Enabling High-Level QoS Metrics for Interactive Online Applications Using SDN

Sergei Gorlatch
University of Münster, Germany
Email: gorlatch@uni-muenster.de

Tim Humernbrum
University of Münster, Germany
Email: humernbrum@uni-muenster.de

***Abstract*—We aim at enhancing the Quality of Service (QoS) management in modern Internet applications that heavily rely on the quality of the underlying network. Our goal is to provide the application developers with mechanisms for specifying and controlling high-level, application-related QoS metrics, rather than the traditional low-level, network-related metrics like latency, throughput, packet loss, etc. We study this problem for the challenging class of Real-Time Online Interactive Applications (ROIA) which include, e.g., multiplayer online games and simulation-based e-learning and training. We leverage the dynamic management features of Software-Defined Networking (SDN) to express, monitor and control applications' QoS demands. Our particular contributions are as follows: 1) we propose a Northbound API for specifying the application-level QoS requirements in ROIA, 2) we demonstrate how the application-level metric "response time" can be automatically translated into network-level metrics understood by the SDN controller, and 3) we report experimental results on managing application-level QoS in an example online game on an OpenFlow-enabled testbed.**

I. Introduction

We consider emerging Internet-based services also known as *Real-Time Online Interactive Applications (ROIA)*. A typical ROIA application connects a potentially huge number of users who interact with each other and with the application in real time; i.e., a response to a user's action is required to happen virtually immediately. Popular representatives of ROIA are multiplayer online computer games, simulation-based e-learning and training, and serious gaming. To serve a variable, potentially very high number of interacting users, ROIA pose strong QoS requirements on the underlying network; these requirements may vary at application runtime.

Controlling the network's QoS in applications that use traditional networks is usually performed using techniques like DiffServ and IntServ [1] which provide only limited means for applications to influence the QoS. This unsatisfactory situation has been changing recently with the advent of *Software-Defined Networking (SDN)* which has opened a potential to manage the underlying network's behaviour at runtime. To achieve this, SDN decouples the control logic from the network infrastructure (switches) into a logically centralised *SDN controller* which can be accessed by applications via the so-called *Northbound API*. While the Southbound API that connects the SDN controller with the network infrastructure is relatively well understood and standardised (a well-known example is OpenFlow [2]), the design of an application-friendly, universal Nortbound API for SDN is an open problem currently gaining attention in both industry and academia.

In this paper, we address the following questions: which metrics for describing the QoS requirements should be provided to the application developer via the Northbound API and how can they be monitored and managed by the SDN controller and, eventually, by the network. Recent SDN-based approaches like Participatory Networking (PANE) [3] and PolicyCop [4] still use the traditional low-level, network-related metrics like latency, throughput, packet loss, etc. For the application developer, these QoS metrics are too low-level and not directly related to *application-level metrics* in which the application's QoS demands are usually stated, e.g., the response time or update rate. This semantic gap, caused by the different perspectives of applications and the controller on the network, is known in the network community as the so-called *application-network divide* [5]. As a consequence, application developers tend to specify significantly stricter QoS requirements than necessary and to not release reserved network capacity when possible, which can lead to an inefficient usage of the network.

In the paper, we propose a novel mechanism for using application-level, rather than network-level QoS metrics. In particular, we design and implement an SDN Northbound API for managing application-level QoS, with the following advantages:

1) Our API simplifies the specification of QoS requirements by making application-level metrics available to the application developer.
2) The application-level metrics are automatically translated by the API implementation into network-level metrics understood by the SDN controller.
3) Runtime monitoring data of the application are used in the translation process to dynamically reflect the actual application's demands in the network-level metrics.

In the following, we first describe how the QoS in ROIA can be managed using SDN. We then demonstrate how the important application-level metric "response time" in online games can be translated into traditional network-level metrics understood by the SDN controller, thus providing an illustrative and at the same time representative real-world example. Finally, we report experimental results on how this application-level QoS metric is dynamically managed in an example online game on an OpenFlow-enabled testbed.

978-1-4799-6959-3/15/$31.00 ©2015 IEEE

II. QoS for ROIA Applications

Figure 1 shows the structure of a typical ROIA (Real-Time Online Interactive Application). In the figure, we show, for simplicity, a single *ROIA Process* which serves the connected *ROIA Clients*, but real-world scenarios involve a group of ROIA Processes distributed across several server machines.

In a running ROIA, the application state is continuously updated in real time in an infinite loop, called *real-time loop*. A loop iteration (also known as *tick*) consists of three steps defined as follows. First, the clients process the users' inputs and transmit them as actions via the network to the ROIA Process (step ① in Figure 1). Then, the process calculates a new application state by applying the received user actions and the application logic to the current application state (step ②). As the result of this calculation, the states of several dynamic entities may change. The third step ③ of the loop transfers the new, updated application state to the clients. If computing the new application state takes too long, (e.g, due to a high number of connected clients or a complex application logic), or if the communication is too slow, the delivery of the state updates to the clients becomes delayed, which usually has a visible negative effect on the QoS perceived by the end-user, also known as *Quality of Experience (QoE)*.

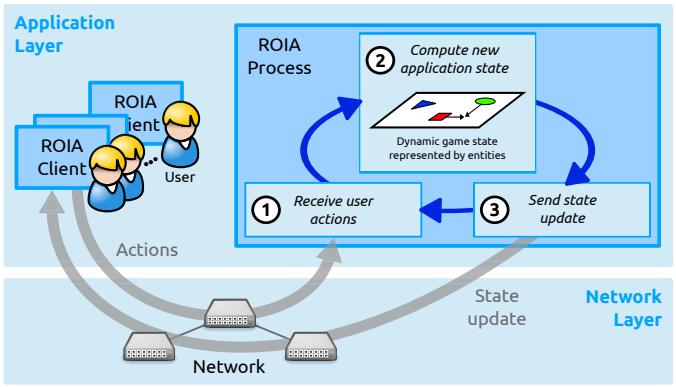

Fig. 1. Structure of a ROIA and its real-time loop.

In our recent work [6], we systematically analyse real-world application scenarios in order to specify a Northbound API for QoS management in ROIA. We design our Northbound API as consisting of two parts which reflect two very different perspectives on the network: the one of ROIA developers vs. the one of the SDN controller. While the controller can only manage network-level metrics that it is able to monitor (e.g., by evaluating the packet and byte counters of flow tables), ROIA developers often do not possess enough technical knowledge about networking details and, thus, cannot express their application-specific QoS demands using the network-specific metrics understood by the controller.

Figure 2 shows both parts of the SDN Northbound API which is realised by the so-called SDN Module:

1) The *application-level API* (① in Figure 2) enables the application developer to specify the application's QoS requirements using application- and network-level metrics.
2) The *network-level API* (② in Figure 2) offers generic network control functions to applications. This API connects the SDN controller and the SDN Module and is used by the controller for receiving network-level QoS requirements from the application and for managing QoS, e.g., rejecting requirements which cannot be accommodated. This part of our Northbound API is specified as a REST API [7] to separate the communication between the application and the controller from their specific implementation languages (e.g., C++ or Java). Furthermore, many modern SDN controllers like Floodlight [8] already provide a REST-based Northbound API and, thus, can be extended to implement our network-level Northbound API for QoS management in ROIA.

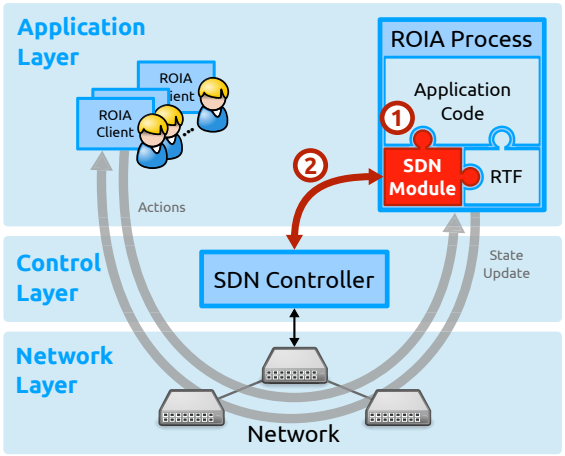

Fig. 2. Architecture of SDN-based QoS management for ROIA.

As our running example of an application-level metric we use one of the most important high-level metrics in ROIA – the *response time* – which is the time elapsed between the moment when a client starts sending user's inputs to the server and the moment when this client has completely received from the server the corresponding updates as results of those inputs. This time includes delays of the network (e.g., caused by switching and routing), as well as the processing time at the client and the server, most notably the time needed for one iteration of the server's real-time loop *(tick duration)*. Experienced players of shooter-like games notice even short delays in the application's reaction to their inputs (response time >20 ms) which make it hard to successfully aim and shoot at enemy players. If the response time is too high, the client-side extrapolation of the application state in online games (e.g., for predicting player positions, animations, and game world physics) becomes problematic: the predicted application state has to be undone if it differs from the delayed state update. Hence, the response time is viewed both by the gaming industry and by the gamers as a good indicator for the Quality of Experience (QoE) perceived by the end-user.

Listing 1 shows a small example of how the application-level part of our API is used for specifying a maximum response time for a specific flow in an application code. In

lines 2–5, the SDN Module is initialized with the address and port of an SDN controller. After activating the SDN Module, a new flow is created in lines 8–10, specified by the destination address and port, the source port, a flow label, and the direction of the flow (outgoing, incoming, or bidirectional). The creation of such a flow triggers a communication between the SDN Module and the controller, because the controller assigns a `FlowID` to the new flow that is internally used to identify this flow in all further communications with the controller. In lines 13 and 14, a QoS requirement is specified using the application-level metric `MAX_RESPONSE_TIME` (in milliseconds). This requirement is then passed to the flow's method `requestQoS` in line 17 which at runtime will invoke the translation of the response time into network-level metrics.

```
1  // initialise SDN Module
2  sdn::SDNModuleProperties props{10.10.0.254, 6063};
3  sdn::SDNModule sdn_mod{};
4  sdn_mod.registerMonitor(rtf_monitor);
5  sdn_mod.activate(props);
6
7  // create flow
8  sdn::Flow& flow = sdn_mod.createFlow(
9    10.10.0.2, 1002, 1003, 1,
10   sdn::Flow::Direction::BIDIRECTIONAL);
11
12 // specify QoS requirement
13 sdn::QoSRequirement req{
14   sdn::QoSRequirement::MAX_RESPONSE_TIME, 50};
15
16 // request for accommodation of QoS requirement
17 flow.requestQoS(req);
```

Listing 1. Specification of a QoS requirement for a specific flow.

We explain the translation from the application-level to the network-level metrics in the next section. The main feature of the translation process is that it is completely transparent for the developer and requires no further intervention. Moreover, the application's demands are continuously checked by the SDN Module using runtime monitoring data and the corresponding network-level QoS requirement is automatically updated if necessary.

III. Translating High-Level to Low-Level Metrics

Since only network-specific metrics are understood by the SDN controller, the application-level metrics which our Northbound API provides to the application developer have to be translated into network-level metrics which are then used in the low-level QoS requirement forwarded to the controller.

Our general concept for translating application-level metrics is as follows:

a) QoS requirements are specified by the developer in the application code at development time using application-level metrics of the application-level Northbound API.

b) To make the translation process adaptable at runtime, we integrate runtime application monitoring data, e.g., the tick duration, into the translation process. Which specific monitoring data are used, depends on the application-level metric that has to be translated. In the SDN Module, we provide a generic interface for querying the application at runtime for particular monitoring data. Before the application developer can use application-level metrics for specifying QoS requirements, he registers an implementation of the monitoring interface at the SDN Module as shown in line 4 of Listing 1. In our implementation and experiments, the monitoring data are provided by the *Real-Time Framework (RTF)* [9] – a C++ library for development and runtime support of ROIA that has been developed at the University of Münster. The SDN Module is directly integrated with the RTF, such that they can be used together.

c) When all information required for the translation process have been collected via the monitoring interface, the SDN Module translates the specified application-level metric at runtime into one or several network-level metrics, as we illustrate in the following.

d) The translated QoS requirement (which is now specified using exclusively network-level metrics) is serialized and transferred by the SDN Module via the network-level API to the SDN controller which then attempts to accommodate the requested network-level QoS.

Our network-level API is also used for receiving negative responses from the controller, e.g., if a QoS requirement cannot be accommodated in the network any longer.

In the following, we describe in detail how the high-level metric "response time" is translated into network-level metrics by the SDN Module and what specific monitoring information is used during the translation process.

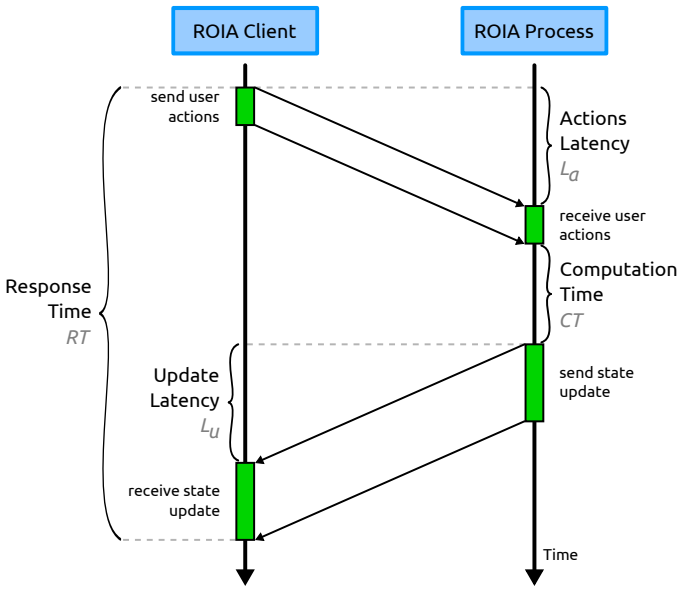

Fig. 3. Time diagram of the response time in ROIA.

Figure 3 shows our model of the response time represented as a time diagram. In particular, it shows the delays caused by the network and the application that together influence the response time (RT) in ROIA. On the network side, the response time consists of the transfer latency of the user actions (L_a), the state update latency (L_u), and the time it

takes to completely send and receive the user actions and the state update. This latter time can be calculated by dividing the data size of the actions (S_a) and the size of the state update (S_u) by the available network throughput (T). On the application side, the response time includes the computation time (CT) for updating the application state based on the received user actions.

From Figure 3, it follows that the response time in ROIA can be expressed as

$$RT = L_a + L_u + \frac{S_a + S_u}{T} + CT \qquad (1)$$

Here, CT and $S_a + S_u$ depend on the type and implementation of the application. They cannot be optimized at runtime, but rather by the application developer at development time, e.g., by using advanced data structures or a lightweight transport protocol like UDP.

However, the network latency and throughput can be managed at runtime by using SDN. Therefore, if the developer uses the application-level metric "maximum response time" for the QoS specification, the SDN Module can translate it into the network-level metrics "maximum latency" and/or "minimum throughput" as described below, taking into account the monitored computation time and data sizes at runtime.

There are three possible cases of translation using (1):

1) If the network throughput can be neglected as compared to the latency because of small data sizes, then the SDN Module translates the response time requirement into a maximum latency QoS requirement: $L_{max} = RT - CT$.
2) If the data size to be transferred is large, e.g., because of a high entity density, then the network latency can be neglected as compared to the throughput. In such a case, the SDN Module translates the response time requirement into the following minimum throughput requirement: $T_{min} = (S_a + S_u)/(RT - CT)$.
3) If neither the latency nor the transfer time can be neglected, then the SDN Module creates a combined QoS requirement for latency and throughput. In this case, a fixed maximum latency is chosen by the SDN Module and used for translating the maximum response time specified by the application developer into the minimum throughput requirement as follows:

$$T_{min} = \frac{S_a + S_u}{RT - CT - L_{max}}$$

If the controller cannot accommodate the requested QoS in terms of low-level network metrics, it sends a reject message back to the SDN Module along with a hint of what value for the corresponding low-level metric can still be accommodated in the network. In case 3, this hint is used by the SDN Module to refine the translation if possible. In the case that even the refined translation cannot be accommodated by the controller, the application is informed of the rejection of the original maximum response time requirement via a callback mechanism.

IV. Experimental Evaluation

In order to evaluate our suggested automatic translation of application-level metrics for ROIA into network-level metrics, we conduct several tests with a multi-player online game in an SDN(OpenFlow)-enabled network.

Figure 4 shows our test network topology. It consists of three software switches using Open vSwitch 1.10.2, each running on an Intel Xeon E3-1240v2 server machine with Intel Gigabit ET dual port server adapters. The game clients run on an identical machine while the game server uses a machine with two Intel Xeon E5-2620 processors and 26 GB RAM. The throughput of the connection between switches 1 and 2 is limited to 10 Mbit/s to simulate a network bottleneck, whereas the available throughput of the remaining connections is deliberately chosen to be high (1 Gbit/s). The network is controlled by our prototype implementation of an SDN controller which monitors the network utilization and attempts to adapt the network to accommodate the requested network-level QoS by optimizing routing decisions.

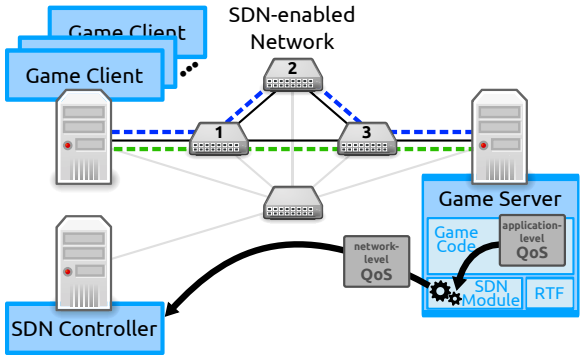

Fig. 4. Network topology used in the evaluation experiments.

Our test ROIA application belongs to the challenging class of fast-paced, first-person shooter (FPS) online games with high QoS demands and is developed using the Real-Time Framework [9]. Depending on the used server hardware, the game server can handle hundreds of clients which are playing simultaneously in the same game world. To simulate a very high number of players which can congest the network, we have implemented an automatic "bot player" that is able to automatically move in the game world and shoot at nearby players.

In order to study the effect of managing application-level QoS, we distinguish two types of game clients: premium clients (for which QoS requirements are managed) and normal clients (no QoS management is done). For every premium client, an application-level QoS requirement with a maximum response time of 50 ms is formulated. In the gaming industry, a response time up to 50 ms is considered satisfactory, because such a delay of the player's inputs cannot be noticed by the human player. It is then continuously translated at runtime by the SDN Module on the server to a network-level QoS requirement (minimum throughput) using (1) as explained above and sent to the controller via our Northbound API.

In our test scenario, the controller initially configures the network such that the switches forward all data via switch 2 in Figure 4, i.e., packets from the game clients to the game server are forwarded via switches 1, 2, and 3. At the beginning of the test, no game clients are connected to the server. Every 10 seconds, a new bot client connects to the server, alternately normal and premium clients, up to a total of 250 clients.

In Figure 5, we show the measurement results for the described test scenario. The blue/black points represent the average perceived response time of all connected normal clients. The green/grey points represent the average response time of the premium clients. Additionally, the server-side CPU load which can influence the computation time of the ROIA's real-time loop is shown by the (red) crosses.

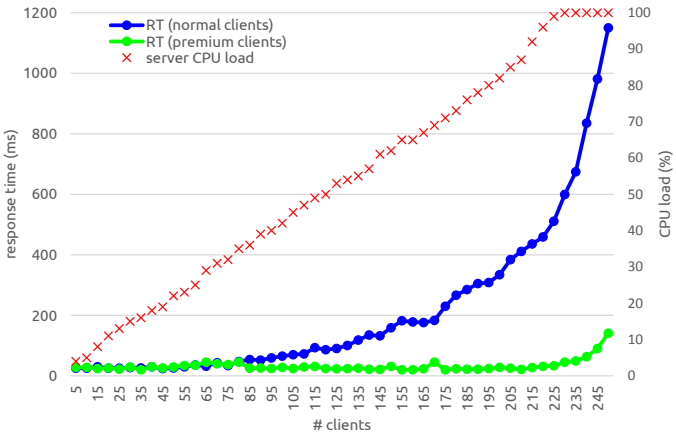

Fig. 5. Response time (RT) of normal and premium clients in FPS-like game.

We observe in Figure 5 that until 80 clients are connected, the measured response time for both normal and premium clients is below 50 ms, which is acceptable for a smooth players' QoE. For more than 80 clients, the response time starts to exceed 50 ms, which is caused by the limited throughput of the connection between switches 1 and 2. At this point, the SDN controller discovers that the needed minimum throughput value for the premium clients (which is determined by continuously translating the application-level metric "response time") can no longer be guaranteed if using this route. Hence, the SDN controller reconfigures the network and redirects the packets of the premium clients via the direct route between switches 1 and 3. Due to this reconfiguration, the response time for the premium clients remains at about 25 ms, while the monitored response time for the normal clients degrades to 600 ms for 230 clients. This demonstrates a significant advantage of our QoS management mechanism (for premium clients) as compared to running the application without QoS management (for normal clients). For even larger numbers of clients (>230), the server-side CPU load achieves 100 %; as a result of this, the response times for both normal and premium clients increase to 1150 ms and 141 ms, correspondingly, i.e. the overloaded computational resources cannot be compensated anymore by adapting the network.

V. Conclusion

Our contributions in this paper are two-fold: we show how SDN is used for an efficient utilisation of network capacity and, at the same time, how the high-level, application-related QoS metrics are made available to the application developer. We motivated and experimentally evaluated our approach using the challenging class of ROIA applications which have strong and dynamic QoS demands.

We have described our implementation of the SDN Northbound API for application-level QoS management, with the following benefits for the application developer: 1) our API simplifies the specification of QoS requirements by providing application-level metrics to the application developer; 2) the application-level metrics are automatically translated into network-level metrics understood by the SDN controller; 3) by using runtime monitoring data in the translation process, the actual application's demands are precisely and dynamically reflected in the network-level metrics; 4) if the application's demands change, the network-level QoS requirement is automatically updated, transparently for the developer.

We have integrated our approach of using application-level QoS metrics with the main software layers of the OFERTIE project [10]. On the one hand, OFERTIE's SDN controller can be used together with our Northbound API implementation to enforce the translated QoS requirements in the network using OpenFlow. On the other hand, application-level metrics like the response time can be used by OFERTIE's SLA mechanism in long-term SLAs which are agreed between an ISP and a game hoster.

VI. Acknowledgements

We are grateful to the anonymous reviewers for their helpful remarks on the preliminary version of this paper. Our research has benefited from cooperation with industrial and academic partners in the OFERTIE project (EC FP7).

References

[1] S. Alvarez, *QoS for IP/MPLS Networks*. Cisco Press, 2012.
[2] N. McKeown *et al.*, "OpenFlow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.
[3] A. D. Ferguson *et al.*, "Participatory networking: an API for application control of SDNs," in *Proceedings of the ACM SIGCOMM 2013*. ACM, 2013, pp. 327–338.
[4] M. Bari *et al.*, "PolicyCop: An Autonomic QoS Policy Enforcement Framework for Software Defined Networks," in *IEEE SDN for Future Networks and Services (SDN4FNS)*, Nov 2013, pp. 1–7.
[5] T. D. Nadeau *et al.*, *SDN: Software Defined Networks*. OReilly, 2013.
[6] S. Gorlatch *et al.*, "Improving QoS in Real-Time Internet Applications: From Best-Effort to Software-Defined Networks," in *International Conference on Computing, Networking and Communications (ICNC)*, Feb 2014, pp. 189–193.
[7] R. T. Fielding *et al.*, "Principled design of the modern Web architecture," *ACM Trans. Internet Technol.*, vol. 2, no. 2, pp. 115–150, May 2002.
[8] "Floodlight OpenFlow Controller," http://www.projectfloodlight.org.
[9] S. Gorlatch *et al.*, "Designing Multiplayer Online Games Using the Real-Time Framework," in *Algorithmic and Architectural Gaming Design: Implementation and Development*, A. Kumar *et al.*, Eds. IGI Global, 2012, pp. 290–321.
[10] "OFERTIE: OpenFlow Experiment in Real-Time Internet Edutainment," http://www.ofertie.org.