# SDN-Based Architecture for Providing QoS to High Performance Distributed Applications

**Alexandre T. Oliveira***, **Bruno José C. A. Martins***, **Marcelo F. Moreno***, **Alex B. Vieira***
**Antônio Tadeu A. Gomes**†, **Artur Ziviani**†
*Universidade Federal de Juiz de Fora (UFJF), †National Laboratory for Scientific Computing (LNCC), Brazil
Emails: {alexandre.tavares,brunomartins,moreno}@ice.ufjf.br, alex.borges@ufjf.edu.br, {atagomes,ziviani}@lncc.br

***Abstract*—The specification of quality of service (QoS) requirements in traditional networks is limited by the high administrative cost of these environments. Nevertheless, newer network paradigms, as software-defined networks (SDNs), simplify and relaxes the management of networks. In this sense, SDN can provide a simple/effective way to develop QoS provisioning. In this paper, we propose a QoS provision architecture exploiting the capabilities of SDN. Our approach allows the specification of classes of service and also negotiates the QoS requirements between applications and the SDN network controller. The SDN controller, in turn, monitors the network and adjusts its performance through resource reservation and traffic prioritization. We developed a proof-of-concept of our proposal and, our experimental results show that the additional routines present low overhead, whereas- –for a given test application— we observe a reduction of up to 47% in transfer times.**

***Index Terms*—QoS, SDN, network management**

## I. Introduction

We observe an ever-increasing number of applications that have stringent quality of service (QoS) requirements. Online healthcare systems, multimedia streaming, real-time interactive applications and parallel processing in data centers are some of the examples of applications/platforms that require QoS guarantees. Providing applications with such guarantees is still challenging to actual data networks. Indeed, most of current networks are "ossified" and present scarce resources. Turning these networks more flexible and adaptable to high performance distributed systems is important to improve the efficiency of the aforementioned applications/platforms and to allow an enhanced quality of experience (QoE) its users.

The possibilities of specifying QoS requirements in traditional networks are limited due to the complex control of these environments [1]. Modifying the existing infrastructure has a high administrative cost once the subsystem of each service component must have its resources managed by a centralized entity in order to guarantee users the QoS level each one requires. In contrast, Software-Defined Networks (SDNs) constitute an emerging paradigm that facilitates the creation and introduction of new abstractions in the network, simplifying management and facilitating its evolution [2].

One of the fundamental characteristics of the SDN concept is the decoupling of data and control planes of the network devices (switches and routers). The network intelligence is transferred to a logically centralized controller element that, through secure communication protocols, as OpenFlow [3], manages data forwarding devices. In this way, the SDN model provides a global view and an increased level of programmability of the network. Through these flexible capabilities, the network controller can optionally provide simple yet effective QoS provisioning mechanisms. E.g., the controller can perform traffic shaping, guaranteeing to sensitive application, a maximum delay, jitter or throughput.

In this paper, we present a QoS provision architecture, exploiting the SDN capabilities. The proposed model provides an efficient communication platform, by reserving resources and prioritizing data traffic. The approach consists of providing software components through which distributed applications perform function calls. The, they negotiate the parameters of the particular QoS requirements between the components of the architecture (communicating entities and SDN controllers). Classes of service and their traffic specifications are predefined in data structures generated by the network administrator and made available to some elements of the environment. The process of exchanging information of reservation request and reservation confirmation is inspired by the RSVP protocol (Resource ReSerVation Protocol [4]), used in the IntServ architecture (Integrated Services [5]). This procedure is monitored by the SDN controller, which ultimately modifies the network by allocating QoS queues to the individual packet flows at the interfaces of the forwarding devices.

Our proposal stands out for its simplicity. It just manipulates easy to obtain packet headers fields, using native actions of the OpenFlow protocol specification. In this way, the controller becomes aware of application requirements without the need for costly traffic classification methodologies, such as those based on Deep Packet Inspection (DPI) or Machine Learning (ML) [6]. The high-level functional interface offered to distributed applications incorporates a simplified method of QoS provisioning with the granularity of individual data flows, as opposed to certain traditional architectures, such as DiffServ [7], which treat aggregates of flows.

To demonstrate our proposal feasibility, we developed the architecture SDN components for a specific scenario (i.e. data file transference). We set up an SDN environment in Mininet[1]

The authors thank the support of CAPES, CNPq, FAPEMIG, FAPERJ, and FAPESP.

[1]http://mininet.org

emulator, using POX[2] SDN controller. Our evaluations shown our proposal presents negligible overhead. Moreover, given the context we explore, our approach enhanced network performance. In this case, data transference spends up to 47% less time when compared to a traditional network.

## II. Related Work

Over the years, the increasing interested in applications sensitive to QoS requirements has motivated many research works in this field. The efforts resulted in several proposals for QoS-related mechanisms and architectures, such as IntServ, DiffServ, MPLS (Multiprotocol Label Switching [8]), among others. Unfortunately, administration difficulties and the less flexible configuration of traditional networks have always hampered the widespread deployment of these models.

Due to the maturing of the SDN paradigm, the study of QoS provisioning in this context gained strength [9]. In this scope, there are in the literature solutions highly dependent on context or the specific application to be served. For example, Yu *et al.* [10] propose a solution for video streaming applications that uses adaptive routing to improve the quality of such streams over SDN. In this approach, called ARVS, the base layer packets and enhancement layer packets of video bit streams are treated separately as two levels of QoS flows. Depending on the shortest path delay variation constraints, the different flows may be isolated from each other and re-routed to new paths, so that the base layer packets will occupy the shortest path or a new path that offers available bandwidth. Thus, alternatively to what our work proposes, QoS levels are guaranteed through the selection of new feasible routes. However, the choice of alternative paths is not always possible, either by the topology or by the routing policy itself.

Diorio *et al.* [11] present and discuss the use of resources to forwarding multimedia flows with QoS support. In the network, these resources are provided by a multimedia gateway that acts as a complementary component of the OpenFlow controller and as a network gateway of the end-systems. This new element is able to identify and classify multiple multimedia traffic flows, enabling, for example, that such flows receive a differentiated treatment as to their processing and targeting. Despite the resemblance of the packets identification/classification to our work, the QoS provision proposed by Diorio *et al.* is based on traffic engineering techniques, which may also restrict its adoption in certain environments.

Humernbrum *et al.* [1] also focus on a specific application. The authors create a SDN Northbound API, so Real-Time Online Interactive Applications (ROIA) specify their dynamic network requirements and have met them at runtime. Applications as multiplayer online games demand for high QoS due its intensive and dynamic interactions. The solution in this case is similar to the two previous works [10], [11], where the SDN controller reconfigures the network, accommodating the traffic quality requirements of those applications, transmitting the sensitive data flows by a faster connection.

[2]https://github.com/noxrepo/pox

In the scope of data center networks, Afaq *et al.* [12] use rate limiting techniques to guarantee QoS to short flows often termed as mice flows. These latency-sensitive flows, such as VoIP, compete with long-lived flows also referred to as elephant flows, which are generated by backup operations, for instance. The authors use a sampling-based framework to detect these time-consuming flows and submit them to a QoS module managed by an SDN controller that routes them to paths where restricted throughput rates are applied. The QoS provision approach of this paper adopts traffic shaping techniques only on the elephant flows. Our proposal, in turn, allows the creation and molding of different levels for different flows, not being restricted to the problem reported by the authors and being applicable in other scenarios of use. In addition, our architecture predicts that the mapping between a given traffic and its respective QoS category, if any, will be performed by the final hosts.

Finally, few proposals address QoS without dependence on a target application. For example, Tomovic *et al.* [13] present the original design of an SDN/OpenFlow control environment that provides bandwidth guarantees for priority flows through rate limiting. The controller performs route calculation and resource reservation, but the creation of queues at the interfaces of the forwarding devices is static, initiated during the SDN controller bootstrap. Thus, as it will be evidenced in Section IV, this proposal can be considered less flexible than the one conceived in our architecture.

## III. Brief Background on QoS

A set of QoS requirements is a feature that varies among services, being generally a function of the type of application [14]. Two basic principles related to the QoS provision can be enumerated: (i) specification of user's requirements; and (ii) provisioning of mechanisms that ensure user's requirements. The requirements are generally specified in terms of maximum delay, statistical variation of the delay (jitter), throughput, error rate, and loss rate. Providing mechanisms that guarantee such requirements means that the specified parameters are guaranteed end-to-end to the applications along their execution.

Based on these principles, we define four generic phases for QoS provisioning: (i) service request; (ii) establishment of service agreements; (iii) maintenance of service agreements; and (iv) closure of service agreements. At the service request phase, static mechanisms, which impose a costly administrative burden, or dynamic ones, such as those using signaling protocols, may be employed. Once the request is made, an implicit agreement between the application and the network is established. Then, the network ensures the transport of the application flows, as long as the application keeps the traffic generation accordingly to what has been specified at the request. This agreement is maintained by appropriate mechanisms until its closure, which can be done also in a static or dynamic way, when the reserved resources are released.

Considering these principles and phases, the necessary functionalities for a generic QoS provisioning framework are the following: (i) parameterization of services specifies parameters

and service categories; (ii) resource sharing addresses allocation, classification, and scheduling mechanisms; (iii) resource orchestration considers negotiation, admission control, and tuning functions; and (iv) adaptation of services changes the behavior of the above mechanisms and functions.

The configuration of functions and mechanisms of QoS provision, in each network element in a conventional data network, imposes a high administrative cost. SDN, in contrast, simplify the management of these networks. At a glance, SDN refers to a network architecture based on four pillars: (i) decoupling of data and control planes; (ii) forwarding decisions based on flows; (iii) transfer of control logic to a logically centralized external controller entity; and (iv) network programming through software applications running on the top of the model [2]. Fig. 1 depicts a typical SDN architecture.

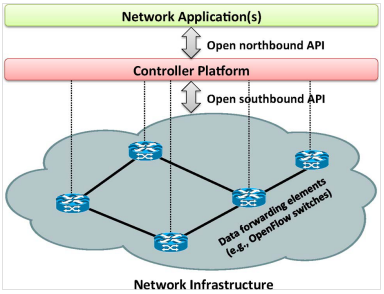



Fig. 1: Simplified view of an SDN architecture [2].

The interface between the SDN controller and the network infrastructure devices (southbound API) is provided by secure protocols, as OpenFlow [3]. Unlike the northbound API, which is not yet standardized, OpenFlow is considered a *de facto* standard. Thus, in a typical SDN environment, data forwarding elements (switches and routers) are OpenFlow enabled, i.e. they can be managed by a OpenFlow-compatible controller.

The OpenFlow API, since its version 1.0, provides native functions that enable the development of QoS provision mechanisms. One can easily implement packet allocation, classification, and scheduling. The granularity in the treatment of individual flows and the global view of SDN also enable the provision of QoS negotiation, admission control, and tuning mechanisms. These capabilities are exploited in QoS solutions that use packet re-routing and traffic balancing of sensitive flows. However, these methods are not always feasible. In many cases, the routing policy applied to the network prevents the establishment of alternative paths. In other cases, there may be no available routes, or these routes may be simply congested. In this sense, traffic shaping techniques by end-to-end bandwidth reservation and rate limitation are simple and effective QoS provisioning solutions that are capable of guaranteeing the QoS requirements of the applications.

## IV. Proposed Architecture

The proposed architecture takes into account the principles, phases, and functionalities discussed in Section III, which directly guide the design of the components and the development of the prototypes used in the evaluation. Nevertheless, it is important to emphasize that the elaboration of this architecture is restricted to the provision of quality levels in the communication network subsystem, without worrying about the QoS provision in the final hosts (input/output buffers, OS processes, protocol stack, among others), as proposed by [15]. Fig. 2 shows a high-level representation of the proposed architecture.

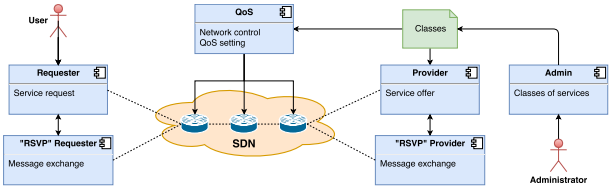



Fig. 2: Diagram of the QoS provision architecture.

The architecture consists of modules related to communicating entities (Requester and Provider, which represent the distributed applications themselves), message exchange ("RSVP" Requester and "RSVP" Provider), the SDN controller (QoS), and the network administration (Admin) along with the data structure (Classes). The requester module is responsible for requesting services from a distributed application to the provider module. The latter, besides holding the data requested by the former, has the function of verifying the possibility of offering the service with the required quality guarantees. In order to do this, the provider examines the data structure that contains the service category identification (e.g. VoIP, video, best effort) and its parameters (e.g. average and maximum throughput). The network administration module, made available to the network management, has the task of generating this database of classes of service, i.e., it implements the functionality of service parameterization. It also performs the initial configuration of network QoS policies, since OpenFlow does not have this particular capability. Message exchange modules are responsible for the procedures for requesting and confirming resource reservations. The controller module is responsible for the reservation admission control and the adjustments in the network for the QoS provision.

In the proposed architecture, distributed applications are minimally modified. First, following the query to the data structure, the provider module informs the requester module the expected QoS provision. Thus, in a timely manner, both initiate calls to the corresponding functions of the reservation request and reservation confirmation modules. The provider module also interacts with the respective "RSVP" module, indicating the service category related to the application. From these changes, a simple and efficient interface is created so that the distributed applications can request a certain QoS to the network, at the same time that it requires little intervention of the developers of these applications, being enough, to that end, to implement the queries to the classes and the calls to the functions of the message exchange modules.

The process of reservation request and reservation confirmation is inspired by the RSVP protocol, as shown in Fig. 3. The procedure is monitored by the SDN controller, which checks the packets containing the respective messages and implements

the QoS provisioning mechanisms. Initially, (i) the provider-side process sends a "PATH" message to the requester-side process together with a numerical identification corresponding to the class of service to be provided. When processing each packet containing this message, the controller module routines install the forwarding rules on the switches/routers, then establishing an end-to-end path. Then, (ii) the requester process, when receiving the "PATH" message, recognizes the class identifier and generates a "RESV" (reservation request) message by inserting a relative number to the identifier in the ToS field of the packet header. When analyzing this packet in each data forwarding element in the route between the communicating entities, the QoS routines of the controller module perform the admission control of the reservations, calculating in each port of the link the availability of bandwidth required for that class of service, which is obtained by examining the ToS field. The arrival of a "RESV" message on the provider-side indicates that the reservation is feasible throughout the route. In this way, (iii) the provider process generates a new packet with a reservation confirmation message, "RESVCONF", with the ToS field identically marked. Thus, as the controller module analyzes such a packet, it enforces QoS policies on the interfaces of the devices through which sensitive flows should be transmitted as well as configures their queues with the rules corresponding to those flows (through the OpenFlow's ENQUEUE directive). With this, the packet allocation, classification, and scheduling are guaranteed. Finally, (iv) the arrival of the "RESVCONF" message on the requester process confirms the resource reservation along the path, which enables sending the "FIN" message, closing the procedure. From this point, the provider starts sending the data flow, which will have its QoS guaranteed.

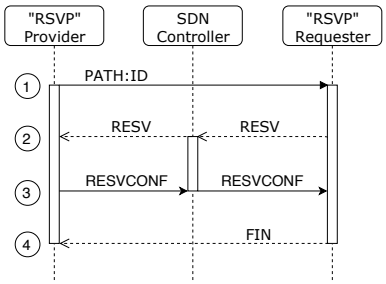



Fig. 3: Sequence diagram of reservation messages.

In the logic described above, we do not show exceptions, such as bandwidth unavailability at the time of reservation. Exceptions are treated with error messages and reserve/path depletion, such as those present in the RSVP. We also omit more complex QoS provision features, such as tuning mechanisms and service adaptation.

## V. Evaluation

To evaluate the feasibility of our proposal, we have developed a prototype of the architecture and its components[3]. This section describes the details of this evaluation.

[3]Sources available at https://github.com/netlabufjf/qos-sdn

### A. Evaluation Scenario

Fig. 4 presents the evaluation scenario considered in this work. It is based on a common SDN environment, offering a data transfer service. This simplistic scenario serves as a building block for more complex scenarios. It consists of two hosts (h1 and h2), two servers (srv1 and srv2), three routers (r1 to r3), and a network controller (c0). Routers r1 and r2 act as gateways for two different networks.

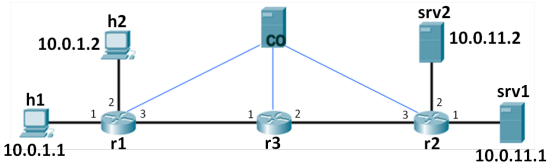



Fig. 4: Evaluation scenario.

This evaluation scenario compose a reduced yet representative data network environment that, according to previous analyses, captures all relevant aspects to our evaluation. In this scenario, applications require a given QoS level from a service provider that provides the required QoS under the control of an administrative entity. The service provider and service requester functions can be assumed by any of the hosts/servers in the same way as in a non-QoS architecture. The single-path topology between stations represents a network where methods of QoS provision by re-routing or traffic balancing are not applicable. The number of routers is justified by the average distance in hops between any hosts/servers in smaller networks, such as those found in universities or small data centers, which makes the scenario realistic.

### B. Evaluation Methodology

The network environment described in Subsection V-A was implemented in the Mininet emulator [16] version 2.3.0d1. Hosts and servers were included with the same characteristics, although they are represented graphically in different ways. Similarly, the routers were added to the topology as switches running Open vSwitch 2.0.2. POX *carp* using OpenFlow 1.0 was employed as an SDN controller. Despite the existence of newer OpenFlow versions, the 1.0 version is still the most common. The entire environment was mounted on a 64-bit Ultrabook with Intel Core i5-3317U CPU 1,70 GHz, 4,0 GB RAM, and Ubuntu 14.04 operating system. The computational resources employed were sufficient for the evaluation and did not influence the results, since memory and CPU consumptions during all tests remained around 30% and 25%, respectively.

We prepared the following test model to reproduce an environment capable of highlighting the impacts of QoS provision on concurrent flows: (i) srv1 server is responsible for data transmission; (ii) host h1 (probe) plays the role of requester, executing data requests to srv1; (iii) unidirectional constant traffic is generated between srv2 and h2; and (iv) the bandwidth of the links is limited to 1 Gbps. In this context, traffic between srv2 and h2 simulates an unsupported QoS data

flow. This flow is generated by the iPerf[4] tool with a default load (TCP at maximum rate). In turn, the data flow between srv1 and h1 is treated as sensitive. Therefore, from a request of h1, the response traffic of srv1 is classified according to a previously established quality level. Thus, during the transfer between srv1 and h1, both flows become concurrent on route r2-r3-r1, so that traffic from srv2 has its bandwidth reduced to a minimum level while the sensitive flow has its bandwidth guaranteed. In non-QoS simulations, as will be seen later, both flows share the maximum bandwidth of the links.

To implement the test model, two applications (requester and provider) were developed for the transfer of network data over TCP/IP protocols. The use of TCP is justified by the fact that this is the predominant transport protocol of the Internet. Nowadays, even delay/jitter sensitive and loss tolerant applications, such as current multimedia streaming solutions, are being deployed on this protocol, especially on adaptive streaming platforms provided by providers such as Netflix. Software components that allow the specification of the parameters and service categories in the network also were implemented. Although the architecture provides great freedom in the definition of QoS classes, in the context of the test model only two levels were required, which varied in relation to the guaranteed throughput. They could be compared, for example, to the typical levels of *controlled load* and *guaranteed service*. For the exchange of messages of quality parameters between the communicating entities, the respective modules were also developed. Finally, the POX's packet forwarding module *l3_learning*[5] was modified with the inclusion of QoS provision routines. All programming was done using the Python language.

For the design of the simulations, two response variables were defined, both measured from the point of view of the requester: (i) overhead added to the transfer time—caused by the processing of the QoS provision routines; and (ii) total data transfer time. For the transfer simulations, we used video files whose size was established as a parameter of the experiments, in this case varying in three file size levels: (i) small (44 MB); (ii) medium (128 MB); and (iii) large (435 MB). These values correspond to sizes relative to three different resolutions of the same video. From this planning, three test sessions were executed, where for each session only one file size was used. In each session three rounds of requests were performed, in which the network behaves under three different conditions: (1) SDN controller with original module (without QoS provision); (2) SDN controller with modified module and 40% bandwidth reservation for sensitive flow; and (3) SDN controller with modified module and 60% bandwidth reservation for the sensitive flow. In tests with QoS provision (cases 2 and 3) the non-QoS traffic was set up at 10% of the bandwidth, so that the data flow between srv2 and h2 fell into this category. In order to determine the impact of the experimental error, in each round 30 requests of the same file were made using the transfer applications developed, with intervals of 120 s between each request.

[4]https://iperf.fr

[5]The original module *l3_learning* allows a switch to also act as a router, routing packets between different networks.

### C. Results

Experimental results indicate our proposal presents low overhead. Indeed, the mean latency overhead situates between 1,05 s to 1,06 s, for a 95% confidence level. Fig. 5 shows the CDF of this overhead. In addition to the message exchange process itself in the reservation request phase, this measure is associated with the number of network hops. In other words,the higher the number of hops, the higher will be the overhead as the reservation procedures —executed by the controller— demands configuration of each network forwarding element. Naturally, for the used simulation scenario, this mean value has a great impact, mainly to shorter data transfers, being minimized in situations where the transmissions are more time consuming. In this respect, it is possible to evaluate that in environments where there are large files or heavy loads, such as video streaming or bulk data transfers (whose duration can reach tens of minutes), the overhead is compensated by the QoS guarantee provided to the users.

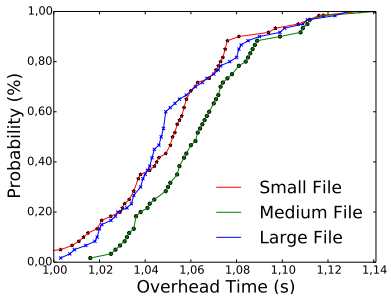
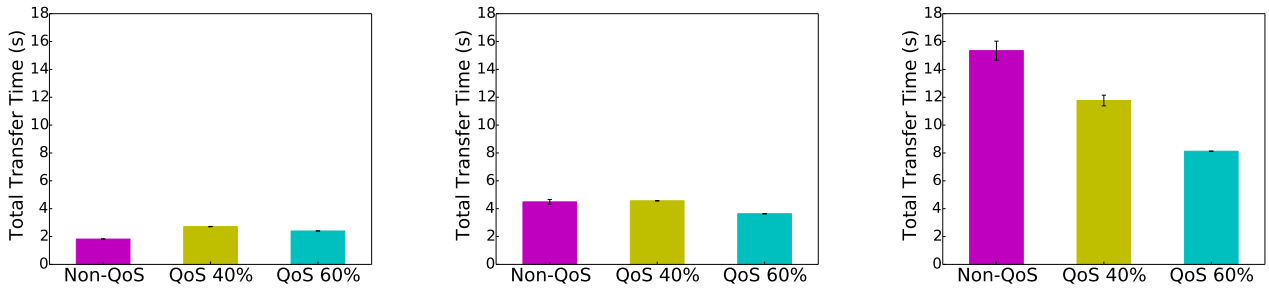



Fig. 5: Overhead distribution for each file size evaluated.

As shown in Fig. 6, we observe a significant reduction in the transmission duration, depending on the parameters and established service categories. For example, for small file transfers (Fig. 6a), a system without any QoS mechanism performs better (1,82 s) when compared with a system using the mechanism we proposed (2,71 s and 2,40 s for 40% and 60% of bandwidth reservation, respectively). As shown in Fig. 6b, the QoS mechanism outperforms a traditional system (for a 60% of QoS reservation), reducing the mean transfer time in practically 20%. Finally, the use of QoS mechanism when transferring large files (Fig. 6c) presents the best performance, enhancing transfer time in approximately 23% and 47%, for 40% and 60% of reservation, respectively. In this case, in spite of the additional overhead, the architecture of QoS provision presents a significant better performance, which corroborates its feasibility.

In a additional evaluation, a comparative analysis by paired observations was performed between the non-QoS application execution approach and the executions with resource reservation. We computed confidence intervals and mean values, for a 95% confidence level, as shown in Table I. The




(a) Small File (b) Medium File (c) Large File

Fig. 6: Average of total data transfer times.

negative measures of the confidence intervals in the "Non-QoS/QoS 40%" comparison and in the "Non-QoS/QoS 60%" comparison, using the small file, indicate that the traditional approach is superior to that with QoS provision. In the simulation session with the medium file, the value 0,0 included in the comparison interval "Non-QoS/QoS 40%" indicates that, for this configuration, the alternatives are equivalent. By increasing the reservation to 60%, the application of the architecture with QoS is already advantageous, as indicated by the interval of positive values. Finally, the comparisons from the measurements with the large file show positive intervals in both cases, which shows the superiority of the QoS provisioning approach, especially for larger files where the gain in performance compensates for the introduced overhead. This evaluation methodology certifies the previous analysis. Naturally, the particular gains are dependent on the deployed applications and the QoS specifications applied in the network.

TABLE I: Comparative analysis by paired observations.

| | Non-QoS/QoS 40% | Non-QoS/QoS 60% |
|---|---|---|
| Small File | (-0,92 , -0,87) | (-0,61 , -0,55) |
| Medium File | (-0,25 , 0,10) | (0,68 , 1,02) |
| Large File | (2,89 , 4,27) | (6,54 , 7,90) |

## VI. Conclusion and Future Work

In this work, we have presented a QoS provision architecture for high performance distributed applications exploiting the capabilities of SDN. In this architecture, simple software components enable a process of message exchanging that includes reservation request and reservation confirmation. This procedure allows a network controller to configure QoS queues to individual packet flows. We developed the prototypes of the software components and evaluated them in a reduced scenario, as a proof-of-concept. Our QoS provisioning architecture presents low overhead and, reduced the transfer time by up to 47%. Despite the specific scenario evaluated, results indicate our solution feasibility.

As future works include the development of renegotiation and adaptation functions. Moreover, we also expect to evaluate this architecture in more complex scenarios.

## References

[1] T. Humernbrum, F. Glinka, and S. Gorlatch, "A northbound api for qos management in real-time interactive applications on software-defined networks," *Journal of Communications*, vol. 9, no. 8, pp. 607–615, 2014.

[2] D. Kreutz, F. Ramos, P. Veríssimo, C. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[3] 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.

[4] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource reservation protocol (rsvp) – version 1 functional specification," Internet Requests for Comments, RFC Editor, RFC 2205, September 1997. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2205.txt

[5] R. Braden, D. Clark, and S. Shenker, "Integrated services in the internet architecture: an overview," Internet Requests for Comments, RFC Editor, RFC 1633, June 1994. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1633.txt

[6] Z. Qazi, J. Lee, T. Jin, G. Bellala, M. Arndt, and G. Noubir, "Application-awareness in sdn," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 487–488, 2013.

[7] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," Internet Requests for Comments, RFC Editor, RFC 2475, December 1998. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2475.txt

[8] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol label switching architecture," Internet Requests for Comments, RFC Editor, RFC 3031, January 2001. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3031.txt

[9] M. Karakus and A. Durresi, "Quality of service (qos) in software defined networking (sdn): A survey," *Journal of Network and Computer Applications*, vol. 80, no. Supplement C, pp. 200–218, 2017.

[10] T. Yu, K. Wang, and Y. Hsu, "Adaptive routing for video streaming with qos support over sdn networks," in *Proc. of the IEEE ICOIN*, 2015.

[11] R. Diorio and V. Timóteo, "Per-flow routing with qos support to enhance multimedia delivery in openflow sdn," in *Proc. of the Brazilian Symposium on Multimedia and the Web*, 2016.

[12] M. Afaq, S. Rehman, and W. Song, "Visualization of elephant flows and qos provisioning in sdn-based networks," in *IEEE APNOMS*, 2015.

[13] S. Tomovic, N. Prasad, and I. Radusinovic, "Sdn control framework for qos provisioning," in *Proc. of the IEEE TELFOR*, 2014.

[14] ITU-T, "Definitions of terms related to quality of service," Recommendation E800, September 2008. [Online]. Available: http://www.itu.int/rec/T-REC-E.800-200809-I/en

[15] M. Moreno, C. Neto, A. Gomes, S. Colcher, and L. Soares, "Qosos: An adaptable architecture for qos provisioning in network operating systems," *Journal of Communication and Information Systems*, vol. 18, no. 2, pp. 118–131, 2003.

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