PolicyCop: An Autonomic QoS Policy Enforcement Framework for Software Defined Networks

Conference Paper · November 2013

DOI: 10.1109/SDN4FNS.2013.6702548

4 authors, including:

Shihabur Rahman Chowdhury
University of Waterloo
22 PUBLICATIONS **160** CITATIONS
SEE PROFILE

Reaz Ahmed
University of Waterloo
54 PUBLICATIONS **350** CITATIONS
SEE PROFILE

Raouf Boutaba
University of Waterloo
56 PUBLICATIONS **424** CITATIONS
SEE PROFILE

PolicyCop: An Autonomic QoS Policy Enforcement Framework for Software Defined Networks

Md. Faizul Bari, Shihabur Rahman Chowdhury, Reaz Ahmed, and Raouf Boutaba
David R. Cheriton School of Computer Science, University of Waterloo
[mfbari| sr2chowdhury | r5ahmed | rboutaba]@uwaterloo.ca

***Abstract*—Network management is becoming increasingly challenging with the relentless growth in network size, traffic volume, and the diversity in QoS requirements. Traditionally, the concept of predefined Service Level Agreements (SLAs) has been utilized to establish QoS parameters. However, state-of-the-art technologies in this area are both proprietary and inflexible. To this end, Software Defined Networking (SDN) has the potential to make network management tasks flexible and scalable, and to provide an open platform to encourage innovation. In this paper, we present *PolicyCop* – an open, flexible, and vendor agnostic QoS policy management framework targeted towards OpenFlow based SDN. PolicyCop provides an interface for specifying QoS-based SLAs and enforces them using the OpenFlow API. It monitors the network and autonomically readjusts network parameters to satisfy customer SLAs. We present experimental results to demonstrate PolicyCop's effectiveness in ensuring throughput, latency, and reliability guarantees.**

I. Introduction

Network management systems are becoming more complex and sophisticated in order to support continuously evolving online applications and their QoS requirements. A wide variety of online and real–time applications, like video streaming, video–on–demand, online interactive gaming, video conferencing, virtual collaborative environments *etc*. have emerged over the last decade. These applications impose diverse QoS requirements on latency, throughput, error-rate, jitter, and redundancy. In addition, the relentless growth in network size, sophistication and geographic span have lifted the network management challenges to the next level. The convergence of these services impose strict performance isolation with stringent and fine-grain constraints on the underlying switching fabric. As a direct consequence, network management systems are continuously challenged to satisfy these ever increasing requirements while adhering to the resource constraints.

Traditionally, the concept of predefined Service Level Agreements (SLAs) has been utilized to establish QoS parameters for traffic management. Usually, each SLA consists of a set of Service Level Objectives (SLOs). Network managers use the SLOs to derive network level policies, which are in turn translated into device level primitives (*e.g.*, forwarding rules, queue configurations, packet dropping rate, traffic shaping policies, *etc*.). Policy-based QoS management is a well investigated topic in network management. A good number of research works have proposed autonomic policy based management systems [12, 24, 28, 29, 32]. However, most of these approaches are based on either DiffServ [1] or MPLS-DiffServ [25], which come with a number of problems. Firstly, they offer static traffic classes with a coarse granularity of QoS levels. Secondly, they require installation of specialized software or hardware components in the network. Automation approaches based on specialized policy specification languages like Ponder [28, 29] can be applied to other QoS techniques like MPLS–TE. However, this kind of methods require proprietary policy servers and policy configuration agents to be deployed on network devices.

Recently Software Defined Networking (SDN) has emerged as a promising approach for providing flexible network programmability. It facilitates dynamic configuration, operation and monitoring of a network. SDN proposes to separate the network's control plane from the data plane and provide a single system image of the control plane to the data plane [13, 15]. However, the controller itself can be implemented as a distributed [11, 17, 18, 22, 30] system. This separation enables rapid network application development. OpenFlow [9] has become the *de facto* standard of communication between the control and data planes. The control plane provides a rich northbound API and a global view of the network, which provides the network operator a programmatic and elegant way of dynamically implementing a wide-range of network policies (*e.g.*, fault-tolerance, accounting, routing, security, *etc*.) and rapidly deploy new services.

In this paper, we present the design and implementation of *PolicyCop*, an autonomic QoS policy enforcement framework based on SDN. PolicyCop provides an interface for specifying QoS policies and exploits the northbound API of SDN controller to enforce

them. PolicyCop also takes advantage of the control applications to monitor the compliance of the policies and autonomically adapts the control plane rules with changing traffic conditions.

The rest of the paper is organized as follows. First we discuss our motivation in Section II, then we present related work on policy based QoS management in Section III. Next we describe the architecture of PolicyCop in Section IV. We present experimental evaluation results in Section V to demonstrate the effectiveness of PolicyCop and finally, we conclude with some future research directions in Section VII.

II. Motivation

Providing appropriate QoS to user traffic is primarily related to managing routing mechanisms, metric, and protocol updates. It may seem that the management of routing has been completely automated. But in reality, network routing is not automated despite a plethora of protocol and software. Establishing and maintaining routes still remains one of the most challenging aspects of network management [31]. Typical routing protocols running on a collection of distributed switches or routers can present obstacles when a manager needs to override the choices made by the protocol. PolicyCop coupled with the flexible programmability features offered by SDN, enables the network manager to describe his requirements as high level network-wide policies, which are implemented, monitored, and enforced by PolicyCop. Now, instead of running a distributed routing protocol that is hard to control, the switch forwarding table can be populated by the SDN controller based on the high level policies defined by the network manager. The advantages offered by PolicyCop over traditional autonomic QoS frameworks are described below:

- *Per flow control and dynamic flow aggregation:* Traditional QoS architectures aggregate traffic based on the Type of Service (TOS) field in IPv4 packets, or the Class of Traffic field in IPv6 packets. However, OpenFlow permits us to provide per flow QoS control in a scalable and flexible manner.
- *Flexible programming model:* PolicyCop has a layered architecture and the APIs between the layers are defined using JSON based RESTful API. This enables the option for using different programming languages in different layers in the framework. A network administrator can introduce new modules at any layer in PolicyCop to deploy new services as long as these modules implement the standard APIs defined by PolicyCop.
- *Dynamically configurable traffic classes:* DiffServ and MPLS–DiffServ offer a predefined number of traffic classes. In contrast, PolicyCop can define new traffic classes at runtime without requiring to bring down any network device or service.
- *Reduced operational overhead:* Traditional QoS services require a number of concurrent protocols to run for performing tasks like routing, MPLS label exchange, resource reservation, *etc*. This problem is well known as the *Protocol Clutter* problem. PolicyCop avoids this issue by mediating all communication between network services and network devices over the OpenFlow protocol. PolicyCop exploits this feature to bring a new genre of services for a network operator, *e.g.*, an operator can use PolicyCop to offer a combination of guaranteed QoS for VoIP traffic and best effort service for HTTP traffic to the same customer.
- *Ease of deployment:* PolicyCop can be deployed in a network consisting of switches from different vendors as long as all switches support the same OpenFlow version. It does not require any software or hardware modifications. Its components can be deployed on the same or different physical machines based on scalability requirements.

III. Related Work

Various policy enforcement frameworks have been proposed for adaptive and autonomic service management in QoS enabled networks [10, 28, 29]. Most of the existing works target inflexible QoS architectures like DiffServ or MPLS–DiffServ, which lack broader network picture, reconfigurability, and adaptivity [23]. Our work aims at bringing together the flexible programmability and monitoring capabilities of SDN with autonomic policy-based service management to enable a network operator to provide better service offerings.

The IETF policy working group has proposed a QoS management framework using the X.500 directory where policies are represented as *if–then–else* rules [10]. However, the network level policies cannot be directly mapped to devices. This mapping functionality has to be done by relay nodes. Moreover, the interaction between the application and the network policy have been ignored. In [28, 29] the authors propose a policy automation framework based on Ponder (a policy specification language). However, it requires specialized policy servers and installation of additional software modules in the network devices.

DiffServ based policy adaptation frameworks are introduced in [12, 24, 32]. DiffServ provides a fixed number of pre–configured traffic classes, which cannot be used in practice due to the ever changing nature of traffic. Moreover they require custom scripts to be downloaded to network devices to adapt to traffic conditions. During this download time, devices cannot provide regular traffic forwarding functionality.

There has been also some work in the SDN realm related to policy management. However, most of these works focus on either static [20] or dynamic [16, 19, 21] checking for policy rule installation in network devices. To the best of our knowledge this is the first work that explores autonomic QoS policy enforcement framework in the realm of SDN.

IV. Proposed Framework

As depicted in Figure 1(a), PolicyCop's architecture is organized in three planes: data plane, control plane, and management plane. The data plane (Section IV-A) consists of OpenFlow enabled switches. The control plane (Section IV-B) contains one or more OpenFlow controller(s). We have developed a few *control applications* that provide different control functions to the management plane. The management plane (Section IV-C) is divided into two components: (i) Policy Validator, and (ii) Policy Enforcer. The former monitors the network to detect policy violations and the later adapts control plane rules based on current network conditions and policies.

A. Data Plane

In this work, we are assuming that the underlying network is built from OpenFlow enabled switches. In this section we provide an overview of the QoS related features in OpenFlow protocol that we have used.

OpenFlow API: OpenFlow defines a traffic flow as a sequence of packets having the same 12-tuple containing switch ingress port, Ethernet MAC addresses (source and destination), Ethernet type, VLAN id, VLAN priority, IP addresses (source and destination), IP protocol, IP Type of Service (ToS) bits, and TCP/UDP ports (source and destination). These fields either contain exact values or wildcards to match a set of values. The OpenFlow specification also provides standardized APIs for the controller to manage the data plane switching fabric. Each switch connects to the controller over a secure TCP channel. The controller uses the OpenFlow API to install traffic forwarding rules in the switches' flow tables, discover network topology, monitor flow statistics, and track device up/down status.

QoS features: Starting from OpenFlow Specification 1.0 [6], packets belonging to a flow can be enqueued in a particular queue of an output port. The queue can be configured through standard protocols like SNMP, CLI and NetConf [3]. Controllers can query configuration and statistics parameters from the existing queues. These switches can rewrite the IP ToS field in the IP header, which can be used to implement QoS mechanisms like DiffServ [1]. Support for rewriting the Explicit Congestion Notification (ECN) bits has been incorporated since OpenFlow version 1.1.0 [7].

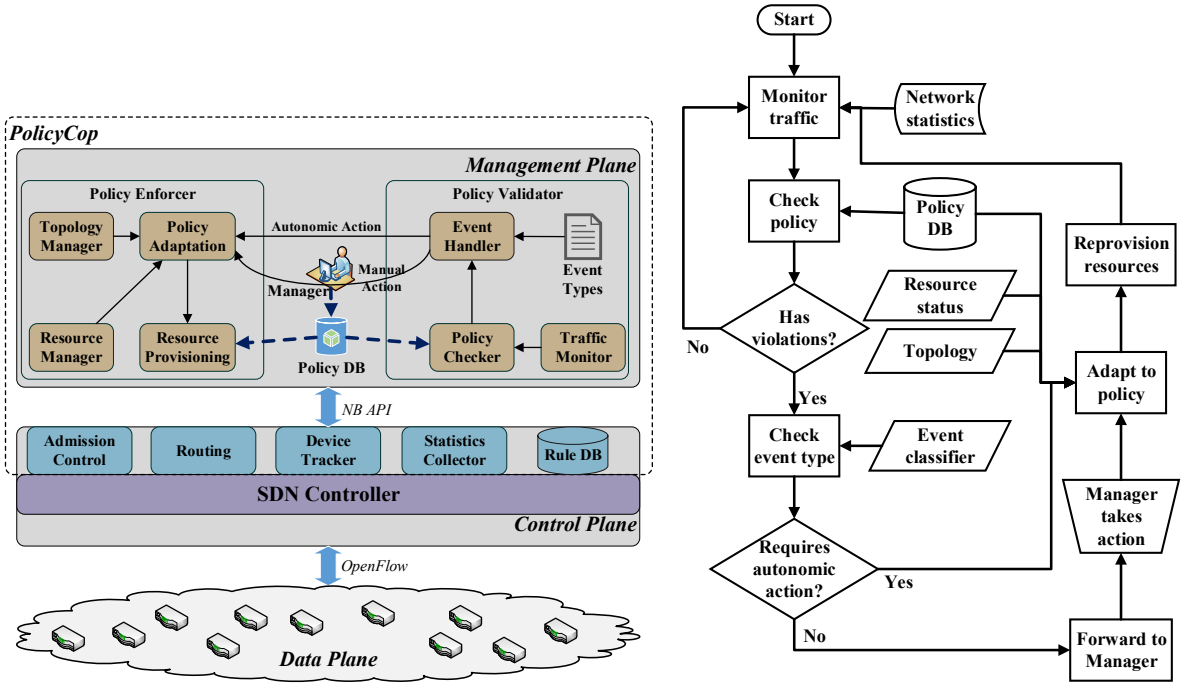
Open Networking Foundation (ONF) has created an auxiliary protocol called OF-Config [5] to support configuration of various features of an OpenFlow switch. OF-Config can be used to configure the minimum and maximum transmission rates of a queue in an OpenFlow switch. According to OpenFlow specification 1.2 [8], an OpenFlow controller can also read these rates from a switch. The most recent QoS related additions in OpenFlow are *meter tables* and *meter bands*, which can be used to limit the transmission rate of an output port. A meter band specifies a transmission rate and an actions set to be performed once the specified rate has been exceeded. Meter tables are used to store a collection of meter bands. Multiple meter bands can be associated with a single flow entry. Based of the transmission rate of the flow(s) matching this entry one or more meter band action(s) can be triggered.

In terms of monitoring, an OpenFlow controller can query a switch for statistics at different aggregation levels: table, flow, port, and queue. *Table level* statistics provide information regarding an entire flow table. *Flow level* statistics provide information about a particular flow, *e.g.*, how many bytes were matched against this flow, how many packets were forwarded, how many packets were dropped, how many errors occurred, duration for which this flow entry was active *etc*. *Port level* statistics provide more specific information about a particular port. *Queue level* statistics provide information about how many bytes and packets were enqueued at a particular queue attached to a particular output port, how many packets were dropped, duration for which this queue was active *etc*. In OpenFlow specification 1.3.0, support for querying meter level statistics was also added. *Meter level* statistics contain similar information *e.g.*, how many bytes and packets were forwarded, duration of this meter *etc*. To summarize, OpenFlow provides extensive support for configuring and monitoring QoS related features that can be used to overcome un-necessary complexities introduced by distributed routing and traffic engineering mechanisms.

B. Control Plane

The control plane consists of one or more OpenFlow controller(s) and a set of controller applications that implement different network functions, *e.g.*, admission control, device tracking, statistics collection, routing *etc*. These functions are implemented as pluggable software modules on-top of the OpenFlow controller, which typically provides either a RESTful or language specific API (henceforth referred as North Bound API or NB-API). The management plane uses these control applications to implement policy validation and enforcement. The management plane translates high-level network-wide policies to low-level rules, called *control rules* in the rest of this paper. Control rules can be used by an SDN controller to compute the FIB entries (or flow entries) for each network device.

PolicyCop requires four control applications and a database for storing control rules. These components are explained in detail below:



(a) Architecture (b) Workflow

Fig. 1. PolicyCop architecture and workflow

Admission Control: This application receives resource provisioning requests from the management plane and decides whether to accept or reject the request. It uses the SDN controller's NB-API to provision the requested resources in network devices. The NB-API can be used to reserve network resources like queues, flow-table entries, bandwidth, *etc*. If the network devices have adequate resources then the resources are provisioned and the application accepts the request from the management plane, otherwise the request is rejected.

Routing: The routing application determines path availability. It calculates route(s) based on the control rules in *Rule DB*. Suitability of a route to serve a request is determined by network topology and a collection of performance metrics like latency, throughput, error-rate, jitter and redundancy. The management plane collects these data using the *Statistics Collector* and *Device Tracker* applications.

Device Tracker: This application tracks the up/down status of network switches and their ports by listening to the asynchronous status messages exchanged between the OpenFlow controller and switches. The data collected by this application helps the management plane to maintain a global view of the network.

Statistics Collector: This application uses a mix of passive and active monitoring techniques to measure different network metrics, like bandwidth usage, residual capacity and number of dropped packets, at different aggregation levels, *e.g.*, per flow, per switch port/link, per user, *etc*. It also measures per flow latency, error–rate and jitter by inserting packet probes [26] in the network. We have developed a monitoring framework for SDN, which provides a clean an flexible way of writing monitoring applications that require such statistics in different aggregation levels [14].

Rule DB: The management plane translates high-level network-wide policies to control rules and stores them in the rule DB. The controller and other control applications (*e.g.*, routing) use these rules to compute the flow table entries for each switch.

C. Management Plane

PolicyCop's principle functionality is delivered by the management plane. PolicyCop consists of two components: (i) Policy Validator, and (ii) Policy Enforcer. These two components comprise a feedback loop for enforcing SLAs. The Policy Validator monitors the network to detect policy violations, while the Policy Enforcer adapts control plane rules based on network conditions and high-level policies. A network manager can specify network-wide policies that are stored in the *Policy DB*, which is a general purpose database for storing policies. It can be implemented using an LDAP server, a NoSQL key-value store, or a relational database. Management plane components are explained in detail below:

1) **Policy Validator:** The policy validator component periodically collects network traffic data and detects policy violations. In case of a violation, it forwards an action request to either the autonomic policy

adaptation module or the network manager based on the violation type. This component consists of three modules, which are explained below:

Traffic Monitor: This module collects the active policies from *Policy DB*, and determines appropriate monitoring interval, network segments and metrics to be monitored. Based on traffic characteristics, monitoring data might be collected more frequently from some switches compared to the others. This module utilizes the statistics collector application to collect data.

Policy Checker: The objective of this module is to identify policy violations. This module collects data from both the *Policy DB* and *Traffic Monitor*. It then analyzes the collected data to identify policy violations and forwards them to the *Event Handler*.

Event Handler: This module examines the violation events. It uses a pre-specified list of "Event Types" to determine the severity of a violation event. Depending on the event type, an action request is either forwarded to the network manager for manual action or to the policy adaptation module for autonomic action.

2) **Policy Enforcer:** The objective of this component is to re-provision network resources to adhere to the network-wide policies once the policy validator component detects a policy violation. The policy enforcer consists of the following four modules:

Topology Manager: This module collects data from the device tracker application in the control plane. It maintains a complete view of the network, which is used by the policy adaptation modules to make resource re-provisioning decisions.

Resource Manager: This module keeps track of currently allocated resources in the network using the admission control and statistics collector control applications. The data collected by this module can be used by other modules to make informed decisions when reconfiguring the network.

Policy Adaptation: The policy adaptation module consists of a set of Policy Adaptation Actions (PAAs). PAAs are distinguished by the type of metric that has been violated. A separate PAA is designed for handling each type of policy violation. For example, latency violation and throughput violation events are handled by separate PAAs. Table I shows the functionality of some example PAAs. The PAAs are pluggable components and the network manager can specify them though PolicyCop's programming interface.

Resource Provisioning: This module re-provisions network resources when a policy violation occurs. The policy adaptation module invokes this module, and provides necessary details for provisioning. The resource provisioning module either allocates more resources or releases existing ones or both based on the violation event. The policy adaptation module also provides data regarding the QoS nobs to be tuned, and where these changes should be applied.

The process workflow in PolicyCop is shown in Figure 1(b). The traffic monitoring module collects network statistics through the statistics collector application in the control plane. This data is used by the policy checker module to detect policy violations. If no violation is detected then the policy validator just keeps monitoring the network without taking any action. If a violation is detected then the event is forward to the event handler module. The event handler examines the violation event and forwards it either to the network manager or to the policy adaptation module. If the event requires manual intervention, then the network manager chooses appropriate actions based on the event, its corresponding data, and current network condition. On the other hand, if the event can be handled by the autonomic handler in the policy adaptation module, the violation event is directly forwarded to the policy adaptation module. This module determines the appropriate action based on the event type, current network topology, resource allocation, traffic condition and informs the resource provisioning module to reallocate network resources. The resource provisioning module makes the appropriate changes in the network devices to enforce the contracted policy.

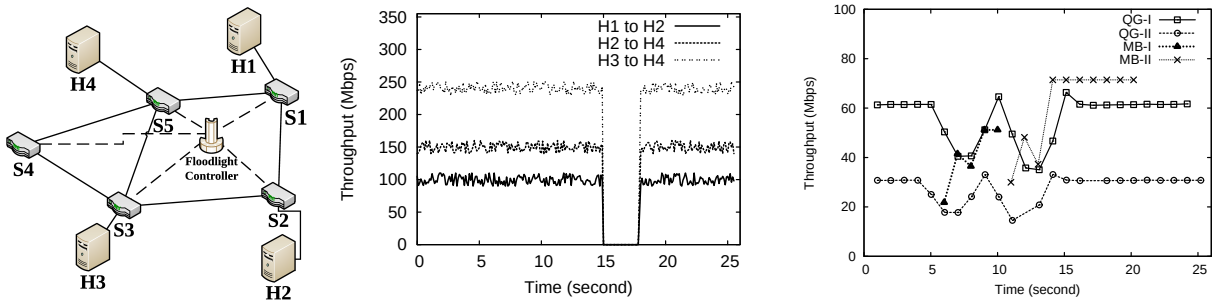
V. Experimental Evaluation

We deployed a test network consisting of five Open vSwitches [4] (OVS) and four hosts. Each OVS runs on a separate physical machine. OVS switches are inter-connected using GRE tunnels. Link bandwidth and delay are simulated using the Linux `tc` command. All OVSs are controlled by a Floodlight controller [2] as shown in Figure 2(a). The hosts run *iperf* servers and clients to generate traffic. We performed two experiments for demonstrating how PolicyCop reacts to link failure and policy violation related to throughput. In this section, we present the obtained results.

Link failure: For this experiment, we started six TCP flows between each pair of physical hosts: H1 to H2, H1 to H3, H1 to H4, H2 to H3, H2 to H4, and H3 to H4. Three flows passed through the link S3-S5: H1-S1-S5-S3-H3, H2-S2-S3-S5-H4, and H3-S3-S5-H4. Bandwidth requirements of these flows are 100 Mbps, 150 Mbps, and 240 Mbps, respectively. Now, after 15 seconds we physically disconnected the link S3-S5, which dropped the throughput to zero (see Figure 2(b)). PolicyCop detected this event as the three flows mentioned above triggered violations. These events were handled by the policy adaptation module and the three flows were re-routed though alternative links. H1-S1-S5-S3-H3 was re-routed through H1-S1-S2-S3-H3, H2-S2-S3-S5-H4 was re-routed through H2-S2-S3-S4-H4, and finally H3-S3-S5-H4 was re-routed through H3-S3-S4-S5-H4.

SLA Parameter	PAA Functionality
Packet loss	Modify queue configuration or reroute to a better path
Throughput	Modify rate limiters to throttle misbehaving flows
Latency	Schedule flow though a new path with less congestion and suitable delay
Jitter	Reroute flow though a less congested path
Device Failure	Reroute flows though a different path to bypass the failure

TABLE I. FUNCTIONALITY OF SOME EXAMPLE POLICY ADAPTATION ACTIONS (PAAS)



(a) Experimental setup (b) Effect of link failure (c) Flow throughput

Fig. 2. Throughput and bandwidth usage

Throughput: In this experiment, we started two flows with soft QoS guarantee (QG-I and QG-II) from H1 to H3 along path H1–S1–S5–S3–H3 at epoch 1s (Figure 2(c)). The maximum capacity of each link is capped at 100 Mbps using the Linux *tc* command. QG-I and QG-II were transmitting at 60 Mpbs and 30 Mbps, respectively. To introduce interference we started two misbehaving flows: MB-I from H2 to H4 at time 5s along path H2–S2–S1–S5–H4 with a duration of 6s and MB-II from H4 to H2 along the same path at time 10s with a duration of 10s. MB-I and MB-II transmit at 50 Mbps and 70 Mbps, respectively. The throughput of these four flows are shown in Figure 2(c). When MB-I started at 5 ms, the throughput of both QG-I and QG-II degraded. This was detected by PolicyCop at the 10th second (as the monitoring interval was set to 5s) and MB-I was rerouted though S2–S3–S4–S5 and QG-I and QG-II returned to their expected throughput. However, MB-II started at 10s, again reducing the throughout of QG-I and QG-II. This event was detected by PolicyCop and MB-II was rerouted to restore the throughput of QG-I and QG-II.

For our current experiments we have considered simple policy adaptation algorithms as outlined in Table I. From the results presented above, we can see that PolicyCop can detect policy violations and quickly react to overcome the problem.

VI. ACKNOWLEDGEMENT

This work was supported by the Natural Science and Engineering Council of Canada (NSERC) in part by the NSERC Discovery program and in part under the Smart Application on Virtualized Infrastructures (SAVI) NSERC Strategic Network.

VII. CONCLUSION AND FUTURE WORK

In this paper, we have presented the design of PolicyCop, an autonomic QoS policy enforcement framework for SDN. To the best of our knowledge, this is the first attempt towards an autonomic QoS management framework for SDN. We have demonstrated the effectiveness of PolicyCop through a working implementation and a set of representative experiments in real network. PolicyCop's capability in autonomic adaptation of simple QoS specifications is well reflected in the experimental results.

The next step in our work is to complete the design and implementation of all the components of PolicyCop, streamline the interfaces between different components, provide a full-fledged collection of controller applications on top of an SDN controller, and experiment on an OpenFlow testbed we have developed [27]. We envision that our work will produce a generic collection of control applications while creating an abstraction layer on top of the control platform for rapid development of network applications that rely on high level primitives.

REFERENCES

[1] DiffServ: RFC 4594. http://tools.ietf.org/html/rfc4594.

[2] Floodlight openflow controller. http://www.projectfloodlight.org/floodlight/.

[3] Network Configuration (NetConf). http://datatracker.ietf.org/wg/netconf/.

[4] Open vSwitch, An Open Virtual Switch. http://openvswitch.org/.

[5] OpenFlow Management and Configuration Protocol (OF-Config) 1.1.1. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1-1-1.pdf.

[6] OpenFlow Specification 1.0. http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf.

[7] OpenFlow Specification 1.1.0. http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf.

[8] OpenFlow Specification 1.2. https://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.2.pdf.

[9] OpenFlow Specification 1.3.0. https://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.0.pdf.

[10] N. Badr, A. Taleb-Bendiab, and D. Reilly. Policy-based autonomic control service. In *POLICY 2004*, pages 99–102, 2004.

[11] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba. Dynamic Controller Provisioning in Software Defined Networks. In *9th IEEE/ACM/IFIP International Conference on Network and Service Management 2013 (CNSM 2013)*, pages 18–25, Oct 2013.

[12] S. Cabuk, C. I. Dalton, K. Eriksson, D. Kuhlmann, H. V. Ramasamy, G. Ramunno, A.-R. Sadeghi, M. Schunter, and C. Stüble. Towards automated security policy enforcement in multi-tenant virtual data centers. *Journal of Computer Security*, 18(1):89–121, 2010.

[13] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 1–12. ACM, 2007.

[14] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba. PayLess: A Low Cost Netowrk Monitoring Framework for Software Defined Networks. In *14th IEEE/IFIP Network Operations and Management Symposium (NOMS 2014) (To appear)*.

[15] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

[16] N. Handigol, B. Heller, V. Jeyakumar, D. Maziéres, and N. McKeown. Where is the debugger for my software-defined network? In *HotSDN 2012*, pages 55–60, 2012.

[17] S. Hassas Yeganeh and Y. Ganjali. Kandoo: a framework for efficient and scalable offloading of control applications. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 19–24. ACM, 2012.

[18] B. Heller, R. Sherwood, and N. McKeown. The controller placement problem. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 7–12, 2012.

[19] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte. Real time network policy checking using header space analysis. In *NSDI 2013*, pages 99–112, 2013.

[20] P. Kazemian, G. Varghese, and N. McKeown. Header space analysis: static checking for networks. In *NSDI 2012*, pages 9–9, 2012.

[21] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey. Veriflow: verifying network-wide invariants in real time. *SIGCOMM CCR*, 42(4):467–472, Sept. 2012.

[22] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, et al. Onix: A distributed control platform for large-scale production networks. In *OSDI*, volume 10, pages 351–364, 2010.

[23] F. Le Faucheur, W. Lai, et al. Requirements for support of differentiated services-aware mpls traffic engineering. Technical report, RFC 3564, July, 2003.

[24] L. Lymberopoulos, E. Lupu, and M. Sloman. An adaptive policy-based framework for network services management. *Journal of Network and systems Management*, 11(3):277–303, 2003.

[25] I. Minei. MPLS DiffServ-aware traffic engineering. *Juniper Networks*, 2004.

[26] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk Jiri Navratil, and L. Cottrell. pathChirp: Efficient Available Bandwidth Estimation for Network Paths. In *PAM 2003*, Apr 2003.

[27] A. R. Roy, M. F. Bari, M. F. Zhani, R. Ahmed, and R. Boutaba. Design and Management of DOT: A Distributed OpenFlow Testbed. In *14th IEEE/IFIP Network Operations and Management Symposium (NOMS 2014) (To appear)*.

[28] N. Samaan and A. Karmouch. An automated policy-based management framework for differentiated communication systems. *IEEE JSAC*, 23(12):2236–2247, 2005.

[29] S. Shanbhag and T. Wolf. Automated composition of data-path functionality in the future internet. *Network, IEEE*, 25(6):8–14, 2011.

[30] 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*, 2010.

[31] Z. Wang and J. Crowcroft. Quality-of-service routing for supporting multimedia applications. *Selected Areas in Communications, IEEE Journal on*, 14(7):1228–1234, 1996.

[32] K. Yoshihara, M. Isomura, and H. Horiuchi. Distributed Policy-based Management Enabling Policy Adaptation on Monitoring using Active Network Technology. In *DSOM*, pages 265–277, 2001.