new menu SDN) provides a graphical interface to orchestrate the controller operation following this API. This tool adds an additional panel to the offline network design tool, so-called "Connection manager", which handles the connection with the controller (see Fig. 4). Please, note that the use of REST API implies that no actual connection is

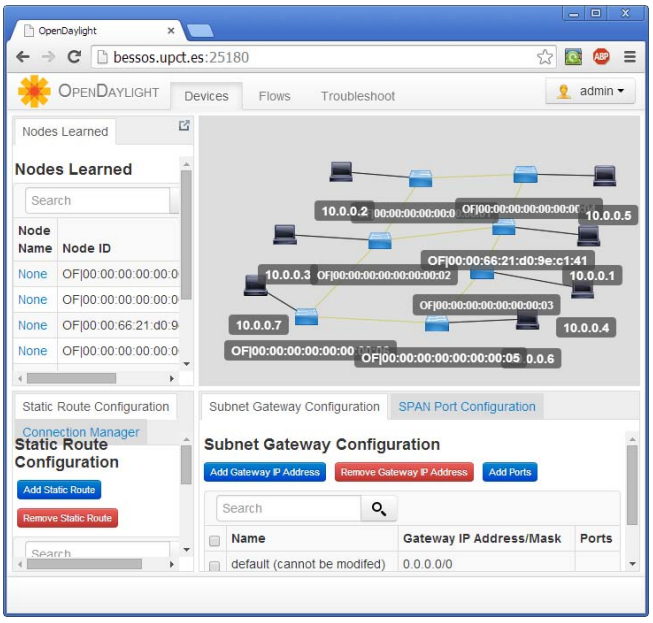

Fig. 3. WebUI of OpenDaylight.

established (the interaction is stateless), so information such as authentication to the controller is mandatory for any HTTP message.

This panel is divided into three parts. First, we have some fields to complete with the information to connect to the controller: IP address, port, username and password. Then, there is a textarea where some logging about successive connections is shown, e.g. for debugging purposes. Finally, there are three buttons corresponding to each of the implemented functionalities. The "Retrieve topology" button allows to import a topology discovered by OpenDaylight, so that becomes available into Net2Plan. The "Estimate traffic matrix" is able to estimate end-to-end traffic volumes from link load measurements. This is a nice input for routing problems, i.e. to minimize the maximum link utilization. The last option, "Update routing", translates Net2Plan routes (sequence of traversed links) into OpenFlow rules, and sends them back to the controller to install it inside the switches.

In the next section, we test this new tool using a set of experiments. In addition, implementation details are discussed therein.

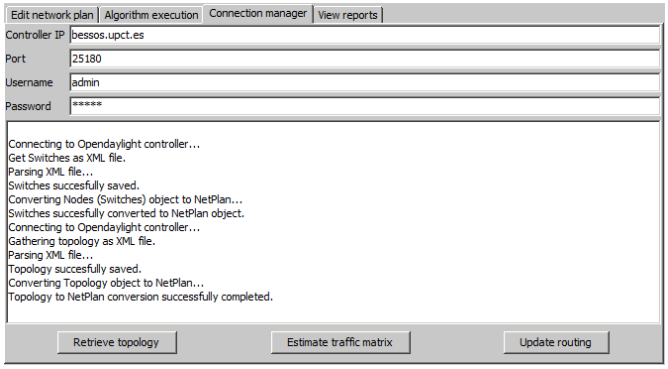

Fig. 4. Panel for connecting to OpenDaylight.

VI. EXPERIMENTS AND DISCUSSION

In order to evaluate our implementation of the northbound API, we conduct several experiments to test the interaction between Net2Plan and OpenDaylight.

A. Simulation environment

We execute our experiments in an OpenFlow-network emulated using Mininet [Lantz2010]. Mininet is a simple but powerful software that enables the emulation of a large number of switches and hosts in any desired topology thanks to the out-of-the-box virtualization capabilities of Linux systems.

We use Mininet to emulate the 7-node network shown in Fig. 5. Since Mininet uses Python scripts for configuration, we added a new option to the "Save design as…" option that is able to export a given Net2Plan design as a Python script for Mininet, according to the conventions specified in its documentation. We assume that each Net2Plan node consists of a switch and a host, which will be used to inject/receive traffic.

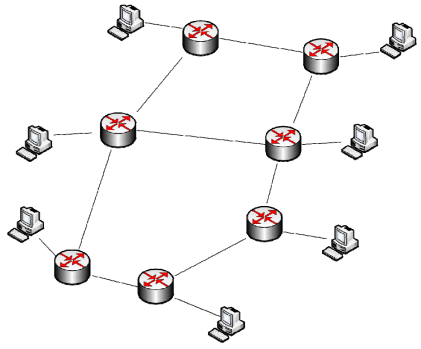

Fig. 5. Emulated topology (`example7nodes.n2p` in Net2Plan distribution).

Once the configuration file is generated, then OpenDaylight and Mininet (preferably in this order) should be executed.

B. Retrieving topology information

First, we want to retrieve the information about the emulated topology. Interestingly, OpenDaylight is able to detect all the switches via LLDP (link-layer discovery protocol), but hosts remain hidden until they inject/receive some traffic. In order to get the full topology (switches and hosts), we should issue the `pingall` command in Mininet. It is worth noting that switches do not implement any flow rule when initialized, so routing is working in reactive mode. OpenDaylight implements a routing module based on Dijkstra shortest-path algorithm (in number of hops), so rules are generated on demand, and installed into switches.

Once the topology is discovered by OpenDaylight (see Fig. 3), we could use the "Retrieve topology" button in Net2Plan to ask OpenDaylight for the information. This task is a little bit complex, since OpenDaylight divides the topology information into switches (SwitchManager API), links between switches (Topology API), and hosts (HostTracker API), and three queries are required. Anyway, Net2Plan has been engineered to get this information and transform it into a Net2Plan design. In addition, data such as link bandwidth or name of network interfaces are also introduced into the design.

Fig. 6 shows the discovered topology into Net2Plan. Please note that real locations are not indicated by LLDP packets, so positioning should be manual. Integration of graph layout algorithms (i.e. force-directed) is left as further work. Also, bidirectional links from Mininet are emulated by Net2Plan with unidirectional links in both directions.

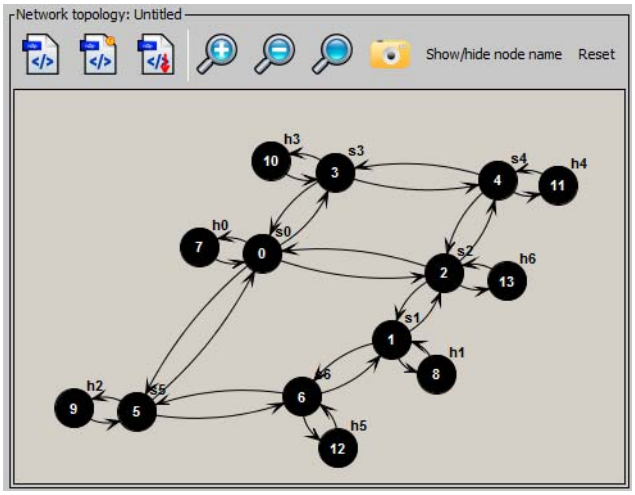

Fig. 6. Emulated topology retrieved by Net2Plan-OpenDaylight interface.

C. Estimation of end-to-end traffic volumes

Capacity planning and traffic engineering are critical task in operational networks. In this sense, the knowledge (or estimation) of aggregate traffic volumes between each ingress-egress pair, in the form of traffic matrix, is a critical input for those tasks. As the next step, we are interested on getting traffic information, which will be an input for the routing optimization algorithm.

Although OpenDaylight is able to provide per-flow information (Statistics API), in large production networks this is a heavy process. Instead, operators can use simpler and faster approaches, to obtain slightly accurate estimations of traffic matrices from link traffic, gathered e.g. from SNMP pollings. In our study, we use a quite simple approach such as the gravity model [11], to infer origin-destination traffic matrices from the information in the links (also from Statistics API).

According to this model, the traffic from node i to node j is given by (1):

$$T(i,j) = T^{in}(i)\frac{T^{out}(j)}{\sum_k T^{out}(k)} \quad (1)$$

where $T^{in}(i)$ is the traffic entering the network via the host-to-switch link at node i, and $T^{out}(i)$ is the traffic leaving the network via the switch-to-host link at node j. Please, note that the Statistics service in OpenDaylight gives us absolute traffic values rather than the difference between the last reported values and the current ones, so we need to calculate this difference to get the up-to-date traffic matrix. Finally, please note that any other (maybe more powerful) traffic matrix estimation could be used, by implementing it in Net2Plan. In our tests, several `iperf` processes are executed in background to generate traffic.

D. Real-time link utilization monitoring

Following with the Statistics service, we include an additional feature to our Net2Plan-OpenDaylight application which is able to make periodic polls (e.g. in intervals of 10 seconds) to monitor link utilization in real-time. To access this feature, users have just to right click over any link and execute the "View real-time traffic" option.

Here, OpenDaylight lacks of support to provide single link information, forces Net2Plan to get all the information from the origin/destination node and filter the data to get only the specific link values.

E. Installing forwarding rules

As mentioned before, OpenDaylight is able to configure shortest-path routing when no rules are specified. In this final experiment, we use Net2Plan to optimize those rules to achieve a minimum congestion routing. We skip mathematical details of the algorithm due to the lack of space. To summarize, we use an algorithm (`FA_minBottleneckUtilization_xde` in the repository [9]) that optimally solves an integer linear problem (ILP) which minimizes the maximum link utilization under non-bifurcated routing.

Once the solution is obtained, Net2Plan updates routing using the FlowProgrammer API. It is worth noting that Net2Plan routes are defined as the sequence of links traversed by each path. Hence, Net2Plan converts this sequence of links into a set of OpenFlow rules, one per switch, each of them indicating in/out-port and origin/destination IP addresses.

VII. Roadmap

Preliminary work in this paper is focused on understanding how a third-party application is able to communicate with OpenDaylight to perform some plain tasks such as reading topology and traffic information or applying simple forwarding rules. However, Net2Plan will eventually evolve into a complete platform able to perform different day-by-day tasks including the following: on-demand allocation, network diagnostics and troubleshooting (e.g. triggered by events from the southbound API such as link status changes or over-threshold link utilization), and so on. To this respect, REST APIs are limited in the sense that only provide an app-to-controller communication, so the development of an OSGi module, which allows communication in both directions, seems mandatory.

A clear example of OSGi module would be one converting Net2Plan into a path computation element (PCE). A PCE is a network component, application or node that can apply computational constraints and compute a network path or route based on a network graph. The PCE has access to the current state of the entire network and applies it in path computations. When a network needs a path for a new connection, it makes a request to the PCE by using a certain protocol. The challenge here is not to take an offline-optimized routing and to deploy it, but also to interact as a real-time online network controller.

Still, some interesting features can be explored with the existing functionalities. As an example, OpenFlow standard [12] describes how to implement load-balancing over multiple paths as forwarding rules, without relying into an external routing module inside the controller, using the 'group table' feature. In this sense, Net2Plan implements a number of algorithms (available in [9] under 'OSPF' keyword) able to compute IGP weights under different scenarios such as minimizing network congestion or guaranteeing a certain link utilization under single failures. Since the Internet core today is completely based on IP routers [1], we may apply equal-cost multi-path (ECMP) policies [13] to OpenFlow-enabled networks with seamless changes into the Net2Plan-OpenDaylight interface.

VIII. Conclusion

In this paper, we discuss the opportunities and challenges of the interaction between SDN controllers and third-party applications such as network planning and operation tools. Specifically, we present our Net2Plan tool, an open-source network planner, as a candidate application to orchestrate an OpenFlow-based network on top of the OpenDaylight controller. We present a case study in which Net2Plan is able to communicate with OpenDaylight and perform different experiments. Although it is still in the early stages, we expect our preliminary work may represent a promising ground for SDN community engagement.

References

[1] S. Das, G. Parulkar, and N. McKeown, "Rethinking IP Core Networks," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 5, no. 12, pp. 1431-1442, December 2013.

[2] N. McKeown *et al.*, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69-74, April 2008.

[3] OpenDaylight [Online]. Available: http://www.opendaylight.com/ [Last accessed: April 2014]

[4] B.A.A. Nunes, M. Mendonça, X.N. 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.

[5] N. Foster *et al.*, "Languages for Software-Defined Networks," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 128-134, February 2013.

[6] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114-119, February 2013.

[7] P. Pavon-Marino and J.L. Izquierdo-Zaragoza, "Net2Plan: An open-source network planning tool for bridging the gap between academia and industry," *IEEE Network*, in press.

[8] A. Aguado, V. Lopez, J. Marhuenda, O. Gonzalez de Dios, and J.P. Fernandez-Palacios, "ABNO: a feasible SDN approach for multi-vendor IP and optical networks," in *Proceedings of the 2014 Optical Fiber Conference* (OFC 2014), San Francisco (USA), March 2014.

[9] Net2Plan – The open-source network planner [Online]. Available: http://www.net2plan.com/ [Last accessed: April 2014]

[10] 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* (HOTNETS-IX), Monterey (USA), October 2010.

[11] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg, "Fast accurate computation of large-scale IP traffic matrices from link loads," in *Proceedings of the International Conference on Measurements and Modeling of Computer Systems* (SIGMETRICS 2003), June 2003.

[12] *OpenFlow Switch Specification 1.1.0*, Open Networking Foundation, February 2011.

[13] C. Hopps, "Analysis of an Equal-Cost Multi-Path Algorithm," RFC 2992, November 2000.