2014 IEEE 28th International Conference on Advanced Information Networking and Applications

# Towards SLA Policy Refinement for QoS Management in Software-Defined Networking

Cristian Cleder Machado, Lisandro Zambenedetti Granville, Alberto Schaeffer-Filho, Juliano Araujo Wickboldt

Computer Networks Group
Institute of Informatics
Federal University of Rio Grande do Sul
Porto Alegre, Brazil
Email: {ccmachado, granville, alberto, jwickboldt}@inf.ufrgs.br

***Abstract*— *Software-defined networking* (SDN) is a dynamic, adaptable, controllable and flexible network architecture. It provides an extensible platform for delivery of network services, capable of responding quickly to service requirement changes. As a result, SDN has become a suitable scenario for the application of techniques and approaches for improved infrastructure management, such as *policy-based management* (PBM). In PBM, using techniques such as refinement, a high-level policy – *e.g.*, specified as a *service level agreement* (SLA) – can be translated into a set of corresponding low-level rules, enforceable in various elements of a system. However, when using SLAs, their translation to low-level policies, e.g., for controller configuration, is not straightforward. If this translation is not done properly, the controller may not be able to meet the implicit requirements of the SLA, failing to satisfy the goals described in the high-level policy. This paper proposes a novel approach towards SLA policy refinement for *quality of service* (QoS) management (based on routing) in *software-defined networking*. It consists of an initial manual process performed by an administrator, followed by an automatic policy refinement process executed by an OpenFlow controller. As a result, our approach is capable of identifying the requirements and resources that need to be configured in accordance with SLA refinement, and can successfully configure and execute reactive dynamic actions for supporting dynamic infrastructure reconfiguration.**

***Keywords*—*refinement; policy; management; sdn;***

## I. Introduction

For many years, organizations have been employing management strategies for dealing with the scale and computational complexity of their Information and Communications Technology (ICT) infrastructures [1]. *Software-defined networking* (SDN) is a dynamic, adaptable, controllable and flexible network architecture. It provides an extensible platform for delivery of network services, capable of responding quickly to service requirement changes. SDN facilitates network operations based on the idea of controlling and programming the behavior of network devices, where the rules for packet forwarding can be controlled by software applications developed independently from the hardware [2]. SDN enables the deployment of network applications that perform sophisticated traffic monitoring and traffic processing. This allows an effective way for providing dynamically – and at runtime – services that support, for example, quality of service (QoS) reconfiguration, access control and load balancing.

SDN is characterized by a centralized control plane, which allows moving part of the decision-making logic of network devices to external controllers [3]. This characteristic provides the controller device with the ability to have an overall view of the network infrastructure, thus becoming aware of network elements and network characteristics. As a result, SDN has become a suitable scenario for the application of techniques and approaches for improved infrastructure management, such as *policy-based management* (PBM). Policies can be analyzed and modified as necessary at different levels of abstraction, without changing the functionality implemented in network devices. The behavior of certain features can be modified without changing the code or the implementation of the system, or even, without manual intervention. In addition, at runtime, policy rules can be triggered by conditions monitored in the network, thus providing support for the adaptive reconfiguration of devices enforcing these rules.

Several approaches [4], [5], [6], [7], [8], [9], [10] have been using policies to introduce control rules that govern the operation of SDN. Most of these approaches use rules created directly in a low-level notation, and the rules are introduced straight into the controller. However, when using *service level agreements* (SLAs), their translation to low-level policies, *e.g.*, for controller configuration, is not straightforward. If this translation is not done properly, the controller may not be able to meet the implicit requirements of the SLA, thus failing to satisfy business level goals. To address this issue, our goal is to develop techniques for *policy refinement*, where a high-level policy specification can be translated into a set of corresponding low-level policies, which are deployed in various elements of a network [11]. Thus, controllers can be configured with the specific objectives, written in a low-level language (device rules), that accurately satisfy the high-level policy goals. The use of PBM and especially policy refinement has been historically investigated [12]. Although there have been some promising developments in the area of policy analysis, policy refinement is a nontrivial process and it remains to a large extent a much neglected research area.

This paper proposes a novel approach to SLA policy refinement for QoS management (based on routing) in SDN. Our approach is based on an initial manual process performed by an administrator, followed by an automatic policy refinement process executed by an OpenFlow controller. Therefore, our

1550-445X/14 $31.00 © 2014 IEEE
DOI 10.1109/AINA.2014.148

proposed approach is able to identify properties and characteristics of applications that require QoS. This is accomplished by following the specifications of the high-level SLA policies, and refining them into low-level policies. These low-level policies are understood and enforced by network devices for automatic reconfiguration and optimization of network resource usage at runtime. The controller was designed using POX [13], written in Python Language [14], and uses the OpenFlow protocol [15] to coordinate network devices. Experiments have been performed using the Mininet emulator [16].

To the best of our knowledge, this is the first time that policy refinement has been applied to SDNs. The development of a policy refinement strategy specifically for SDNs can benefit from several aspects found in these environments. Firstly, it is possible to more easily collect monitoring information and network traffic data, with the aim of checking whether high-level goals are being fulfilled by low-level rules or not. Secondly, we have the ability to refine policies to lower level ones, which can be (re)implemented in all network device, since the SDN controller uses a standard protocol and interface to determine rules and actions on any network device.

The paper is organized as follows. Section II provides a brief description of the main concepts used in our approach. Section III presents an overview of our policy refinement approach. Section IV presents the operation of the controller. Section V describes the scenarios, experiments and initial results. In Section VI some studies related to our approach are introduced and discussed. Section VII concludes the paper with final remarks, along with a proposal for future work.

## II. Background

This section provides an overview of the main concepts employed in our approach. Section II-A briefly presents software-defined networking (SDN) and OpenFlow. Section II-B explains the notions of *policy-based management* (PBM) and policy refinement. Finally, Section II-C describes properties and requirements for quality of service (QoS).

### A. Software-Defined Networking (SDN) and OpenFlow

Software-defined networking is a dynamic, adaptable, controllable and flexible network architecture. It provides an extensible platform for delivery of network services, capable of responding quickly to service requirement changes [2]. SDN is characterized by a logically centralized control plane, which allows moving part of the decision-making logic of network devices to external controllers (Figure 1). This provides controller devices with the ability to have an overall view of the network and its resources, thus becoming aware of all the network elements and their characteristics. Based on this centralization, network devices become simple packet forwarding elements, which can be programmed through an open interface, such as the OpenFlow protocol [2].

Briefly, the main elements of an OpenFlow-based SDN architecture are (depicted in Figure 1): (*i*) a flow table that contains an input and a specific action to be performed for each active flow, and (*ii*) an abstraction layer that communicates securely with a controller reporting on new input flows that are not present in the flow table. Each entry in the flow table consists of: *(a)* a mask of fields found in the packet header, which is used to match the incoming packets, *(b)* counters for collecting statistics for each specific flow, such as number of bytes, number of packets received, and flow duration, and *(c)* a series of actions to be performed when a packet matches the corresponding mask [17].

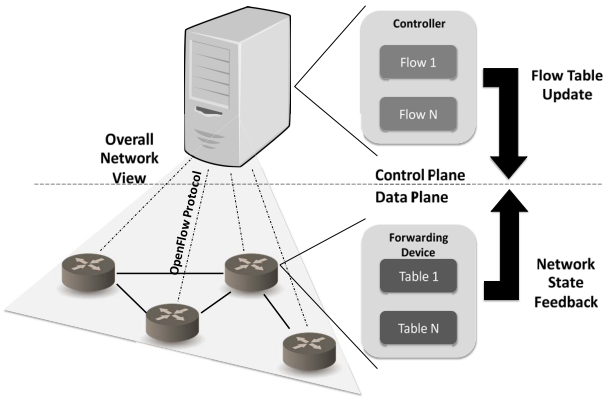


Fig. 1. OpenFlow Architecture.

OpenFlow is an open protocol that allows the development of programmable mechanisms based on a flow table in different forwarding devices. The OpenFlow protocol establishes a secure communication channel between OpenFlow switches and the controller, using this channel for controlling and establishing flows according to customizable programs [15].

The OpenFlow controller is the central element of the approach. The controller runs customized programs to decide which rules and actions are going to be installed to control packet forwarding in each switch element. OpenFlow is also characterized by the separation between the data plane and the control plane. On the one hand, the data plane is concerned with the forwarding of packets based on rules, called OpenFlow actions, associated with each table entry in the switch. On the other hand, the control plane enables the controller to manage the entries in the flow table and the rules associated with the desired traffic [17].

### B. Policy-Based Management (PBM) and Policy Refinement

Policies are defined as a collection of rules which express and enforce the required behavior of a resource. RFC 3198 [18] provides the following definitions for a policy:

- A defined goal or action that determines how present and future decisions are taken. Policies are established or executed within a particular context;
- Policies refer to a set of rules to manage and monitor access to features of a particular ICT infrastructure [19].

In PBM an administrator specifies the infrastructure objectives/goals and constraints in the form of rules to guide the behavior of the elements in a system [20]. The use of PBM presents three main benefits [21]. Firstly, policies are predefined by administrators and stored in a repository. When an event occurs, these policies are requested and accessed automatically, without the need of manual intervention. Secondly,

the formal description of policies permits automated analysis and verification with the aim of guaranteeing consistency to some extent. Thirdly, because of the abstraction of technical details, policies can be inspected and changed dynamically at runtime without modifying the underlying system implementation.

Policies may be seen in two principal levels of abstraction: *low-level policies*, which are related to a domain or a device, and *high-level policies* that are more user-friendly. A simple example of a low-level policy is the settings on routers so multimedia traffic packets have higher priority over peer-to-peer (P2P) [22] traffic packets. An example of high-level policies are SLAs [23]. Policy refinement aims to translate a high-level policy into a set of corresponding low-level policies. In other words, using techniques for refinement, a high-level policy such as a *service level agreement* (SLA) can be translated into low-level policies that are applicable in various elements of a system [24], [25].

SLAs are generally business-oriented, and they leave aside the technical details, which are guided by a *service level specification* (SLS) and a *service level objective* (SLO). The SLS is a technical interpretation of the SLA. The SLO is a sub-item of the SLS that contains the parameters to achieve the SLS [26].

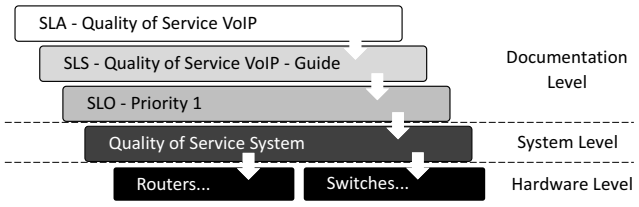


Fig. 2. Example policy refinement.

Figure 2 presents an overview of the process of refining SLAs. The SLA, SLS and SLO represent, respectively, the *documentation* describing the service in a formal way, the technical form (guide) for its functioning requirements, and the parameters aimed at quality and satisfaction. At the *system level*, a *quality of service* system interprets the management requirements, and enforces policies for the configuration of elements at the *hardware level*.

The main objectives of policy refinement are identified by Moffett and Sloman [25] as:

- To determine what resources are needed to fulfil policy needs;
- To translate the high-level policy into a set of operational policies that the system can enforce;
- To examine whether the low-level policies actually meet precisely the requirements specified by the high-level policy.

The refinement process typically involves stages of decomposition, operationalization, implementation, operation and re-refinement of goals and subgoals [27], [28]. Policy refinement aims to automate these stages to get the translation of policies relating to objects and implementable actions, and ensure that the low-level policies still satisfy the goals defined by the high-level policy.

### C. Quality of Service (QoS)

Quality of service refers to a combination of many properties or features of a service, such as:

- *Availability*: related to the percentage of time that a service is operating;
- *Security*: includes the presence and types of authentication mechanisms, data confidentiality and data integrity, non-repudiation of messages, and resilience to denial-of-service attacks [29];
- *Response time*: time required for a service to respond to individual types of requests;
- *Throughput*: rate at which service requests are processed. QoS measurements include full capacity or a ratio that describes how the throughput is changed according to load [30].

In computer networks, the term *quality of service* [31] is related to a set of standards and mechanisms for ensuring high performance for critical applications. Through the use of QoS mechanisms, network administrators can use existing resources efficiently and thus ensure the required level of service without the need to expand or over-provision their networks.

## III. Policy Refinement Approach

In this section we present our SLA policy refinement model, and illustrate it with a case-study. We limit ourselves to a Voice over IP (VoIP) QoS scenario for better presentation of the approach, but our approach is generic and can be used for various applications, such as video streaming, replication, and backup. Similarly, it can be applied to various services, such as monitoring, access control, and load balancing.

Figure 3 shows an overview of our approach for SLA policy refinement. We assume as input an SLA that has a high-level specification and determines which guarantees must be fulfilled for the optimal operation of the services. Currently, the refinement process includes a significant amount of manual labor, in which case it requires an administrator to define the initial rules, interpreting and translating each policy level, and identifying the objectives to be fulfilled at the level below, *i.e.*., SLA $\rightarrow$ SLS and SLS $\rightarrow$ SLOs (*top-down process*). The actions and objectives described in lower-levels should be verified in order to determine whether they are faithfully related to the levels above (*bottom-up process*). In addition, policies can be (re)evaluated based on the feedback provided by the network controller. Finally, the concrete and executable policies translated from the high-level policies are stored in a repository that will serve as input for the implementation of system elements.

The refinement process comprises three stages, which are represented in Figure 3 (the manual and the automatic processes are also indicated in the diagram). In the *first stage*, a high-level policy (SLA) is interpreted and translated manually

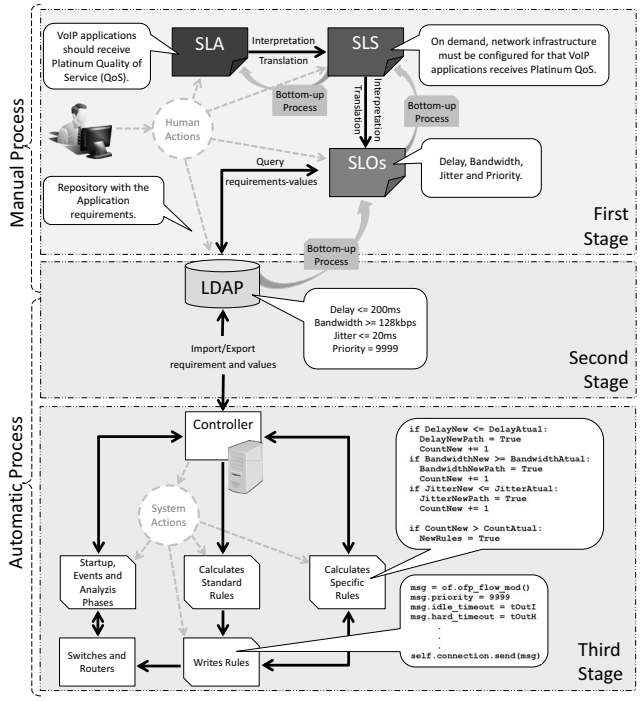


Fig. 3. Refinement of QoS VoIP SLAs.

by an administrator so that all its technical requirements can be defined. In our case-study, the SLA specifies that *"VoIP applications should receive Platinum quality of service (QoS)"*. The interpretation and translation of the SLA result in the technical SLS: *"On demand, the network infrastructure must be configured so that VoIP applications receives Platinum QoS."*. Next, the interpretation and translation of the SLS are performed in order to analyze the possible objectives (SLOs) that satisfy VoIP Platinum QoS. The value of each objective is obtained by querying a specific QoS class (in our case Platinum QoS) in an LDAP repository [32]. We assume that this repository has been previously populated by an administrator with *(i)* the set of QoS classes (*e.g.*, Platinum, Gold, and Silver), *(ii)* class requirements (*e.g.*, Delay, Jitter, and Bandwidth) *(iii)* and the requirement values (*e.g.*, 200ms, 20ms, and 128kbps). Through a *bottom-up process* we can verify that the *"Delay* $\leq$ *200ms"* is the value of the requirement *"SLO - Delay"* for services that require QoS Platinum class. In addition to QoS classes, the LDAP repository also contains a protocol list (*e.g.*, FTP, HTTP, SIP, RTP, and RTSP). This list is also previously supplied by the administrator, using RFC 1700 [33] recommendations and service analysis that will run in the infrastructure, along with the ports and protocols that are not in RFC 1700. The information in this list will be used later, in the Events Phase and Analysis Phase (see Section IV).

In the *second stage*, the administrator associates the VoIP service with a specific protocol (the list of protocols can have multiple entries because there might be VoIP services using different protocols). For our tests we set the protocol H.323 [34] in Platinum class registered in the LDAP repository. This association is performed to define which protocols receive Platinum QoS in the controller.

In the *third stage*, the controller loads the rules of the repository, filtered by all classes of QoS with all requirements, values and associated protocols, storing the concrete rules in a policy dictionary. Next, for each new network flow, the controller performs an analysis phase to enforce the QoS requirements for the application. It verifies each requirement in order to identify the best path in the network, which priority should be given and what network elements should receive the rules. To support dynamic reconfigurations, for each valid change in the *second stage*, an identifier (*Serial #*) is incremented to allow version control of the repository. The network controller is configured to periodically read the repository (*e.g.*, every *1 minute*), and if the *Serial #* previously loaded into the system is equal to the one in the repository, nothing needs to change. However, if the *Serial #* has increased, the controller reloads the services and requirements and applies the new rules.

In order to apply our approach there was the need to customize some functionality in the SDN controller. This was required for collecting information about the network infrastructure, which is later used, for example, to calculate optimal routes. This customization was based on SDN native features only, and thus can be applied to any controller implementation. Therefore, our approach is not tied to any specific controller design or language. For example, topology discovery, which is available in a POX controller from the discovery module, is a native feature of SDN achieved by all controllers in different implementations.

The advantage of our approach, compared to previous investigations of policy refinement in non-SDN deployments, is that a controller holds updated information about the domain and the elements that should be considered for deployment of the concrete low-level policies. From this, the administrator has an overview of the current status of the domain, and can recognize the limits that may affect each SLA beforehand. We plan to develop a Graphical User Interface (GUI) that decomposes policies in order to achieve low-level policies with refinement techniques. Thus, infrastructure operators may introduce SLAs for automatic decomposition. Subsequently, developers of controllers can interpret each low-level policy generated with a system description that indicates the rules that must be applied by the controller for execution.

## IV. Controller Prototype

This section presents the operation of the controller, which is divided into three phases (Figure 4): *(i) Startup Phase*: discovers services and possible paths between network elements; *(ii) Events Phase*: identifies service events and determines the best path based on the characteristics of the network and service requirements; and *(iii) Analysis Phase*: implements the rules and monitors the network in order to identify possible enhancements for the active flows and reconfigure the structure. The operation of each phase is detailed in the following.

### A. Startup Phase

In this phase, the controller reads information from a repository that contains descriptions and requirements of all services that are initially scheduled to run in the network. This information is identified and set by the administrator

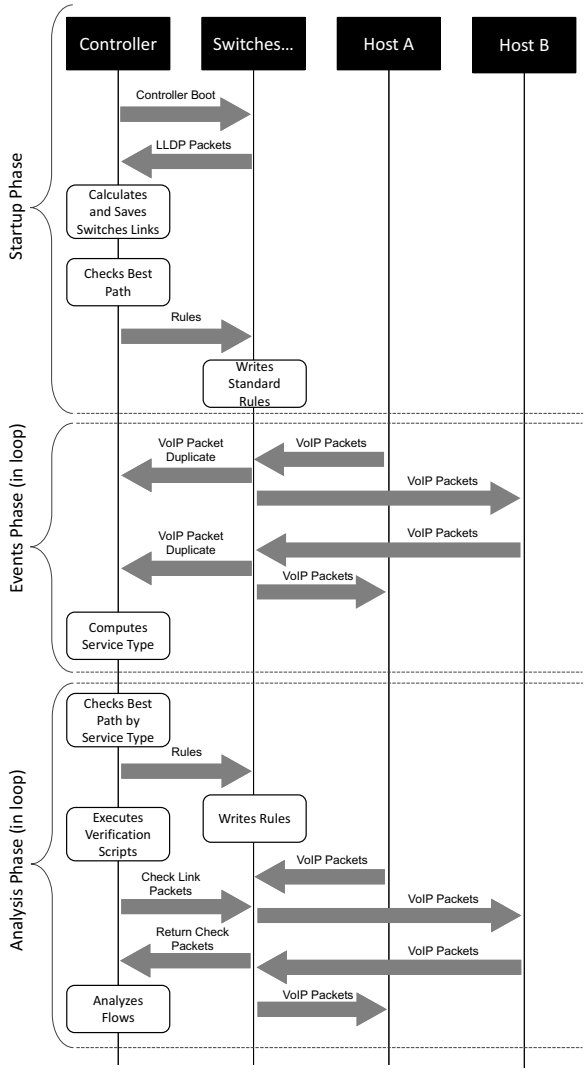


Fig. 4. Flow Diagram VoIP.

as requirements to perform the low-level policies in order to fulfill the SLA definitions. Thereafter, the controller monitors the network in order to discover the devices and topology. Network devices are instructed to send Link Layer Discovery Protocol (LLDP) [35] packets to report their location on the topology, which is stored by the controller in an internal data structure.

While the LLDP packets are received and stored, a routine for calculating paths between all elements is performed using Dijkstra's Shortest Path Algorithm [36]. We assume at first that the best path is the one with the smallest number of hops, because at this stage values such as delay and jitter are unknown, thus we consider that these values are initially zero. Subsequently, all paths are sorted from the shortest to the longest path and stored into a list.

Once all possible paths between network elements have been calculated, the controller writes rules (which we call *standard rules*) in the flow table of the switches that are in the best path between each of the network elements. The purpose of these rules is to handle each new service flow in such a way that its first packet is duplicated, *(i)* forwarding one copy of the packet to the controller in order to inform it that there is a new service flow, which will be further evaluated (see Section IV-C), but also *(ii)* handling the duplicate packet with the idea of best-effort network and no-act delay Round-Trip Time (RTT) [37]. After setting up the rules in the switches, the controller inserts in the repository all the update network information collected at this stage.

### B. Events Phase

This phase aims to identify events in the network infrastructure. At the moment, *(i) Dataflow Event* is the only event handled by our controller, but we may extend it to handle events such as *(ii) New Device Addition to the Topology* and *(iii) Dropped Communication Link*. *Dataflow Events* are generated by a new type of service in the network (*e.g.*, a video streaming), and are stored in a services dictionary.

The *Events Phase* stays in a loop during the operation of the infrastructure. When running a specific protocol, such as VoIP communication, the first few packets of the communication are initially treated by the standard rules and switches are instructed to duplicate the first packet of each new service flow (*e.g.*, between host A and host B), sending a copy to the controller. The controller stores in a list the protocols that are running in a link, which are identified by a function that reads the *IP Packet Header* and gets the information from the *Protocol Field*. Then, the controller checks what are the necessary requirements for the proper functioning of such protocol through the information loaded from the repository and decides whether the controller should start the *Analysis Phase* (see Section IV-C). The *Analysis Phase* is initiated immediately if the protocol of the new flow is not yet included in the list of protocols for the link, or if some other flow is already sharing the same path but using a different protocol.

### C. Analysis Phase

After identifying the service requirements, the controller calculates the new rules for best path, taking into account the specific weights of the service. The calculations are carried for the paths using as weights the bandwidth (BW), throughput (T), delay (D), jitter (J), loss rate (LR), and number of hops (NH) in each link of the physical topology [38]. *E.g.*, for VoIP we define the requirements [39] for calculation of the best link as an amount of bandwidth that varies according to the encoding used, low delay and low jitter, and priority (P). We check the value of each of these requirements using an analysis function that sends Internet Control Message Protocol (ICMP) [40] packets and store the return value in a state vector of links. When calculating the jitter, we store a vector with the 30 last values and calculate the average. For the purposes of our experiments, the requirements of VoIP SLA goals were considered as $D \leq 200ms$, $BW \geq 128kbps$, $J \leq 20ms$, and $P = 9999$. Each requirement/value pair is presented in order of importance. The link that satisfies the largest number of requirements with priority numbered from left to right is chosen as the best path.

These new rules are reconfigured at runtime and only on switches that have the flow, aiming to reduce processing

overhead where there is no need for such rules. Unlike the standard rules, these *specific rules* are configured with a timeout in the flow table of each element. In our experiments, we set the value of the timeout to *15s* (but it can be easily adjusted based on the analysis of the services being executed).

Periodically, the controller checks if the the configured links remain the best choices for the current flow. In our experiments, we set the checking intervals to *10s* (but these can also be easily adjusted). If at any time the controller identifies that there is a better alternative path, new rules are sent to the switches in order to process the flow as efficiently as possible. If the current path remains the best, the controller only increases the value of the timeout for the rules on each switch for the corresponding flow. This phase stays in a loop during the operation of the infrastructure.

## V. Scenarios, Experiments and Initial Results

This section describes our test environment and some initial results. The experiments were performed on an Intel 2.4 GHz QuadCore processor with 6 GB RAM memory. The scenarios were created using the Mininet emulator. Five scenarios were created in our experiments. An overview of scenarios A, B, and C is shown in Figure 5. Due to the large number of elements, scenarios D and E could not be clearly presented in a figure, but the elements that comprise each scenario are described in Table I.

### A. Scenarios

The scenarios that we built had increasing numbers of switches and redundant links, thus increasing path diversity between any two hosts. Table I shows the number of hosts, switches and links in each scenario.

TABLE I. Number of hosts, switches and links in each scenario

| Scenario | A | B | C | D | E |
|---|---|---|---|---|---|
| Hosts | 4 | 8 | 16 | 32 | 64 |
| Switches L0 | 2 | 4 | 8 | 16 | 32 |
| Switches L1 | 2 | 4 | 8 | 16 | 32 |
| Switches L2 | 0 | 2 | 4 | 8 | 16 |
| Switches L3 | 0 | 0 | 0 | 4 | 8 |
| Links | 4 | 12 | 32 | 80 | 96 |
| Total number of switches | 4 | 10 | 20 | 44 | 88 |

The topologies used in the experiments were based on Fat-Tree topologies [41] (depicted in Figure 5). By increasing the number of switches and redundant links, these topologies can be used to, for example, maintain system availability in the presence of problematic links and to reduce traffic congestion in the infrastructure.

### B. Experiment 1 - Standard Rules

Experiment 1 measures the time spent on the initial calculation of all paths between switches and the establishment of standard rules on all switches in the network. Figure 6 shows the time for calculation and installation of the standard rules during the *Startup Phase* of the controller.

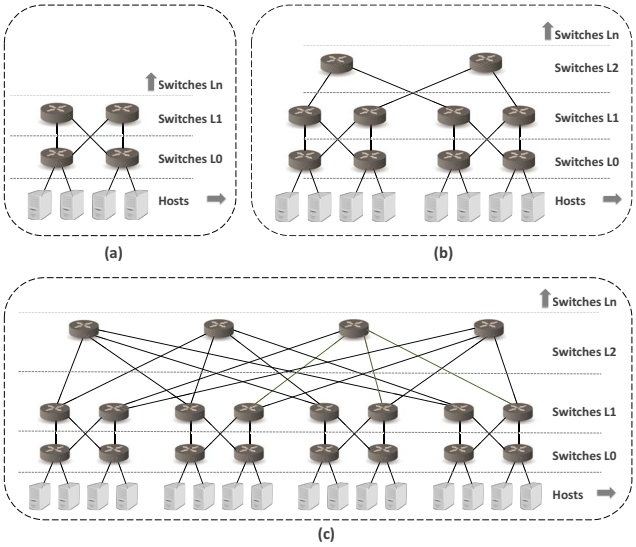


Fig. 5. Scenarios for the experiments with increasing number of switches and link redundancy.

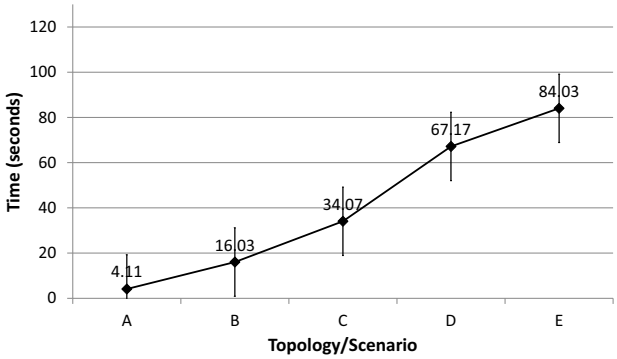


Fig. 6. Time for the calculation and installation of standard rules in the switches in the startup phase and recognition of topology in each experiment.

As can be observed in Figure 6, with the increase in the number of elements and links in the topology, the time to calculate all paths between any network element increases. Even with this growth, we justify the execution of this calculation and installation of these initial standard rules because we want avoid any delay when new flows arrive. We also take into consideration that, due to the initial calculation, later, in the *Analysis Phase* (see IV-C), we have already all possible flow paths, which makes the calculation of specific rules faster (as can be seen in Figure 7).

### C. Experiment 2 - Specific Rules

Experiment 2 measures the time spent on calculating and installing specific rules when a service is identified. The scenarios used in Experiment 2 are the same as previously described. Figure 7 shows the calculation and installation time of specific rules for the identified service.

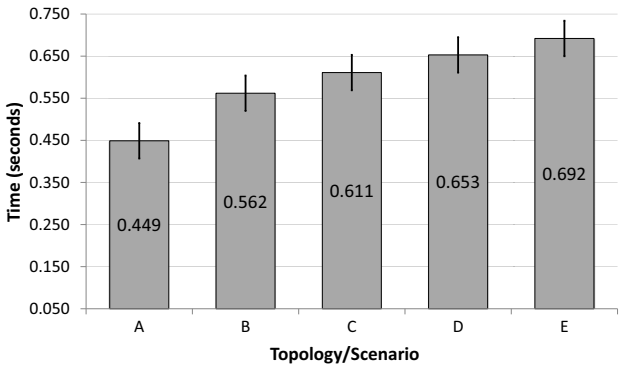


Fig. 7. Time for calculation and installation of specific rules in the switches in each experiment.

The increase in the number of elements in each topology is reflected in an increase in the time for calculation and installation of the rules. There is a minimum time necessary for calculating the specific rules, but that does not delay service flow processing due to the pre-established standard rules.

These results show that due to the calculations performed initially, our proposal is able to perform a quick reconfiguration of specific rules, since we already have a populated list of the best links between switches. As part of our future work, we intend to run more experiments that generate background traffic flows competing with other applications and validate dynamic reconfigurations in our approach.

## VI. Related Work

Rubio-Loyola *et al.* [5] have investigated the sharing of virtualized network resources in software-defined networking. It shows that the ability to program network elements helps to dynamically adapt the network to both predictable and unpredictable changes. The authors present an orchestration plane (OP) which aims to manage the system behavior in response to context changes, and in accordance with business goals and policies. However, the authors do not specify if there is any refinement approach to translate policies in the different abstraction levels.

Regarding policy refinement, Bandara *et al.* [12] have presented the use of goal design and abductive reasoning to derive strategies that attain a specific high-level goal. Policies can be refined by combining strategies with events and restrictions. The authors provide tool support for the refinement process, and use examples of DiffServ [23] QoS management. The refinement is built on a systematic approach, making strategies derived for the lower levels satisfy the requirements of a high-level policy. As part of our future work, we intend to use similar techniques for improving our refinement process, and explore the dynamic aspects of software-defined networking.

## VII. Conclusions and Future Work

QoS mechanisms allow network administrators to use existing resources efficiently and ensure the required level of service without the need of expanding or over provisioning their networks. However, to ensure that QoS requirements are satisfied across the network is difficult, as network devices such as switches and routers are heterogeneous and have proprietary interfaces. Moreover, QoS architectures such as DiffServ [23] and IntServ [42] are built over current networks. These are based on distributed hop-by-hop routing, without a broader perception of global, network-wide capabilities. Alternatively, in SDN networks, the controller element has an overall view of the infrastructure and the services running on it.

Policy-based management in SDNs can be used to specify goals and constraints in the form of rules to guide the operation of network elements. Policy abstractions can be used not only to adapt the controlled system, but also to adjust the policies themselves, changing their behavior to better achieve the system goals. For example, a policy may add QoS control rules for network monitoring and traffic analysis. If the rate at which the network is read is too high, communication costs for monitoring will end up interfering, and generating more traffic. However, if it is excessively low, the system may not be sufficiently aware of changes, failing to fulfill specific objectives. To solve this problem, trends in behavior can be analyzed and policies adjusted dynamically to better reflect the needs of specific resources.

In this paper, we advocate the use of policy refinement techniques in SDNs. We aim to remove much of the manual workload of administrators in the configuration of network elements. In particular, we focus on the refinement of QoS requirements for different applications and services (specified in SLAs) into the configuration of controllers and switches. As a result of our approach, we identified the resources that need to be configured in accordance with the SLAs, and successfully executed reactive dynamic actions used in the reconfiguration of the infrastructure. Our experiments have shown that an initial calculation of best-effort paths between network elements can increase the performance of the network during runtime. Certainly there is an overhead related to the startup phase of the controller. However, this is justifiable as it improves the performance of the network during runtime. In a traditional SDN network, the first packet of the a new service flow is forwarded to the controller, incurring an RTT delay at the start of each flow, where there is still no rules in the flow tables of switches. Our standard rules created at the startup phase of the controller help to mitigate this problem. Our approach provides an improved flow processing strategy, by reorganizing flows upon the arrival of new service requests.

The policy refinement approach just described is still a work in progress. As part of our future work, we plan to study other case-studies to validate the broader applicability of our work. We are also going to assess how the work presented in this paper can be generalized to other types of SLA and QoS requirements. Furthermore, we are going to investigate techniques that can be used to automate the refinement process. We also intend to investigate techniques for detection and resolution of policy conflicts. Conflicts may arise due to omissions, errors or differing requirements of administrators when specifying policies. One common source of policy conflicts is the refinement process itself, during the translation of high-level goals into implementable low-level policies [43].

## REFERENCES


[1] J. O. Fitó, M. Macias, F. Julia, and J. Guitart, "Business-driven it management for cloud computing providers," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*. IEEE, 2012, pp. 193–200.

[2] O. W. Paper, "Software-Defined Networking: The New Norm for Networks," Open Networking Foundation, Tech. Rep., April 2012.

[3] S. Sezer, S. Scott-Hayward, P. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for sdn? implementation challenges for software-defined networks," *Communications Magazine, IEEE*, vol. 51, no. 7, pp. 36–43, 2013.

[4] A. Patel, P. Ji, and T. Wang, "Qos-aware optical burst switching in openflow based software-defined optical networks," in *Optical Network Design and Modeling (ONDM), 2013 17th International Conference on*, 2013, pp. 275–280.

[5] J. Rubio-Loyola, A. Galis, A. Astorga, J. Serrat, L. Lefevre, A. Fischer, A. Paler, and H. Meer, "Scalable service deployment on software-defined networks," *Communications Magazine, IEEE*, vol. 49, no. 12, pp. 84–93, 2011.

[6] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium, Tech. Rep*, 2009.

[7] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*. USENIX Association, 2010.

[8] P. Calyam, S. Rajagopalan, A. Selvadhurai, S. Mohan, A. Venkataraman, A. Berryman, and R. Ramnath, "Leveraging openflow for resource placement of virtual desktop cloud applications," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, 2013, pp. 311–319.

[9] A. Khan and N. Dave, "Enabling hardware exploration in software-defined networking: A flexible, portable openflow switch," in *Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on*, 2013, pp. 145–148.

[10] G. Hampel, M. Steiner, and T. Bu, "Applying software-defined networking to the telecom domain," in *INFOCOM, 2013 Proceedings IEEE*, 2013, pp. 3339–3344.

[11] N. Mavrogeorgi, S. Gogouvitis, A. Voulodimos, G. Katsaros, S. Koutsoutos, D. Kiriazis, T. Varvarigou, and E. K. Kolodner, "Content based slas in cloud computing environments," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. IEEE, 2012, pp. 977–978.

[12] A. K. Bandara, E. C. Lupu, A. Russo, N. Dulay, M. Sloman, P. Flegkas, M. Charalambides, and G. Pavlou, "Policy refinement for diffserv quality of service," *IEEE eTransactions on Network and Service Management*, vol. 3, no. 2, 2005.

[13] POX, "Pox openflow controller," 2013, Accessed: Sept. 2013. [Online]. Available: http://www.noxrepo.org/pox/about-pox/

[14] Python Software Foundation, "Python language reference, version 2.7," 2013, Accessed: Sept. 2013. [Online]. Available: http://www.python.org

[15] 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.

[16] 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*. ACM, 2010.

[17] K. Bakshi, "Considerations for software defined networking (sdn): Approaches and use cases," in *Aerospace Conference, 2013 IEEE*. IEEE, 2013, pp. 1–9.

[18] A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, and S. Waldbusser, "Terminology for policy-based management," RFC Editor, Tech. Rep., 2001.

[19] B. Moore, E. Ellesson, J. Strassner, and A. Westerinen, "Policy core information model–version 1 specification," RFC 3060, February, Tech. Rep., 2001.

[20] D. C. Verma, "Simplifying network administration using policy-based management," *Network, IEEE*, vol. 16, no. 2, pp. 20–26, 2002.

[21] W. Han and C. Lei, "A survey on policy languages in network and security management," *Computer Networks*, vol. 56, no. 1, pp. 477–489, 2012.

[22] S. Androutsellis-Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies," *ACM Computing Surveys (CSUR)*, vol. 36, no. 4, pp. 335–371, 2004.

[23] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," RFC 2475, December, Tech. Rep., 1998.

[24] A. Bandara, E. Lupu, J. Moffett, and A. Russo, "A goal-based approach to policy refinement," in *Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on*, 2004, pp. 229–239.

[25] J. Moffett and M. Sloman, "Policy hierarchies for distributed systems management," *Selected Areas in Communications, IEEE Journal on*, vol. 11, no. 9, pp. 1404–1414, 1993.

[26] I. Aib and R. Boutaba, "On leveraging policy-based management for maximizing business profit," *Network and Service Management, IEEE Transactions on*, vol. 4, no. 3, pp. 25–39, 2007.

[27] R. Craven, J. Lobo, E. Lupu, A. Russo, and M. Sloman, "Policy refinement: Decomposition and operationalization for dynamic domains," in *Network and Service Management (CNSM), 2011 7th International Conference on*, 2011, pp. 1–9.

[28] ——, "Decomposition techniques for policy refinement," in *Network and Service Management (CNSM), 2010 International Conference on*, 2010, pp. 72–79.

[29] J. Mirkovic, S. Dietrich, D. Dittrich, and P. Reiher, *Internet Denial of Service: Attack and Defense Mechanisms (Radia Perlman Computer Networking and Security)*. Prentice Hall PTR, 2004.

[30] D. A. Menascé, "Qos issues in web services," *Internet Computing, IEEE*, vol. 6, no. 6, pp. 72–75, 2002.

[31] J. Korhonen, H. Tschofenig, M. Arumaithurai, A. Lior, and M. Jones, "Traffic classification and quality of service (qos) attributes for diameter," RFC 5777, February, Tech. Rep., 2010.

[32] W. Yeong, T. Howes, and S. Kille, "Lightweight directory access protocol," 1995.

[33] J. Postel and J. K. Reynolds, "Rfc 1700 assigned numbers," *Network Working Group*, 1994.

[34] S. Das, E. Lee, K. Basu, and S. Sen, "Performance optimization of voip calls over wireless links using h.323 protocol," *Computers, IEEE Transactions on*, vol. 52, no. 6, pp. 742–752, 2003.

[35] IEEE, "Ieee draft standard for local and metropolitan area networks–station and media access control connectivity discovery," IEEE, Tech. Rep., 2009.

[36] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[37] P. Karn and C. Partridge, "Improving round-trip time estimates in reliable transport protocols," *ACM SIGCOMM Computer Communication Review*, vol. 17, no. 5, pp. 2–7, 1987.

[38] C. Partridge, "A proposed flow specification," RFC 1363, September, Tech. Rep., 1992.

[39] B. Keepence, "Quality of service for voice over ip," in *Services Over the Internet - What Does Quality Cost? (Ref. No. 1999/099), IEE Colloquium on*, 1999, pp. 4/1–4/4.

[40] J. Postel *et al.*, "Rfc 792: Internet control message protocol," *InterNet Network Working Group*, 1981.

[41] C. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *Computers, IEEE Transactions on*, vol. C-34, no. 10, pp. 892–901, 1985.

[42] R. Braden, D. Clark, and S. Shenker, "Rfc 1633: Integrated services in the internet architecture: an overview, june 1994," *Status: Informational*, 1994.

[43] E. C. Lupu and M. Sloman, "Conflicts in policy-based distributed systems management," *Software Engineering, IEEE Transactions on*, vol. 25, no. 6, pp. 852–869, 1999.