YouQoS – A new Concept for Quality of Service in DSL based Access Networks

Sebastian Meier, Alexander Vensmer, and Kristian Ulshöfer*

Institute of Communication Networks and Computer Engineering (IKR),
University of Stuttgart, 70569 Stuttgart, Germany,
{sebastian.meier,alexander.vensmer}@ikr.uni-stuttgart.de
kristian.ulshoefer@gmail.com

Abstract. Today's Internet users typically own multiple devices and consume several services simultaneously. Due to this usage pattern, bandwidth in *Access Networks* (ANs) is often insufficient. Increasing bandwidth is not always feasible or economic. A well-known approach to cope with limited bandwidth is *Quality of Service* (QoS) enforcement. However, today's QoS solutions are neither accepted by providers nor by users. Users are concerned because of network neutrality. Operators hesitate adopting QoS frameworks because they typically require end to end deployment. In this paper we present YouQoS – a solution, which addresses operators' as well as users' concerns by providing a QoS solution, which works locally in the AN based on user defined QoS policies. We introduce our evolutionary approach to QoS management in today's *Digital Subscriber Line* (DSL) based ANs by utilizing and enhancing existing QoS mechanisms. Furthermore, we provide an initial proof of concept with a Linux demonstrator.

Keywords: Quality of Service, Access Networks

1 Introduction

Bottlenecks in packet switched networks negatively influence QoS. In particular they have impact on packet loss, delay, and bandwidth. In the past, many approaches (such as IntServ [5] and DiffServ [4]) have been proposed to migrate from best effort to QoS enabled networks. QoS architectures have been deployed to some extent in ANs to manage services offered by the *Internet Service Provider* (ISP) (e.g. IP telephony). However, to the best of our knowledge these approaches have not been deployed widely for managing QoS of *Internet* traffic in the AN. One of the reasons for not adopting these approaches (in particular IntServ) is that they typically assume end to end deployment.

However, in today's networks, bottlenecks are commonly located in the ANs. Particularly in rural areas, many households are connected to the Internet with low bandwidth. A Study for DSL ANs in Germany has shown that 38% of

* At the time of writing, Kristian Ulshöfer was a student at the IKR.

subscribers are connected to the Internet with as little as 6 MBit/s. The "ICT Facts and figures" report from 2014 [9] shows that these observations can be transferred to other countries, as well.

These conditions are insufficient for today's user behavior, which is characterized by consuming several services (e.g. VoIP and Download) and using several end-devices simultaneously. We assume that this problem persists even with improving (e.g. very high speed digital subscriber lines [11]) and new AN technologies (e.g. gigabit passive optical networks [10]) as service demands will increase as well (e.g. 4K video streaming).

In contrast to previous end to end QoS approaches we propose a QoS solution, which operates locally in the AN based on user defined policies. This approach is promising, since it has several advantages for network operators as well as users (subscribers) as outlined in the following.

On the one hand, our approach supports gradual deployment as YouQoS extensions can be limited to a single AN. In the simplest case, upgrading a single AN device is sufficient. On the other hand, user satisfaction can be increased by efficient QoS management without having to increase the subscriber's bandwidth.

Furthermore, the user is able to specify the prioritization of the services she consumes. This is beneficial in two ways. On the one hand, the user knows best, which services are important compared to others. On the other hand, enabling the user instead of the service provider to influence service prioritization mitigates the problem of network neutrality.

Our solution enforces QoS on the last mile, which is usually the *Local Loop* (LL) in DSL ANs. The last mile is typically assigned to an individual subscriber. In contrast to DiffServ we therefore can trust a user's prioritization as his decisions only impact his services but never influence the services of other users.

The remainder of this paper is structured as follows. Section 2 gives an overview of the state of the art of QoS management in DSL ANs. Section 3 presents an overview of the YouQoS architecture, its functional blocks and their placement. We detail the architecture of the functional block for YouQoS Policy Selection in section 4, and its implementation in section 5. In section 6 we conclude the paper and discuss our next steps.

2 State of the Art of QoS Management in DSL ANs

Figure 1 depicts the basic structure of a DSL AN. Although we simplified the topology, it contains all relevant network elements, including the *Local Area Network* (LAN) of a residential customer.

A *Home Gateway* (HG) connects one or several *User Devices* (UDs) (e.g. PC, smart TV, mobile) to the network. Today's HGs usually deploy basic QoS mechanisms (e.g. stochastic fairness queuing [12]). Although statically configured, they guarantee some fairness between several UDs or services competing for upstream bandwidth.

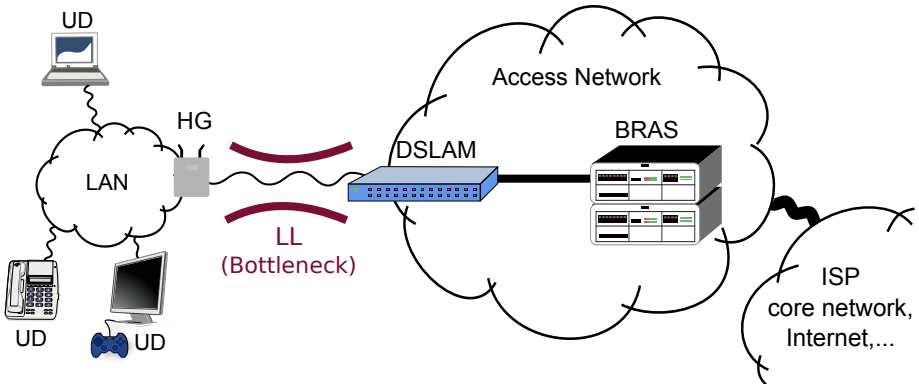


Fig. 1. DSL Access Network

A *Digital Subscriber Line Access Multiplexer* (DSLAM) terminates DSL LLs of several subscribers. Depending on the deployment model, a DSLAM connects from several dozens subscribers (decentralized outdoor deployment) up to several thousand subscribers (central office deployment) [14]. First generation DSL ANs utilized *Asynchronous Transfer Mode* (ATM) on the data link layer. Although ATM offers sound QoS mechanisms, network operators typically made little use of ATM's QoS features in DSLAMs. Today's Ethernet based DSLAMs become increasingly sophisticated. Recently, IP functionality is being added to DSLAMs. This trend is motivated by operators' demand for "*Broadband Remote Access Server* (BRAS) offloading", i.e. moving functionality from BRAS to DSLAMs. Offloading may include QoS differentiation, e.g. based on priority levels of virtual LAN tags [8].

A BRAS connects DSLAMs to the ISP's core network. It may handle up to 128.000 subscribers [13]. Since the introduction of "Triple-Play", the BRAS plays a central role in managing QoS for downstream traffic [2], [13]. Early architecture proposals [2] envisioned the BRAS to perform scheduling on packet level based on policies associated with subscriber sessions. However, these policies were rather static and only able to apply QoS differentiation to a small set of services offered by the ISP (e.g. IPTV). They didn't cover Internet traffic, which was always classified as best effort traffic. New proposals [6] identify requirements regarding extensions to the existing multi-service architecture, e.g. "QoS on Demand" and "near real time QoS changes". These proposals introduce sophisticated queuing and scheduling hierarchies within a BRAS for QoS enforcement on subscriber and even application level. These architecture extensions would allow applying QoS differentiation to Internet services and applications.

However, to the best of our knowledge, there are still plenty of open issues regarding the specification and realization of this architecture. In this paper we tend to go beyond the state of the art by providing a solution for swift determining and signaling of application QoS requirements.

3 YouQoS Architecture

This section provides an overview on the overall YouQoS architecture. We introduce its functional blocks, which are depicted in figure 2 and discuss their placement. Additionally, we introduce our approach for specifying user defined QoS policies.

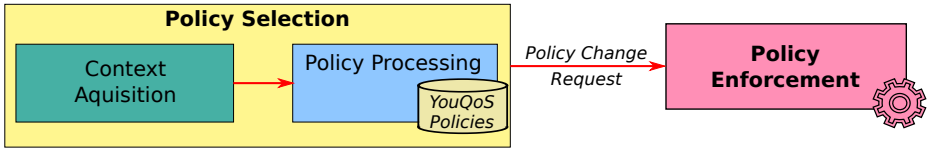


Fig. 2. Functional blocks of the YouQoS architecture

3.1 YouQoS Policies

YouQoS Policies are user defined rules for specifying the prioritization of the services that the user consumes. The underlying assumption is that there doesn't exist a "one size fits all" QoS parameterization. For instance, a gamer might be interested in prioritizing gaming traffic while a home office user might want to prioritize video conference calls.

Furthermore, we assume that QoS prioritization depends on context information. For instance, a video stream played back on a smart phone might decrease in priority, if the user places the phone screen side down on a table. Detecting this kind of dependencies automatically can be challenging if not impossible.

Therefore, our YouQoS policy approach combines context information as well as user preferences for determining QoS prioritization. To achieve this task, we propose a policy based approach. A YouQoS Policy consists of attributes and a prioritization result. Attributes may match on flow state and properties as well as device state and properties, see table 1 for examples. Currently, a policy's prioritization result encodes a relative priority.

The following subsection details, how our architecture selects YouQoS Policies and utilizes their prioritization result for policy enforcement.

Policy Attribute	**Examples**
User name	Sebastian, Alexander, Kristian
User activity	foreground tab in browser, screen saver state
Application name	Chrome, Skype
Flow information	Transport layer port, source IP address

Table 1. Attributes of YouQoS Policies (excerpt)

3.2 Policy Selection

This functional block selects YouQoS Policies based on context information. It consists of the two sub functions: *Context Acquisition* and *Policy Processing*.

Context Acquisition keeps track of all information, which is relevant to policy attributes, in particular state changes in the system. This includes system wide state information (e.g. active/visible application) on the one hand, as well as flow related state information (e.g. application name) on the other hand.

Policy Processing is triggered by Context Acquisition. Based on the trigger it searches for policies, which are sensitive to the state changes discovered by Context Acquisition. Based on matching policies the selection block generates *Policy Change Requests*, which are signaled towards the functional block *Policy Enforcement*.

A ***Policy Change Request*** specifies a QoS priority and defines, which packets should be treated with the designated priority. For the latter, a Policy Change Request provides a flow definition consisting of source/destination IP address, source/destination port, and transport layer protocol.

3.3 Policy Enforcement

The main task of *Policy Enforcement* is to differentiate the priority of flows. Policy Enforcement achieves this by creating an artificial, yet fully controllable bottleneck. With this approach, QoS differentiation can be achieved by dropping, delaying or preferring individual packets. These tasks are typically carried out by packet schedulers whose internal algorithms decide the order in which packets from each flow are transmitted on the link.

As shown in section 2, QoS adaptations become increasingly dynamic. We therefore assume that network devices (e.g. BRAS) provide interfaces for configuration and parameterization of their queues and schedulers. These interfaces may be proprietary or device specific and their realization is out of our scope.

With QoS Policy Change Requests, we provide a device independent description for signaling per flow prioritization information. Concerning packet processing on the data plane, little extensions to a network device are required. A small extension for translating between our device independent QoS Policy Change Requests and proprietary packet scheduler interfaces is sufficient.

Enforcement of QoS Policy Change Requests should only affect traffic of the user that created the request and must not interfere with the QoS perceived by other subscribers. Typically, this requirement is implicitly fulfilled by the AN topology, in which the LL is the bottleneck and other shared resources are overdimensioned. In case of oversubscribed ANs, the provider typically enforces fair sharing of available resources by traffic shaping. Regarding QoS Policy Change Requests, the network operator has to ensure, that operator defined traffic shaping has precedence over user defined policies. This ensures, that user defined QoS policies cannot impact the QoS of other subscribers.

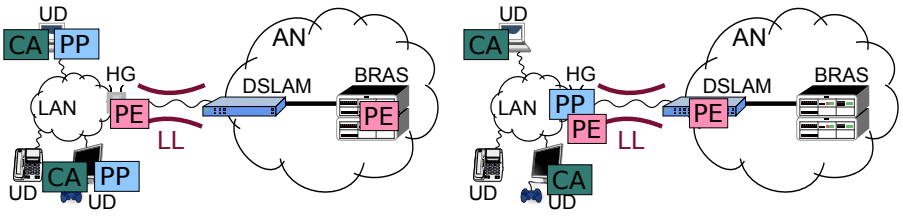
Since network operators are likely critical of user initiated QoS Policy Change Request signaling, special care has to be taken regarding the control plane.

Although control plane details are out of scope of this paper, we have identified several challenges which we intend to address, e.g.:

- *Authentication and authorization of* QoS *Policy Change Requests.* In case of DSL ANs, reuse of existing infrastructure for subscriber management seems feasible at first glance. However, a more thorough evaluation of our assumption is necessary.
- *Robustness in terms of signaling and processing load.* Network devices are typically dimensioned to handle a predefined number of policies per subscriber on the one hand, and a maximum signaling rate on the other hand. Suitable mechanisms are required to enforce these limitations, preventing a (misbehaving) user to compromise the stability of the system. For our scenario, a very simple approach for achieving this goal is discarding QoS Policy Change Requests, in case a user violates any of the aforementioned constraints.

3.4 Placement of YouQoS Functional Blocks

As figure 3 depicts, we consider the following devices for functional placement: UD, HG, DSLAM, and BRAS. Regarding the functions we differentiate between *Context Aquisition* (CA), *Policy Processing* (PP), and *Policy Enforcement* (PE). In this section we discuss, which devices are suitable for hosting which functional blocks.



(a) PP and CA on UDs, PE on BRAS and HG

(b) CA on UDs, PP on HG, PE on DSLAM and HG

Fig. 3. Functional placement examples

Context Acquisition is typically carried out on UDs, as necessary information is only available there. We acknowledge the existence of legacy devices, which might not be able to perform context acquisition themselves (e.g. IP telephones). We cover that case in our Policy Processing function, which we explain in the following.

Policy Processing can be performed on all considered devices, in principle. However, there are advantages and disadvantages regarding each choice.

Processing on the UD has the advantage that the policies are always available, independently of UD location, which facilitates using YouQoS on mobile devices.

On the other hand, keeping policies on separate UDs is cumbersome, in case a user owns several UDs (e.g. smart TV, smart phone, laptop). Furthermore, following this approach strictly, would exclude consideration of legacy devices.

Processing on the HG has the advantage that devices are managed centrally from the user's point of view. Keeping a central repository facilitates management of YouQoS Policies for devices present in a user's LAN. A HG could furthermore support management of legacy devices such as IP telephones, for instance by defining static device policies.

Processing on the DSLAM or BRAS (i.e. on devices owned by the network operator) is possible in principle. However, changes in context information may occur frequently. As these changes trigger Policy Processing, signaling these frequent changes introduces overhead. This overhead might be acceptable while being local to a device or LAN. However, we assume that the overhead becomes unacceptably high, if signaling has to traverse the LL bottleneck. Therefore, we don't consider DSLAM or BRAS as suitable candidates for Policy Processing.

Policy Enforcement has to be placed before the bottleneck. As "before" depends on the traffic direction, we discuss upstream and downstream traffic separately.

For prioritizing upstream traffic HG and UD are suitable candidates. In a LAN environment, typically multiple devices compete for bandwidth on the LL bottleneck. Therefore, policy enforcement is best placed at an element, which has full control over this bottleneck. In our scenarios, this element is usually the HG.

For prioritizing downstream traffic, BRAS and DSLAM are suitable candidates. In contrast to the upstream direction, selecting the best location is less obvious. The decision depends on factors such as network topology, network size, number of subscribers, network element capabilities, and network operator preferences.

In the next section we detail the design of our architecture's functional blocks. Since Policy Enforcement is usually a simple translation between QoS Policy Change Requests and internal scheduler interfaces (see section 3.3), we focus on the Policy Selection functional block in the following.

4 Policy Selection Block Architecture

4.1 Basic principles

Regarding the Policy Selection block, our main goal is to provide an easy to use, application independent solution.

Easy to use means that complexity is hidden from the user. We do not expect that a user knows, whether his services are sensitive to packet delay, loss, or bandwidth. Particularly, we do not want users to define QoS targets (e.g. delay, loss) on a flow basis. Instead, we intend to provide a simple interface to the user, which allows intuitive prioritization, i.e. a "priority dial" for applications. Special care has to be taken that a user can always undo any priority changes, in case they have unforeseen or unintended impact.

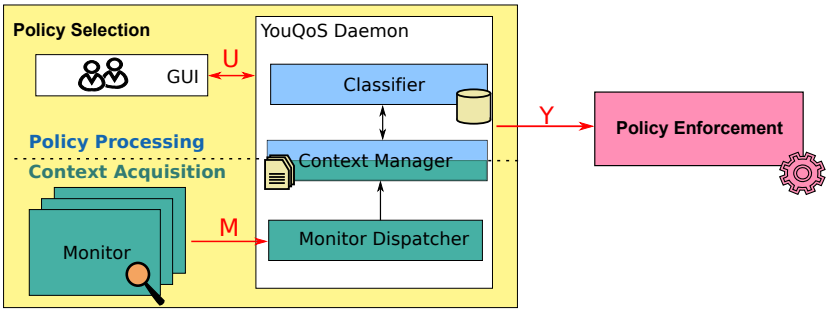


Fig. 4. Functional block for Policy Selection

Application independent means that we don't intend to patch applications for providing YouQoS functionality. Instead our approach is to utilize existing operating system interfaces for acquiring system and application state information. A *YouQoS Daemon* collects and processes this information.

The advantage of this approach is that all applications that run on a YouQoS enabled platform may benefit from our QoS policies. Although our initial design was intended for Linux, our approach is generic enough for being ported to other platforms, such as iOS or Android, for instance.

The following subsection details the internals of the YouQoS Daemon and its interaction with external entities.

4.2 Components and Interfaces

Figure 4 shows the three core components (*Monitors*, *YouQoS Daemon*, and *GUI*) as well as external interfaces (*U*, *M*, *Y*) of the YouQoS *Policy Selection* block.

Monitors are the main component of Context Acquisition. They are responsible for collecting state information related to the entire system, particular applications, or individual flows. Each individual Monitor is responsible to collect exactly one type of state information (e.g. active network flows). Monitor implementations may differ in many aspects, such as exported data (kind, frequency, amount), privileges (user privileges, root privileges), or environment (kernel space, user space). Despite those differences, the YouQoS Daemon provides a unified interface M for interacting with Monitors. Our approach was to define a simple yet flexible and extensible message based protocol for the M interface. In the current implementation of our architecture, we utilize this interface for *Inter-process communication* (IPC) between Monitors and the YouQoS Daemon. However, the design approach supports information exchange across system boundaries, in principle. This allows to carry out Context Acquisition and Policy Processing on different systems, which gives us more degrees of freedom for functional placement, see section 3.4.

The ***YouQoS Daemon*** consists of three sub-components: a *Monitor Dispatcher* for managing Monitor instances, a *Context Manager* for keeping track of state information, and a *Classifier* for selecting policies. It relies on these components to generate Policy Change Requests, which are signaled over the Y interface to the Policy Enforcement. We provide a more detailed description of these sub-components in the following.

The ***Monitor Dispatcher*** selects Monitors, which are suitable for the system environment. For instance, on a Linux system with an X Window System, the Monitor Dispatcher selects a Monitor instance for capturing X Window events (e.g. foreground window). Furthermore, the Monitor Dispatcher parameterizes properties of Monitors. Properties are typically related to the kind and amount of data a Monitor exports. For instance, a Monitor that provides flow information may be configured to not export information about short-lived flows (e.g. DNS requests).

The ***Context Manager*** creates for each application, which is managed by the YouQoS Daemon, context information necessary for policy management. This includes keeping track of and acting upon state changes. However, the Context Manager doesn't perform actions on its own. It rather creates and dispatches events based on context changes and selects suitable sub-components for event handling. For instance, a new connection typically requires classification for QoS policy selection. Therefore, in this example the Context Manager would trigger the Classifier component for further processing. Other events, e.g. when the user closes an application require policy deletion, which is handled by another sub-component (not depicted).

The ***Classifier*** has an internal database for storing user-defined YouQoS Policies. Based on trigger events from the Context Manager, the Classifier creates database requests for retrieving matching policies. A YouQoS Policy matches if and only if all of its attributes coincide with the current system state and application specific context information. In case multiple policies match, the Classifier may select a subset based on further criteria such as policy priorities, or number of exactly matching attributes. The classification result is signaled to the Policy Enforcement by an additional sub-component (not depicted).

The ***GUI*** provides user friendly access to the U interface, which allows to list, add, edit, and delete user defined YouQoS Policies. Similar to the M interface, the U interfaces relies on a simple protocol for managing YouQoS Policies. The architecture doesn't have any constraints regarding the realization of the GUI - in particular neither where nor how the GUI should be implemented.

5 Policy Selection Block Implementation

We implemented a fully functional prototype of the YouQoS Daemon for a Linux system. In addition to the daemon we also implemented two monitors and a web based graphical user interface.

For functional evaluation of interaction between Policy Selection (i.e. YouQoS Daemon) and Policy Enforcement we relied on the Linux traffic shaping frame-

work[1]. This framework is powerful enough to emulate the scheduling capabilities of a DSLAM or BRAS.

In the following we present selected components and aspects of our implementation.

5.1 Functional View

The *X Server Monitor* is realized as a user space process. By using the Xlib helper library it queries information about the graphical user interface environment from the X Server [7]. It monitors which application is running in foreground and whether the screen saver is active.

The *Connection Monitor* is implemented as a Linux kernel module. It utilizes information gathered by the Linux kernel Conntrack subsystem [15], which keeps track of a computer's connections by inspecting network packets.

We implemented the M interface, which we utilize for communication between YouQoS Daemon and Monitors by relying on Netlink [3]. Netlink provides socket oriented communication, which supports IPC between user space processes, as well as exchanging information between user space and kernel space.

The YouQoS Daemon is implemented as a user space process. For our prototype, the Classifier uses a SQL based database as a backend for storing and retrieving YouQoS Policies. We decided to provide a web based graphical user interface and therefore utilize HTTP for the U interface. This enables a user to list, add, delete, and edit YouQoS policies conveniently by using a browser. Therefore, our implementation includes a simple web server for translating between SQL database entries and Ajax enriched HTML.

5.2 Information Flow

In the following we outline interactions and information flow in our prototypic implementation of the YouQoS architecture by using a small example. We consider a UD, which hosts the YouQoS Policy Selection consisting of YouQoS Daemon, Monitors and GUI. Our starting point is that the YouQoS Daemon is running and its initialization (e.g. Monitor setup) is complete. We assume that the user previously defined two simple YouQoS Policies: a gaming policy which gives high priority to an online game application and a second policy giving low priority to FTP downloads. Now, the user starts an online game, while an FTP download is running in the background.

Figure 5 depicts the procedure from initial connection detection until Policy Change Request signaling in the YouQoS architecture.

The Connection Monitor detects a connection establishment of the online game and extracts a 6-Tuple consisting of source and destination IP address, source and destination port number, transport protocol, and process ID. It encapsulates this information in a Netlink messages and forwards this message from kernel space to the user space YouQoS Daemon.

The Monitor Dispatcher receives the message, extracts its payload and forwards the extracted state information to the Context Manager.

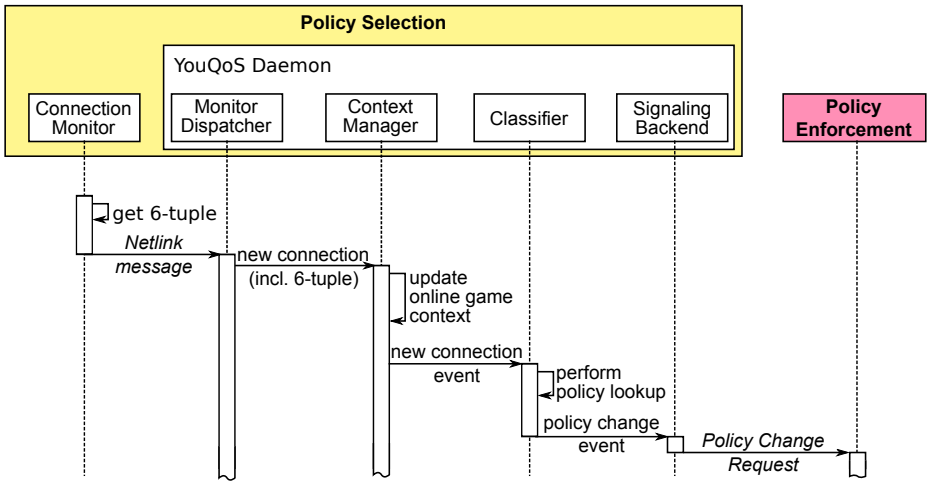


Fig. 5. Message sequence chart for triggering a Policy Change Request

The Context Manager checks, whether there already exists a context for association of the received state information. In our example that's not the case. Therefore, the Context Manager creates a new application context and assigns related state to the newly created context (e.g. connection information). Furthermore, the Context Manager performs a lookup based on the process ID for identification of the application name. Afterwards, the Context Manager creates an event which triggers the Classifier to perform a policy lookup.

For determining matching YouQoS Policies, the Classifier performs a database lookup. The database query is based on the application's context which includes information about it's connections, but also about the overall system state. In our example we assume that the Classifier finds a matching policy. It triggers the Signaling Backend to create a Policy Change Request for the online game, which is sent to the Policy Enforcement.

The Policy Enforcement point receives the Policy Change Request and extracts all required information for identifying packets belonging to the online game's connection. It parameterizes its internal packet scheduler to treat online game traffic with a higher QoS priority than packets belonging to the FTP download. Therefore, the user is able to enjoy the online game, without being impaired by the FTP transfer running simultaneously.

6 Summary and Future Work

In this paper we introduced our YouQoS architecture, its functional blocks and their placement for user defined Quality of Service enforcement in DSL access networks. Based on the current state of the art, we presented how our architecture utilizes and enhances existing QoS mechanisms. We detailed the design of

the YouQoS Policy Selection block and demonstrated its feasibility by a prototypic Linux implementation. It is worth considering, whether our approach can be transferred to other access network technologies beyond DSL, as well.

Our next steps include enhancing our Policy Change Request, which currently requests relative QoS priorities towards requesting QoS requirements in more detail (e.g. delay and bandwidth requirements). Considering the overall architecture, we intend to explore the role of the Home Gateway considering aspects such as integration of legacy devices and central Policy Selection for multiple user devices.

7 Acknowledgements

This work was funded by the Federal Ministry of Education and Research of the Federal Republic of Germany (Förderkennzeichen 16BP1211, YouQoS). The authors alone are responsible for the content of the paper.

References

1. Almesberger, W., Ica, E.: Linux Network Traffic Control - Implementation Overview (1999)
2. Anschutz, T., Allan, D., Thorne, D.: DSL Evolution - Architecture Requirements for the Support of QoS-Enabled IP Services. Tech. rep., DSL Forum
3. Ayuso, P.N., Gasca, R.M., Lefèvre, L.: Communicating between the kernel and user-space in Linux using Netlink sockets (2010)
4. Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., Weiss, W.: An Architecture for Differentiated Services (1998), `http://www.ietf.org/rfc/rfc2475.txt`
5. Braden, R., Clark, D., Shenker, S.: Integrated Services in the Internet Architecture: an Overview (1994)
6. Cui, A., Allan, D., Thorne, D.: Broadband Multi-Service Architecture & Framework Requirements. Tech. rep., DSL Forum
7. X.Org Foundation: Homepage of X.Org Project. `http://www.x.org/` (May 2014)
8. IEEE: 802.1Q-2011, Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks. IEEE Standard 802.1Q-2011 (2011)
9. ITU: The World in 2014: ICT Facts and Figures. Tech. rep., ITU
10. ITU-T: G.984.1 : Gigabit-capable passive optical networks (GPON): General characteristics. ITU-T Recommendation G.984.1 (2008)
11. ITU-T: G.993.2 : Very high speed digital subscriber line transceivers 2 (VDSL2). ITU-T Recommendation G.993.2 (2011)
12. McKenney, P.E.: Stochastic fairness queueing. In: INFOCOM'90, Ninth Annual Joint Conference of the IEEE Computer and Communication Societies. The Multiple Facets of Integration. Proceedings, IEEE. pp. 733–740. IEEE (1990)
13. Shrum, E., Allan, D., Thorne, D.: Broadband Remote Access Server (BRAS) Requirements Document. Tech. rep., DSL Forum
14. Agilent Technologies: Understanding DSLAM and BRAS Access Devices. White Paper (2006)
15. Welte, H., Ayuso, P.N.: Homepage of Netfilter project. `http://conntrack-tools.netfilter.org/` (May 2014)