Configurable and Adaptive QoS Management via SDN

Yi Zhang

CS538 Advanced Computer Networks
Department of Computer Science
University of Illinois at Urbana-Champaign
yzhng173@illinois.edu

Qi Wang

CS538 Advanced Computer Networks
Department of Computer Science
University of Illinois at Urbana-Champaign
qiwang11@illinois.edu

Abstract

In a broadband access network (e.g., home network), the bandwidth is limited. In the meanwhile, different applications have different Quality of Service (QoS) requirements and they all compete for the scarce bandwidth. Unfortunately, many users are not savvy enough to configure QoS parameters of the underlying network to meet their needs. In this paper, we present *QoSManager*, a configurable and adaptive QoS management framework. QoSManager provides an interface for users to specify QoS requirements for different applications at runtime (configurable). It monitors the network and dynamically installs traffic shaping rules for application flows (adaptive). In QoSManager, we propose a novel algorithm that computes an assignment of flows to queues with appropriate rates to maximize the QoS requirements satisfied under the limited bandwidth. We implement QoSManager on top of Ryu and Open vSwitch (OVS) and demonstrate that it can effectively provide QoS guarantee according to users' specifications in the presence of active competing traffic.

1. Introduction

Communication networks nowadays have greatly improved the connectivity of the world. This connectivity inspires and leads to a variety of online and real-time applications, like voice over IP (VoIP), video streaming, online interactive gaming, video conferencing, etc. These applications impose diverse Quality of Service (QoS) requirements on latency, bandwidth, jitter and error-rate. On the other hand, the underlying networks can not provide unlimited bandwidth. According to [4], the average connection speed in the United States is 12.6Mbps. When users use multiple applications at the same time, these applications compete for the relatively scarce bandwidth, thus leading to degradations of overall performance.

A common approach to this problem is to prioritize applications' traffic flows so that QoS of high priority applications can be effectively enforced. To this end, many QoS mechanisms have been proposed and implemented ((**TODO: citations**)). Nonetheless, these mechanisms have not been deployed in small scale broadband access networks (e.g., home networks), for several reasons. First, many home routers have limited memory and computation resources. Therefore, they may not be able to process and enforce complicated QoS requirements "on-the-fly" ((**TODO: citations**)). Second, users' demands of QoS for different applications may change in different scenarios. For example, in a home network setting, a user may demand high definition (HD) quality of videos while no one is playing online video games. When other users are playing online video games, she is willing to accept standard definition (SD) quality of videos. However, many home users do not have knowledge to configure the underlying networks to meet their needs.

Recently Software Defined Networking (SDN) has emerged as a promising approach for providing flexible network programmability. SDN proposes to decouple the control plane from the data plane, and therefore it leaves the existing routers and switches as simple forwarding devices. The control logic is centralized

CS538 2016 Spring, University of Illinois at UrbanaChampaign.

http://dx.doi.org/10.1145/2502323.2502331

and deployed on a server, called SDN controller, which facilitates dynamic configuration, operation and monitoring of a network. **(TODO: More contents can be added here.)**

In this paper, we present QoSManager, a configurable and adaptive QoS management framework based on SDN. In QosManger, users specify high-level QoS requirements (e.g., minimum bandwidth, recommended bandwidth and priority) for different types of traffic, and QoSManager controller will dynamically assign each flow to a queue with an appropriate sending rate to maximize the QoS utility function (Section 3.3.1) under the limited link capacity. QoSManger provides an interface for modifying QoS requirements at runtime (configurable) and recompute the assignments of flows to queues when flow is added to or removed from the network (adaptive).

This paper presents several contributions. First, we design the specification language, we take into account that the QoS requirement for an application may change in different scenarios. Second, we design and implement QoSManager, a configurable and adaptive QoS management framework using SDN support. Especially, we propose a novel algorithm to assign flows to queues with appropriate rates to effectively enforce QoS. Third, we demonstrate the effectiveness of QosManager by using Mininet [3] and D-ITG [1].

The rest of the paper proceeds as follows. Section 2 presents related work in QoS, as well as SDN-based solutions for home and broadband access networks. Section 3 describes the design of our system, as well as its implementation using Ryu and Open vSwitch. Section 4 evaluates our system in the context of competing flows. Section 5 discusses future work and concludes.

2. Related work

2.1 Traditional QoS strategies

There are two types of strategies in traditional QoS: integrated services and differentiated services. Integrated services are fine-grained and per-flow.

- Every network element has to reserve resources for each flow.
- Router has limited computational resources; hard to classify all app flows.
- Not scalable.

Differentiated services [2] are more coarse-grained.

- Rely on the 8-bit DS field in the IP header. DiffServ routers then decide on per-hop basis how to forward packets based on their class.
- It is static (because of the predefined number of classes) and lacks the ability to fine-tune the QoS of separate flows.

2.2 SDN enabled QoS approach

2.2.1 Resource Reservation

FlowQoS [7] and EuQoS [8]

2.2.2 Per-flow Routing Frameworks

For Open-QoS [6], after classification, high priority flows are placed on QoS guaranteed routes.

2.2.3 Queue Management and Packet Scheduling

2.2.4 Policy Enforcement

2.3 QoS in home netowrks

Several other approaches explore QoS in home networks. Yiakoumis et al. proposed letting users notify the ISP about their bandwidth needs for a given application; in this case, provisioning occurs in the ISPs last mile, not in the home. Georgopoulos et al. proposed an OpenFlow-assisted framework that improves users quality of experience (QoE) in home networks for multimedia flows, subject to fairness constraints. The system allocates resources to each device but does not perform per-application or per-flow QoS. Mortier et al. developed Homework, a home networking platform that provides per-flow measurement and management capabilities for home networks. Homework allows users to monitor and control per-device and per-protocol usage, but it does not provide QoS support or perform any application classification.

3. QoSManager Framework

In this section, we describe the design and implementation of QoSManager. We first present an overview of the design and then describe the important components in detail.

3.1 Overview

QosManager is implemented as an application on top of Ryu SDN controller. As depicted in Figure. 1, the main components in QosManager are: **(TODO: redraw the architecture. Can refer to nmeta)**

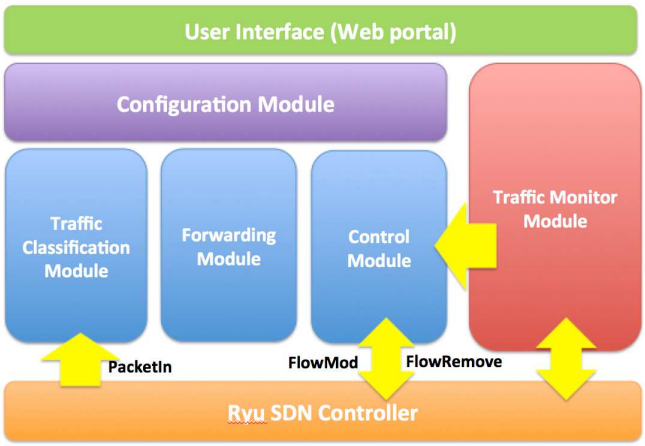

Figure 1: QoSManager architecture

- *Web Portal*: The web portal provides an interface for users to set QoS parameters for different types of traffic. In addition, users could also view the real time statistical information about the traffic flows from the web portal.
- *Configuration Module*: The QoS requirements are stored in a QoS configuration file in YAML format. Configuration module mainly handles the configuration file and passes the QoS requirements to the underlying three modules (traffic classification, forwarding and control, respectively).
- *Traffic Classification Module*: The traffic classification module classifies the application type for each flow. It maintains a lookup table *tc_table* where the key *id* is the hash value of a flow's five-tuple (e.g.,source IP address, destination IP address, protocol, source port and destination port). The value is the application type for the flow. Each table entry is also associated with a timestamp indicating when the flow starts. In the current implementation, the classification is performed by querying a statically defined database.
- *Forwarding Module*: The forwarding module is responsible for making forwarding decisions (e.g., to which port the packet should be sent). Currently, we implement a simple L2 switch. The more sophisticated forwarding module is discussed in Section 5.
- *Control Module*: The control module is the core of QosManger. It dynamically computes the assignments of flows to queues with appropriate sending rates based on the Qos requirements, the global information about the flows in the network.
- *Traffic Monitor Module*: The traffic monitor module monitors the traffic flows in the network and constantly reports the flow status information and the port status information to the web portal.

When the first packet of a flow arrives at a switch, it does not match any entries in the flow table(s) and it is forwarded to the controller. The Ryu SDN controller receives a PacketIn event and passes the packet to QoSManager. QosManager calls the traffic classification module to classify the application type for the flow and stores the result with a timestamp in a lookup table *tc_table*. QoSManager then invokes the forwarding module to make the forwarding decision. The forwarding module basically implements a simple L2 switch and maintains a *mac_to_port* table. Subsequently, QoSManager calls the control module to compute the assignments of flows to queues with appropriate rates. Based on the forwarding decision and the assignments, QoSManager directs the controller to send FlowMod messages to the switch. In addition, when a flow is removed from a switch, the controller will receive a FlowRemove event and pass the flow information to QoSManager. QosManager then invokes the control module to recompute the assignments and send FlowMod messages. Note that a flow added or removed may affect the assignments of multiple flows and thus multiple FlowMod messages will be sent to the switch.**(TODO: Add outline for subsections)**

3.2 QoS Parameters

An entry in a QoS configuration is defined as:

```
<application_type>:
  minimum: <integer> Kbps
  recommended: <integer> Kbps
  priority: <integer>
```

For each type of application (e.g., video, VoIP, gaming, P2P, etc.), users needs to configure 3 parameters. The *priority* parameter specifies the users' preference for the application. The higher the value is, the more important this type of application is. We noticed that applications such as video streaming usually have different levels of QoS. Thus, we introduce two parameters instead of one parameter to specify the requirement for an application. The *minimum* parameter specifies the minimum bandwidth the application requires. The

recommended parameter specifies the desired bandwidth for best performance. **(TODO: explain the parameters with a concrete example)**

3.3 Control Module

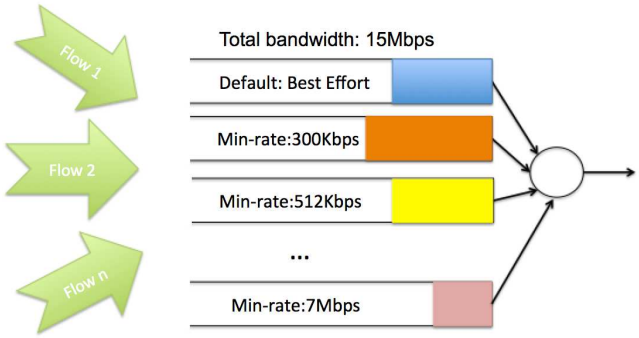

Figure 2

The control module is the core of QoSManager. The goal of the control module is to assign an appropriate rate to each flow so that QoS of high priority applications can be effectively enforced. However, existing home routers do not support per-flow rate control and even the Open vSwitch (OVS) does not yet support per-flow QoS described in OpenFlow 1.3 specification**(TODO: citation)**. To overcome this limitation, QosManager enables per-flow rate control by creating a list of queues with different sending rates at the ports of the shared link [1]. As shown in Figure. 2, suppose the link capacity of the share link is *C*, for each distinct bandwidth *B* in the QoS configuration file, we create $N = \lfloor C/B \rfloor$ queues with minimal rate B and maximal rate C. In addition, we create a special queue q_0 only with maximal rate C. At runtime, QoSManager will assign each flow to a queue with an appropriate rate to control the rate of the flow. Any flow that can not be guaranteed any level of QoS, it will be directly assigned to q_0.

To achieve the goal of the control module, we define the QoS utility function (Section 3.3.1) and propose a queue assignment algorithm (Section 3.3.2) to maximize the function under the limited link capacity.

3.3.1 QoS Utility Function

If the total bandwidth required by all the flows through a link exceeds the link capacity, it is obvious that the QoS requirements for all the flows can not be fully satisfied and we need to somehow prioritize the applications' traffic flows. In order to quantify the QoS requirements satisfied, we assign a score to each flow. The score function for a flow f is defined as:

$$score(f) = \begin{cases} priority & \text{if } bw(f) = \text{ recommended} \\ 0.6 * priority & \text{if } bw(f) = \text{ minimum} \\ 0 & \text{if otherwise} \end{cases} \tag{1}$$

The score function basically says that if a flow can not be guaranteed any level of QoS, it gets 0 point. To distinguish between different levels of QoS, we give full score to a flow if it is given recommended bandwidth and partial score to a flow if it is given minimum bandwidth. We directly use priority as score. Therefore, higher priority application flow will get higher score if it is satisfied. Based on the score function, the goal is to maximize the QoS utility function given a shared link with the capacity C:

$$\begin{aligned} U_{QoS} &= \sum_{i=1}^{n} score(f_i) \\ \text{under } &\sum_{i=1}^{n} bandwidth(f_i) \leq C \end{aligned} \tag{2}$$

It is clear that high priority application flows are more likely to be assigned their recommend bandwidth *all the time* and the QoS of low priority flows may be sacrificed. Moreover, the bandwidth allocated to high priority application will not be affected by new low priority flows going through the link. It is possible that the total score of some lower priority flows is greater than the score of a high priority flow and they cost less bandwidth. In that case, the bandwidth allocated to the high priority flow will be degraded. However, users can always tune the priority at runtime to avoid this problem.

3.3.2 Queue Assignment Algorithm

The optimization of the QoS utility function is very similar to the Knapsack problem **(TODO: citation)** which can be effectively solved by dynamic programming. However, our problem differs form the canonical bounded knapsack problem in two ways. First, in our settings, an "object" (flow) has two "values" (full and partial score) and two corresponding "volumes" (recommended and minimal bandwidth). You can only choose one of them to put into the "knapsack" (shared

[1] Here we assume that all the flows share a single link as shown in Figure. 3

link). Second, in order to avoid fluctuation, we want to keep a flow in the same queue as long as possible. That being said, when new objects come and we need to re-organize the knapsack, we want to keep as many old objects in the knapsack as possible. What's more, since the knapsack is usually much larger than the smallest object, it is not memory efficient to use the dynamic programming. Instead, we use depth-first search (Algorithm **??**) to search for the optimal assignment.

Algorithm 1: Algorithm with procedure

```
input  : A List flowList
output : A Map assign
Sort (flowList, key=λ x: x.timestamp);
Sort (flowList, key=λ x: x.priority, reverse);
best_score = 0 ;
N = flowList.length ;
dfs(0, 0, 0, newList()) ;
Procedure dfs(index, score, bw, temp_assign)
    if index > N then
        if score > best_score then
            best_score = score;
            assign = temp_assig ;
    f = flowList[index];
    if bw + f.recommend < C then
        temp_assign.append(f.id, f.recommend);
        dfs(index + 1, score + f.priority, bw +
        f.recommend, temp_assign) ;
        temp_assign.removeLast() ;
    if bw + f.minimum < C then
        temp_assign.append(f.id, f.minimum) ;
        dfs(index + 1, score + f.priority * 0.6, bw +
        f.recommend, temp_assign) ;
        temp_assign.removeLast() ;
    dfs(index + 1, score, bw, temp_assign) ;
```

The algorithm takes a list *flowList* and the shared link capacity C as input. The *flowList* merges the tc_table and the QoS configuration. We assume that all the flows in the *flowList* are successfully classified by the traffic classification module. If a flow can not be recognized by the traffic classification module, it will be directly assigned to q_0. The algorithm first sort the *flowList* by timestamp and then by priority. Assume that the sorting algorithm is table, and then we will always pick the flows in a "first come first serve" manner if there are multiple flows of the same application type. Then the algorithm invokes depth-first search procedure to search for the optimal assignment.

4. Evaluation

We now present the evaluation of our system.

4.1 Experiment Setup

Our experiments are done on a desktop PC with Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz, 16GB RAM. The network scenario is emulated using the Mininet [3] framework. It uses OS features to instantiate lightweight virtualisation of network hosts, and interconnects them with virtual switches, according to a specified topology configuration. We implemented the control application on top of Ryu [5], an open-source SDN controller.

Figure. 3 shows the experiment setup. We configuraed an Internext connection to be 15 Mbps.

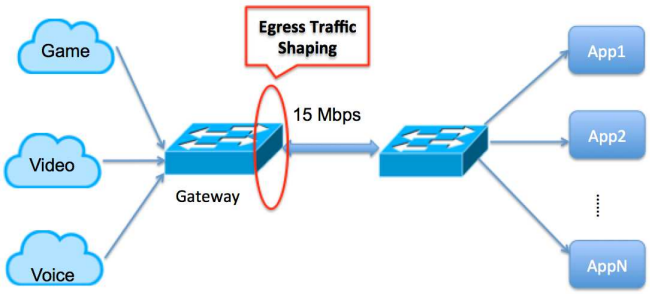

Figure 3

During the experiments, hosts request different service, such as watching a video or making a VoIP call, depending on the experiment.

We now show our system imporve the QoS of services in the face of competing traffic.

4.1.1 Scenario 1

In this scenario, one host is watching videos and the other host is downloading a file. And the priority of video is configured to be higher than file downloading. We generate traffic for the two services at the same time.

Figure illustrates the bitrates of the two services with our system, and Figure shows the bitrates of the two services withour our system.

The results show that our system allows the system to quickly converge to a higher bitrate than it otherwise would without FlowQoS enabled. This prevents bitrate oscillations and ensures the stability of the video player in terms of requested bitrates and video quality. Thus, our system improves the quality of the adaptive streaming video by both reducing bitrate oscillation and achieving a higher overall bitrate.

4.1.2 Scenario 2

The setting of this scenarios is the same as last one. But we generate traffic for file downloading first, then generate traffic for video, then we stop traffic for vidoe to see how the file downloading rate change.

4.1.3 Scenario 3

We also evaluated our system in the context of VoIP application traffic. VoIP is sensitive to delay and variation in packet arrival times, so lower jitter is essential for good performance. We monitor the packet delay and jitter of the VoIP application using ping and iperf to monitor the RTT and the packet arrival times throughout the experiment.

5. Conclusion and Future work

For the future work, we plan to add more features to our system

- Multiple path routing
- Time-based QoS
- Different device QoS

References

[1] D-ITG, Distributed Internet Traffic Generator. http://traffic.comics.unina.it/software/ITG/.

[2] Differentiated services. https://en.wikipedia.org/wiki/Differentiated_services.

[3] Mininet: An instant virtual network on your laptop (or other pc). http://mininet.org/.

[4] Q3 2015 state of the internet report. https://www.stateoftheinternet.com/downloads/pdfs/2015-q3-state-of-the-internet-report-infographic-americas.pdf.

[5] Ryu sdn framework. https://osrg.github.io/ryu/.

[6] H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp. OpenQoS: An OpenFlow controller design for multimedia delivery with end-to-end Quality of Service over Software-Defined Networks. In *Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012 Asia-Pacific*, pages 1–8. IEEE, December 2012.

[7] M. Said Seddiki, Muhammad Shahbaz, Sean Donovan, Sarthak Grover, Miseon Park, Nick Feamster, and Ye-Qiong Song. Flowqos: Qos for the rest of us. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 207–208, New York, NY, USA, 2014. ACM.

[8] Sachin Sharma, Dimitri Staessens, Didier Colle, D Palma, J Goncalves, R Figueiredo, D Morris, Mario Pickavet, and Piet Demeester. Implementing quality of service for the software defined networking enabled future internet. In *The European Workshop on Software Defined Networking, Proceedings*, pages 49–54. IEEE, 2014.