BIQINI – A Flow-Based QoS Enforcement Architecture for NGN Services

Christoph Egger, Marco Happenhofer, Joachim Fabini, and Peter Reichl

Vienna University of Technology
Institute of Broadband Communications
Favoritenstr. 9/E388,
A-1040 Vienna, Austria
Telecommunications Research Center
Vienna (FTW) Donau-City-Straße 1,
A-1220 Vienna, Austria
{christoph.egger,marco.happenhofer,joachim.fabini}@tuwien.ac.at,
reichl@ftw.at

Abstract. Novel mobile cellular access network technologies like Long Term Evolution (LTE) promise capacities exceeding the ones of existing 3G networks by at least one order of magnitude. This evolution will enable the deployment of services which, due to their capacity requirements, are currently restricted to fixed access networks. On the other hand, packet-switched-only architectures raise the need for a reliable and accurate management of these high access capacities, particularly service-specific Quality of Service (QoS) enforcement, in order to prioritize real-time (voice) services and safeguard a satisfactory Quality of Experience (QoE) to the user.

In this paper we present the concept and architecture of a flow-based QoS enforcement architecture called BIQINI which has been developed at the Telecommunications Research Center Vienna (FTW). It consists of a standard-compliant Policy and Charging Rules Function (PCRF) which is supported by an emulated Policy Enforcement Function (PCEF). Extending the FOKUS OpenSource IMS testbed as well as other session-based signaling frameworks, BIQINI's emulated enforcement component enables inexpensive but highly realistic tests on real-time voice and -video traffic, supporting impairments like delay, jitter, loss, and link capacity limitation out-of-the-box. In addition, BIQINI can interface with external policy repositories, thus providing a versatile playground for testing rules and policies in an emulated, realistic environment for real media streams.

Keywords: QoS, IP Multimedia Subsystem, Policy, Policy Enforcement.

1 Introduction

Driven mainly by the high capacities available in fixed access networks, the imminent end-of-life of circuit-switched equipment, and the huge expectations concerning OPEX (operational expenses) reduction when operating one single IP- based platform, the replacement of circuit-switched voice networks by packet- switched data networks is currently experiencing a strong progress. However, the architectural requirements of

© Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2011

such large-scale carrier-grade IP-based telecommunication systems exceed the complexity of plain IETF SIP networks by orders of magnitude in order to enable satisfactory user experience already for the most important basic service, which is still voice. In this context, the IP Multimedia Subsystem (IMS), standardized by the 3rd Generation Partnership Project (3GPP), has become an important candidate architecture for the access-agnostic covering of all relevant aspects from signaling to media, from QoS reservation to security, charging and billing. Recently, several standardization bodies, namely 3GPP, 3GPP2, ETSI TISPAN and PacketCable, have joined their forces to define an interoperable Common IMS platform, agreeing to maintain one single set of 3GPP standards starting with 3GPP Release 9. Moreover, recognizing the need for interoperability and lower IMS complexity, main IMS vendors and operators have engaged in initiatives like One Voice[11] and Rich Communication Suite Initiative (RCSI)[12], defining minimum mandatory sets for IMS system- and service-level capabilities and features.

Released under the GNU Public License in 2006, the Fraunhofer FOKUS Open Source IP Multimedia Subsystem Core (OpenSource IMS) implementation has become a cornerstone of the scientific and industrial IMS research community. OpenSource IMS offers a generic, extensible 3GPP Release 7 IMS core network reference implementation, including signaling and security features as well as modules supporting the extension by means of additional interfaces (reference points). However, today's fixed and upcoming Next Generation Mobile Network (NGMN) access technologies and –services, which require network capacities exceeding the ones of existing 3G networks by an order of magnitude, mandate the use of adequate service-specific QoS enforcement mechanisms to maintain a Quality of Experience (QoE) similar to the one guaranteed by circuit-switched voice services. Despite of this urgent need, this function is not implemented by OpenSource IMS.

Therefore, this apparent gap has been addressed within the application-oriented project BACCARDI (Beyond Architectural Convergence: Charging, SeCurity, Applications, Realization and Demonstration of IMS over fixed and wireless networks) which has been conducted at the Telecommunications Research Center Vienna (FTW) during the years 2008 and 2009. As a result, the BACCARDI IMS QoS Implementation Initiative (BIQINI) has designed and implemented a QoS enforcement function which extends the OpenSource IMS by means of a generic, extensible, 3GPP Release 7 conforming Policy and Charging implementation and the corresponding interfaces. Main parts of the BIQINI concept and implementation have been contributed by the Institute of Broadband Communications (IBK), Vienna University of Technology, with support of FTW and the associated industry project partners Alcatel-Lucent Austria, Kapsch CarrierCom, mobilkom austria, and Telekom Austria.

The main aim of BIQINI is to provide a highly flexible QoS playground and multi-purpose plug-in for policy repositories, implementing a complex, stateful rules function, supporting active network capacity management as well as the PCC push model. With respect to this feature, BIQINI's Policy Enforcement component extends OpenSource IMS to become a reliable testing platform for Quality of Experience (QoE) for real-time multimedia streams. Note, however, that BIQINI does not depend on OpenSource IMS, and instead supports integration with other SIP or non-SIP session-based signaling protocols as well.

In this paper we argue that BIQINI provides a clear advancement compared to the current state of the art, most notably the open source Policy and Charging Control Framework (PCC) published by a group of the University of Capetown (UCT) [9][10] in 2007. In contrast to BIQINI, the UCT PCC operation relies on stateless gate opening and closing, moreover its architectural framework is relatively limited with respect to scalability, extensibility and active link capacity management.

Within the open source community, in November 2009 the NGN working group of Fraunhofer FOKUS has announced an Evolved Packet Core (EPC) implementation which is supposed to include a PCC. However neither the detailed concept nor the implementation of OpenEPC has been released so far. Likewise, Fraunhofer FOKUS' Policy and Charging Control Architecture (PoCCA) is maintained as closed source, being presented briefly in [3] which focuses mainly on rule processing.

To the best of our knowledge, these three contributions already conclude our account of current related work. In contrast, commercial PCC solutions are offered by various IMS vendors. However, two factors are prohibitive in deploying these implementations in applied IMS research at universities or other non-profit institutions: from a technical point of view, these implementations are closed source, which hinders additions and modifications at source code level, whereas the cost factor of PCRFs and particularly of PCEFs provides a huge barrier for IMS-related research activities.

The remainder of this paper is structured as follows: in Section 2, a brief survey of related architectures and open source routing and traffic control tools is provided. Section 3 describes the fundamental concepts and the architecture of the BIQINI implementation, whereas in Section 4 we present some results for selected traffic scenarios. Section 5 concludes the paper with a brief summary and outlook.

2 QoS Enforcement Architectures and Tools

Having recognized that the promotion of three diverging IMS standards for mobile, fixed and cable networks, respectively, entails the severe danger of an overall IMS failure, 3GPP has agreed in 2007 together with other involved standards organizations, notably ETSI TISPAN for fixed networks and PacketCable for cable networks, to harmonize their corresponding standardization activities. Starting with 3GPP Release 8, the 3GPP therefore develops and promotes a Common IMS architecture which conforms to the requirements of all three standardization bodies, whereas in 3GPP Release 7, main QoS-related interfaces (reference points), particularly Rx/Gx for 3GPP and Gq'/Re for the Resource and Admission Control Subsystem (RACS), are not yet harmonized.

As a consequence, the 3GPP Release 7 PCC compliant BIQINI architecture aims at merging the commonalities of the 3GPP and TISPAN architectures towards the framework for policy based admission control [4], which has been defined by the IETF as shown in Figure 1. Here, the Application Function (AF) is positioned within the SIP signaling path, having access to all requests for certain services along with their detailed media descriptions. The AF is responsible to query the Policy Decision Point (PDP), which decides if a specific request is granted or rejected, depending on policies, rules, request information and user profile(s). In case the service request is granted, a request with rules that should be activated is sent to the Policy Enforcement Point (PEP). The PEP enables the requested service flow according to the specifications sent by the PDP.

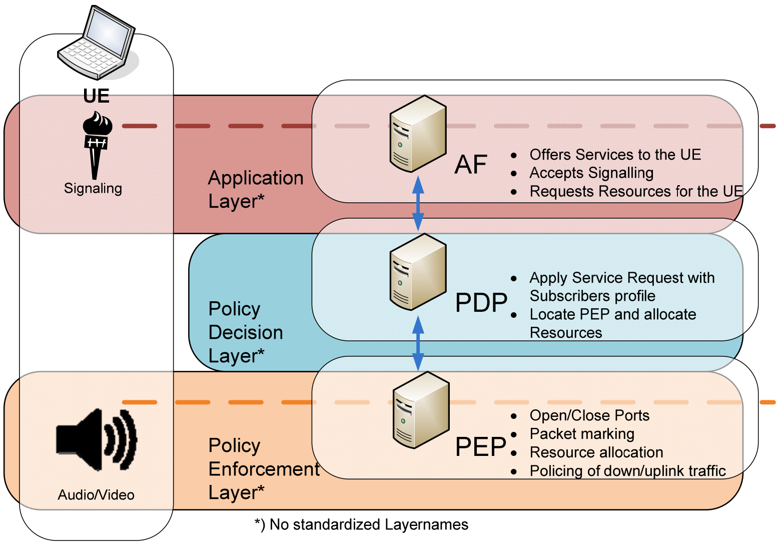
UE
Signaling
Audio/Video
Application Layer*
Policy Decision Layer*
Policy Enforcement Layer*
AF
Offers Services to the UE
Accepts Signalling
Requests Resources for the UE
PDP
Apply Service Request with Subscribers profile
Locate PEP and allocate Resources
PEP
Open/Close Ports
Packet marking
Resource allocation
Policing of down/uplink traffic
*) No standardized Layernames

Fig. 1. IETF Architecture of Policy-based Admission Control

As far as NGN QoS enforcement is concerned, the standardization bodies 3GPP, (targeting mobile networks) and ETSI TISPAN (focusing on fixed NGN networks) have designed their own architectures, depending on the particular access network requirements. In the case of 3GPP, this architecture is called Policy and Charging Control (PCC) [6]. Figure 2 depicts main functions in this architecture which can be easily mapped to layers and functions in the previously presented IETF architecture.

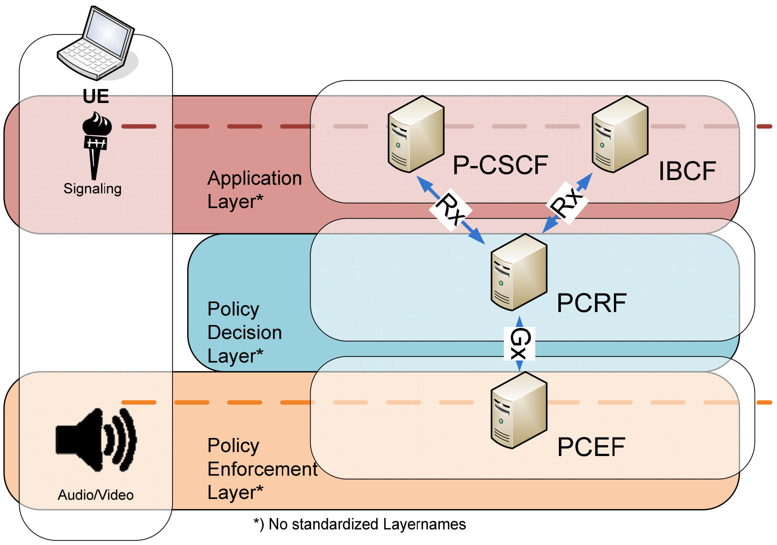

Fig. 2. 3GPP PCC Architecture

On the other hand, as mentioned earlier, ETSI TISPAN has developed its own architecture for QoS enforcement which is called Resource and Admission Control Subsystem (RACS) [7] and illustrated in Figure 3.

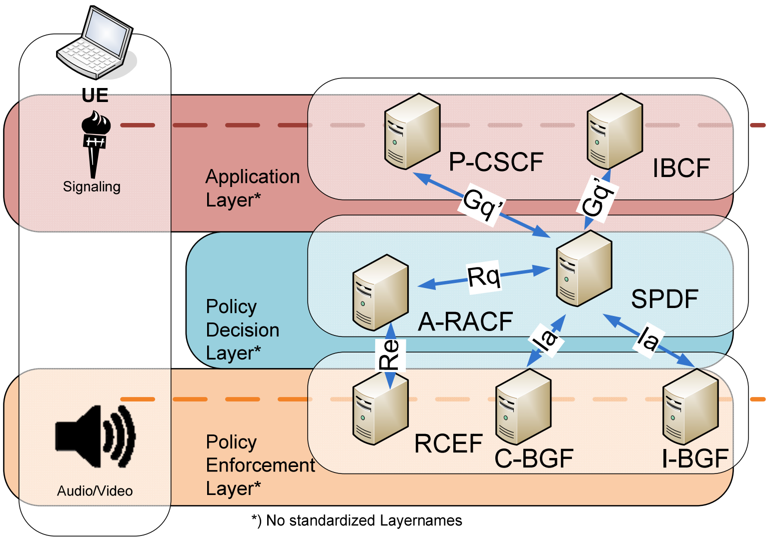

Fig. 3. ETSI TISPAN RACS Architecture

In order to merge these two architectural approaches into a common open source framework and considering the focus on access networks, BIQINI does currently not include an Interconnection Border Control Function (IBCF), nor a Core or Interconnect Border Gateway Function (C-BGF or I-BGF). Charging functionality has been included but interfaces have not been implemented yet. Furthermore we have decided to integrate and harmonize PCRF, Service Policy Decision Function (S- PDF) and Access Resource and Admission Control Function (A-RACF) into one component, namely the Policy Decision Point (PDP). ETSI-defined Resource Control Enforcement Function (RCEF) and 3GPP-specific PCEF functionalities have been merged into a Policy Enforcement Point (PEP) component. The resulting architecture is aligned to the IETF recommendation [4] based on three components: AF, PDP and PEP.

BIQINI is heavily relying on advanced routing and traffic control tools provided by the open source operating system Linux, most notably several system tools which support IP traffic queuing configuration. A detailed description of routing, switching, bandwidth management, queuing and IP security functions in Linux systems can be found in [5], in the rest of this section we will only provide an in-depth view on the default Linux command-line application for queuing configuration which is called traffic control (tc) and supports the modification of the queuing strategies to be used for outgoing and incoming IP packets on specific network interfaces.

In tc, the different queuing types and –strategies are denoted as queuing disciplines. After setting up new queuing disciplines, IP packets must be assigned to certain queues using so-called filters which match the IP packets against specific patterns, corresponding to certain fields or byte sequences in the packet. Examples of these patterns include, e.g., source IP address, ports or any other field in the IP header. Upon successful match, the specific IP packet will be assigned to the corresponding queue. Note that a queuing discipline can, for instance, realize bandwidth management, reorder packets, delay packets, modify packets, etc., depending on the selected queuing discipline. A list of supported and implemented queuing algorithms can be found in [5].

The BIQINI implementation combines several queuing disciplines to realize QoS enforcement. The classful Hierarchical Token Bucket (HTB) algorithm manages the reserved bandwidth by allocating requested bandwidth to microflows. The queuing discipline dsmark manipulates the DSCP field of IP packets, marking all IP packets queued in a specific dsmark queue using a specified DSCP value.

BIQINI uses the queuing discipline netem for implementing realistic access network emulation. The netem algorithm can delay, reorder, drop, and duplicate IP packets. By setting up several netem queuing disciplines, with different delay and loss values for distinct DSCP marking emulates a DiffServ[2] network.

Finally, selective rejection or dropping of IP packets can be configured using the ipTables utility. A rule in ipTables describes traffic patterns and defines corresponding actions for this traffic, e.g., drop, reject, or accept. The traffic can be categorized by means of ports, addresses, protocol numbers, flags, etc. Similar to common firewalls, ipTables supports default rules for packets that cannot be matched to a rule. In most cases it is preferable to drop or reject all traffic that is not explicitly accepted by a rule.

3 BIQINI – Architecture and Basic Concepts

Based on the survey of related architectures provided in the previous section, we will now present the key concepts and the resulting architecture adopted for the BIQINI QoS enforcement framework in detail.

3.1 Architecture

The basic architecture of the BIQINI implementation is sketched in Figure 4. In the depicted scenario, the QoS enforcement is applied on an access link which on both ends is protected by respective PEPs. Whereas PEP 1 on the user side ensures that the micro-flow from the user agent to the core is scheduled correctly such that the service requirements (e.g. bandwidth) are fulfilled, PEP 2 on the core side is performing the same task for the opposite traffic direction. Note that, if PEP 1 were not installed, the user could use the access link excessively with other services, for instance sending extremely large emails or uploading huge amounts of data. Such bandwidth consuming services could then severely reduce the quality of real time services like voice communication. Thus, PEP 1(included in e.g., Customer Premises Equipment (CPE)) ensures proper bandwidth usage in the uplink direction. The finding that a layer-3-QoS enforced access links must be protected on both ends (therefore extending the 3GPP and TISPAN architectures) is an essential outcome of the BIQINI project.

Otherwise QoS enforcement cannot protect uplink and downlink simultaneously, bearing the risk of QoS degradation. This architecture is similar to [1], which covers a study of enforcing QoS on Customer Premise Networks (CPN) and supports the realistic asymmetrical modeling of impairments on access links.

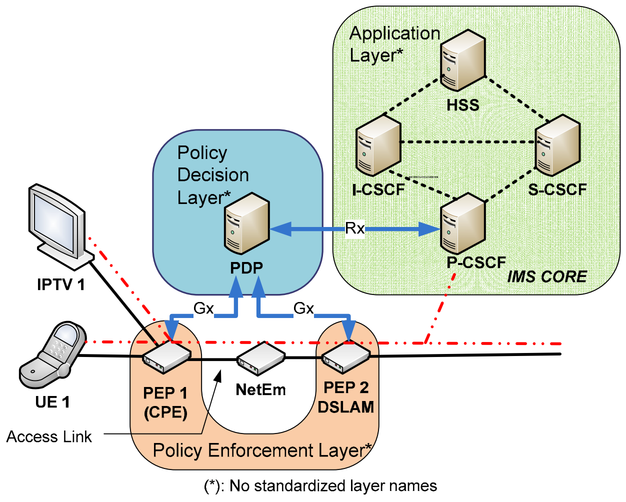

Fig. 4. Architecture of BIQINI QoS Enforcement for Access Links

When no GGSN or DSLAM is available, the characteristics of a specific access network (e.g., ADSL or 3G) can be accurately emulated from the point of view of a layer 3 protocol with the NetEm instance depicted in Figure 4 [14]. This is done by means of the Linux netem queuing discipline.

In order to differentiate between different service classes, we also have installed a netem instance on the PEPs. To be more specific, a microflow of the class “realtime conversational” needs low delay values and loss rates. In this case, our PEPs have to guarantee that the service receives the correct QoS on the access link. To realize this, the PEPs have to mark the IP packets with the correct DSCP value corresponding to this service class and handle it correspondingly. When the packets traverse the core network, DiffServ enabled router can determine which service class is to be used for a specific packet. As an example, we suggest to use the DSCP class 0x03 for realtime conversations. Thus, both PEPs have to mark packets of this class with the DSCP value 0x03, and at the same time the netem at the access link has to assign packets with DSCP value 0x03 to the queuing system handling realtime conversation, which, on its part, must realize low delay values and loss rates. On the other hand, best effort traffic could receive for instance the DSCP marking 0x00, which causes netem to treat such IP packets with a delay of several hundreds of msec and loss rates of 2% and beyond.

In our overall QoS architecture, the PEPs are responsible for realizing bandwidth management. To this end, each incoming flow is shaped according to the installed rules. Flows that utilize too much bandwidth are queued at the PEP, thus increasing the corresponding delay value. In the worst case, this may lead to dropping the packets as soon as the queue is filled.

Our PEPs can be configured with or without ipTables (see section 2). In the case of a configuration without ipTables, any traffic is admitted but marked at the PEP as best effort traffic. Therefore, netem treats this traffic with lowest quality. However, if services are enforced through the PDP, they are marked at the PEP with a different ("better") DSCP value and are treated with higher priority. Additionally, the PEP also ensures that the enforced services can utilize their required bandwidth.

The communication between AF, PDP and PEP is realized using the Diameter [8] protocol. More specifically, the AF utilizes AA-Requests (AAR) and AA Answer (AAA) messages over the Rx interface to transport authorization requests to the PDP, whereas a Session Termination Request (STR) terminates a session. The communication between PDP and PEP is realized over the Gx interface and uses Re- AuthRequests (RAR) and Re-Auth Answer (RAA) messages. Figure 5 illustrates the Diameter messages employed.

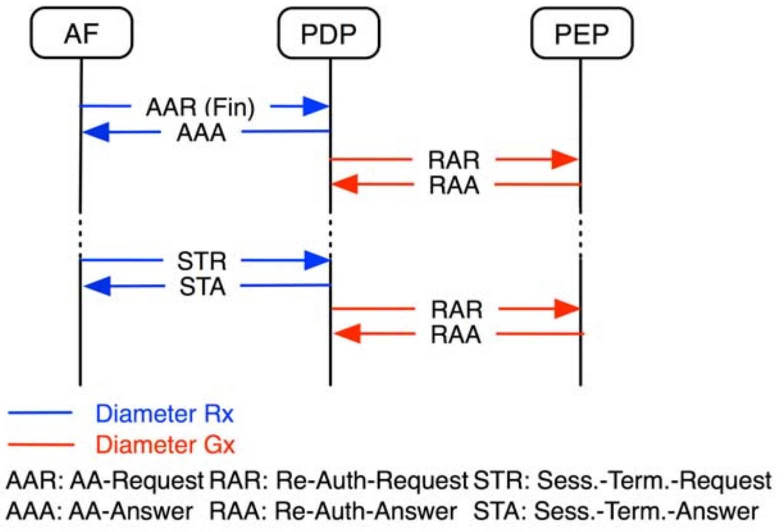
AF
PDP
PEP
AAR (Fin)
AAA
RAR
RAA
STR
STA
RAR
RAA
Diameter Rx
Diameter Gx
AAR: AA-Request RAR: Re-Auth-Request STR: Sess.-Term.-Request
AAA: AA-Answer RAA: Re-Auth-Answer STA: Sess.-Term.-Answer

Fig. 5. Message Flow between AF, PDP and PEP

3.2 PDP – Policy Decision Point

As already mentioned previously, the PDP is the specific component that is responsible for translating the media-specific data of a service request (like codec and media type) to QoS-specific parameters (like bandwidth and delay requirements). In order to realize this task, the PDP uses rules from an external rules repository, which allows deriving the appropriate QoS parameters from service specific and user specific data.

Our implementations of the Rx and Gx interfaces are based on the jDiameterPeer due to the Fraunhofer FOKUS group. On top of it we have built a basic logic, which is able to handle incoming messages, to keep pointers to the corresponding state machines and to forward messages to the Rx and Gx interfaces. We have implemented the resulting state machine (depicted in Fig. 6) as well as "dummy" interfaces to a Subscription Policy Repository (SPR), which handles user and domain policies. Additionally, our implementation provides references to the PEP instances which have to be used for each subscriber.

One of the main tasks of the PDP consists of mapping all requests to sessions, thus enabling the storage of a consistent state for each session. In this context, messages received by the Policy Decision Point (PDP) can be subdivided into preliminary service information messages and final service information messages. Whereas final service information messages install rules at the PEP, preliminary information messages only check if a requested service meets the corresponding policies and if there are enough resources available.

The resulting state machine includes therefore the following set of states:

- *Receiving*: waiting for incoming AAR message
- *Accepted PRE*: received a AAR with Service-Type AVP set to PRELIMINARY_SERVICE_INFORMATION
- *Rejected*: Service information received with not acceptable content (either due to policy or insufficient resources)
- *Accepted Final*: received AAR with Service-Type AVP set to FINAL_SERVICE_INFORMATION
- *Committed*: enforcing the rules at the PEP successful

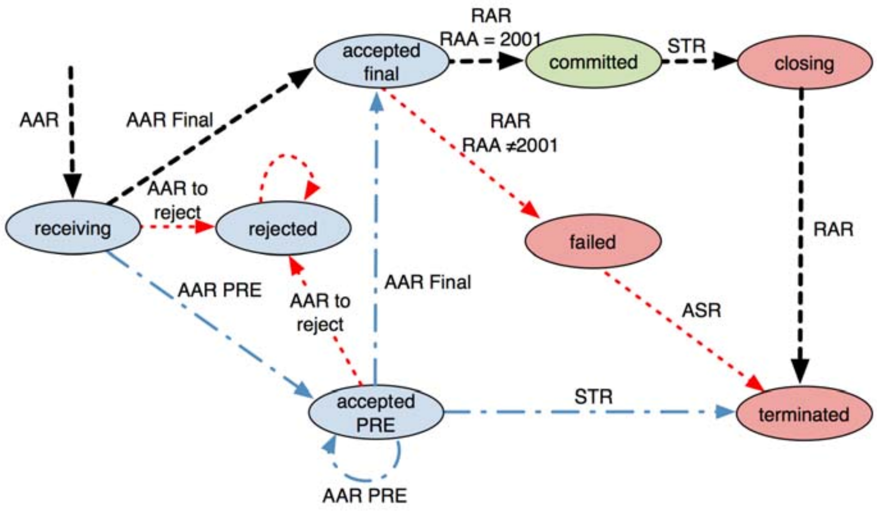

Fig. 6. PDP State Machine

- *Closing*: received STR from the AF, trying to terminate session at the PEP
- *Failed*: enforcing the rules at the PEP failed
- *Terminated*: Session successfully terminated

Note that in Fig. 6, a sequence of black (dashed) arrows depicts the traversal through the states, if the initial request already contains the final service information and if the installing of the rules at the PEP works properly. Blue arrows (dash-dotted) indicate that the initial request contains only preliminary service information, and a second request is sent to install the rules at the PEP. Red arrows (dotted) are used if either the request coming from the AF is not acceptable or the installation of rules at the PEP has failed. In order to maintain the clarity of presentation, the three states necessary for a session update are not shown in the diagram. Note that for each session handled at the PDP, a new state machine is created in order to deal with the corresponding messages.

3.3 PEP – Policy Enforcement Point

The Policy Enforcement Point (PEP) is responsible for detecting microflows and processing them according to the configured QoS parameters. Both processes (detection and processing) are defined by rules which are received via the Gx interface from the PDP.

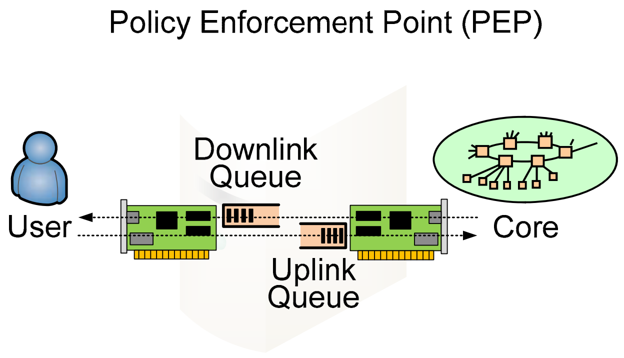

Fig. 7. Interface Cards and Queuing at the PEP

Our implementation is realized with Java, reusing the jDiameter stack already introduced. The detection and enforcement process is realized by Linux tools like tc and ipTables. The PEP stores the current state and configuration of each rule and tracks the bandwidth consumption in the PEP Rules Management module for charging. For a comprehensive illustration of the main PEP components we refer to Figure 7.

The PEP works an intermediate component between the communicating hosts. In order to be fully transparent for IP traffic, it is essential to configure the PEP as a bridge. With Linux, this requires two interface cards at the host, one towards the user agent and one towards the core network. The queuing is always realized at the sending interface, whereas QoS enforcement for traffic directed to the user agent can only be realized on the interface card towards the user agent (i.e. downlink). Similarly traffic directed to the core network can only be treated at the interface card towards the core network (i.e. uplink). In our implementation, we use the Linux system tool brctl to configure a Linux host with two interface cards as a bridge.

Additionally, we have decided to reuse the concepts for installing, modifying and removing of rules by means of the Diameter protocol from the 3GPP PCC architecture, where these rules are called "charging rules" and include four main aspects:

- Each rule has a unique identifier (Charging Rule Name), which is mainly necessary to be able to remove a certain rule.
- Each rule is responsible for a certain microflow characterized by a flow description using the following parameters: source IP address, destination port, destination IP address and protocol number. In the case of a bidirectional flow, two such flow descriptions are required.
- The third part of a rule contains QoS information which is described in terms of the guaranteed bit rate, the maximally requested bandwidth and the traffic class (like streaming, realtime conversational, etc.). The traffic class is used to set the DSCP value properly.
- The last part in the charging rule concerns charging information. As our implementation currently is prepared for charging but does not implement charging interfaces, any information contained in this part is ignored.

Altogether, these four parts are encoded as so-called Charging Rule Definition AVPs, see Figure 8.

```
[Charging-Rule-Install]
         [Charging-Rule-Definition]
               [Charging-Rule-Name]                Video-Rule;12345
               [Flow-Description]
                     permit out 17 from 10.0.0.1 to 10.0.0.6 10001
               [Flow-Description]
                     permit in 17 from 10.0.0.6 to 10.0.0.1 3400
               [Flow-Status]                       ENABLED (3)
               [QoS-Information]
                     [QoS-Class-Identifier]        1
                     [Max-Requested-Bandwidth-UL]  175000
                     [Max-Requested-Bandwidth-DL]  175000
                     [Guaranteed-Bitrate-UL]       150000
                     [Guaranteed-Bitrate-DL]       150000
               [Online]                            DISABLE_ONLINE (0)
               [Offline]                           ENABLE_OFFLINE (1)
               [Metering-Method]                   DURATION (0)
               [AF-Charging-Identifier]            chargingid987654321
```

Fig. 8. Example of a Charging Rule Definition

After starting the PEP, some generic rules have to be installed first (for example signaling traffic should be able to pass under all circumstances). Such predefined rules are stored in the configuration file of the PEP and are activated automatically. As an alternative option, a Charging Rule Command triggered by the PDP can also install these rules.

After receiving an installation request via the Gx interface, the PEP has to configure the bandwidth management (realized with the HTB queuing discipline, see section 2) and activate the firewall for the respective microflow. The rate and ceil parameters of the tc command are configured by the guaranteed bit rate and Max-Requested Bandwidth value of the QoS Information AVP, which has to be completed both for the uplink and downlink direction in the case of a bidirectional microflow. To realize DiffServ marking, we add a dsmark queuing discipline and configure a DSCP value that fits to the traffic class. In the next step, the firewall is informed by the ipTables command that this microflow has to be allowed to pass through. As mentioned before, it is also possible to run the PEP without gate blocking. By using the tc queuing system HTB, it is guaranteed that a certain microflow cannot exceed the reserved maximum requested bandwidth. If too much traffic should be injected into the network, the PEP will shape the traffic according to the installed rules. Additionally, the currently used bandwidth and the total amount of sent bytes are observed for each microflow. These data could be used in later releases to realize flow-based charging. Note that we do not use these data to infer a bearer loss, as individual services could generate traffic patterns with no data sent over a long period of time that would wrongly be detected as bearer loss.

4 Results

We have tested the BIQINI QoS enforcement framework for various typical scenarios including voice and triple play applications in combination with an Open Source IMS testbed. For background load generation, we have used the Jperf frontend, which relies on the functionality of the Iperf traffic generator [13]. The chosen scenarios illustrate the reduction of best effort Constant Bit Rate (CBR) Jperf UDP traffic due to a realtime session, which has been signaled using IMS and enforced using BIQINI. Figures 9, 10 and 11 depict the goodput and jitter of a Jperf best effort stream for three typical cases:

- Scenario 1: 400 kbps best effort traffic is sent over the bridge, while after 15 sec the PEP is activated and consequently throttles down the traffic to 200 kbps, see Fig. 9 top.
- Scenario 2: The 200 kbps traffic is running in the background, while after 60 sec a CBR G.711 call starts for a limited duration of 12 sec, which needs about 82 kbps (excluding IP headers), see Fig. 9 middle. Note that the maximal bandwidth has been chosen in order to illustrate the impact of the G.711 call, which needs nearly half of the bandwidth available at Jperf.
- Scenario 3: A Variable Bit Rate (VBR) HD video stream consumes varying bandwidth depending e.g. on the amount of motion within subsequent frames and thus reduces the Jperf best effort traffic accordingly. In contrast to constant bit rate (CBR) voice streams, allocation of the maximum required bandwidth for VBR video streams leads to a significant waste of resources. This demonstrates one important BIQINI feature: due to the use of tc's HTB queuing discipline, the PEP does not reserve the maximum bandwidth required for the video stream. However, it adapts dynamically and automatically to the video stream's effective bandwidth consumption up to a specified maximum limit of the prioritized traffic.

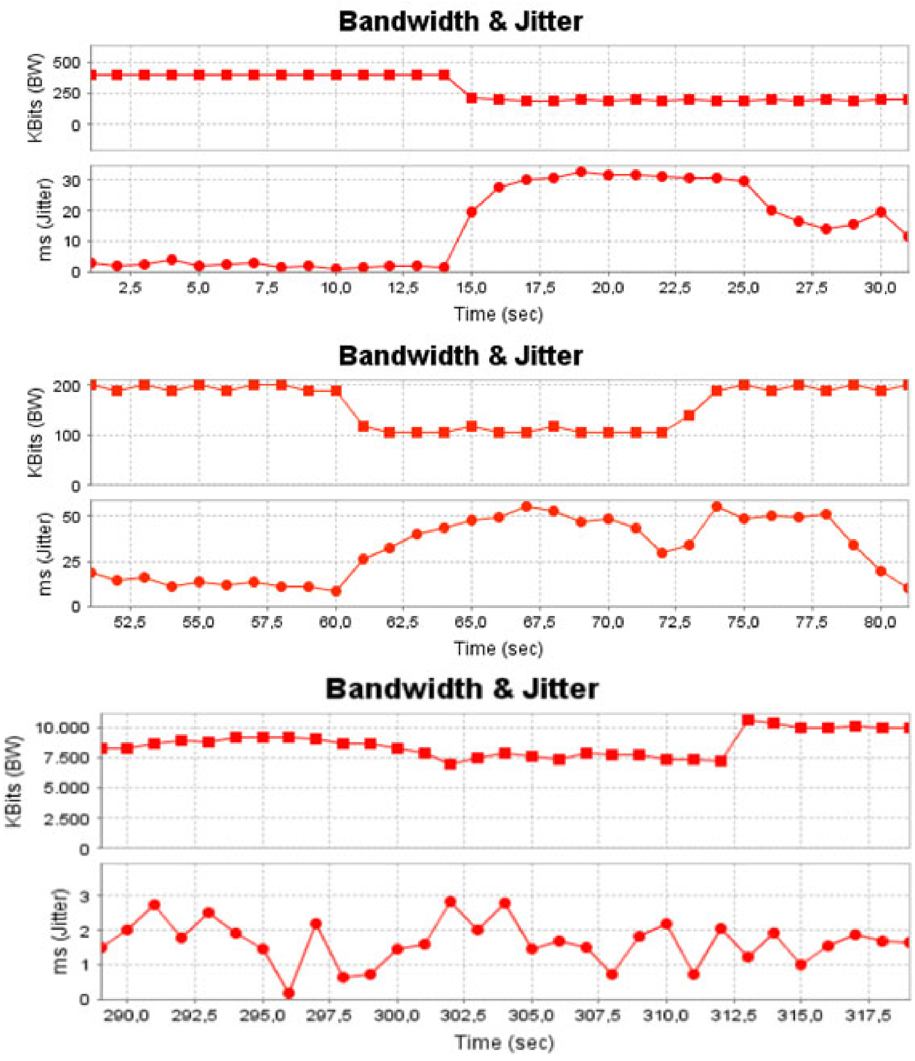

Fig. 9. Three Typical JPerf Scenarios: Reserved Bandwidth (Sc. 1, top), CBR voice call (Sc. 2, mid), VBR HD video call (Sc. 3, bottom)

5 Conclusions and Future Work

This paper presents the fundamental concepts and the architecture of the BIQINI QoS framework. Intended to serve as an add-on to the FOKUS Open Source IMS testbed, it is well suited for use also with other session-based signaling protocols like plain IETF SIP. The implementation comprises a Policy Decision Point (PDP) as well as an emulated Policy Enforcement Point (PEP) enabling transparent, realistic and automated QoS and QoE trials. BIQINI realizes the reference points Rx and Gx specified by 3GPP and provides a policy interface which can be used to experiment with various policy repositories.

One important limitation of the emulated PEP concerns the delay between activation and reaction (i.e. QoS enforcement). In our experience, this delay can amount up to one second which we, however, consider to be acceptable for a prototype implementation.

As future work we plan to publish the BIQINI source code under the GNU Public License as a relevant enhancement of the current functionality of the Open Source IMS testbed. Moreover, recent projects at the FTW Vienna with specific focus on policies and services in IMS and non-IMS environments have started to extend BIQINI's policy repository by interfacing with semantic policy engine implementations.

Acknowledgement

Part of this work has been funded in the framework of the Austrian COMET competence program supported by the Austrian government and the City of Vienna. The authors would like to thank their colleagues Hannes Weisgrab, Ronald Fischer, Joachim Zeiss (FTW), Franz Edler (Alcatel-Lucent Austria) and Wolfgang Brandstätter (Telekom Austria) for their invaluable help with the implementation and all BACCARDI colleagues for their continuous support and enthusiasm. We gratefully acknowledge the tremendous support of our industrial project partners in BACCARDI, namely Alcatel Lucent Austria, Kapsch CarrierCom, mobilkom austria, and Telekom Austria.

References

1. Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); Remote CPN QoS Control; Study on CPN - RACS Interaction, ETSI TR 182 031 (Draft version)
2. Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., Weiss, W.: An Architecture for Differentiated Services. IETF, RFC 2475 (1998)
3. Diez, A., Gouveia, F., Corici, M., Magedanz, T.: The PCC Rule in the 3GPP IMS Policy and Charging Control Architecture. In: IEEE Globecomm, New Orleans (December 2008)
4. Yavatkar, R., Pendarakis, D., Guerin, R.: A Framework for Policy-based Admission Control. IETF, RFC 2753 (2000)
5. Linux Advanced Routing and Traffic Control, `http://www.lartc.org`
6. 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Policy and charging control architecture, TS 23.203 Release 8.0.0
7. Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); Resource and Admission Control Sub-System (RACS): Functional Architecture, ETSI 282 003 V2.0.0
8. Calhoun, P., Loughney, J., Guttman, E., Zorn, G., Arrko, J.: Diameter Base Protocol. IETF, RFC 3588 (2003)

9. Waiting, D., Good, R., Spiers, R., Ventura, N.: Open Source Development Tools for IMS Research. In: Proceedings of the 4th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities (2008)
10. Good, R., Ventura, N.: An end to end QoS management framework for the 3GPP IP multimedia subsystem. In: IEEE International Conference on Communications and Telecommunications 2007, ICT-MICC 2007, Malaysia, pp. 605–610 (2007)
11. One Voice Initiative, Version 1.0.0, `http://news.vzw.com/OneVoiceProfile.pdf`
12. The GSM Association: Rich Communication Suite Initiative, `http://www.gsmworld.com/our-work/mobile_lifestyle/rcs/index.htm`
13. National Laboratory for Applied Network Research (NLANR). Iperf Performance Measurement Tool, `http://dast.nlanr.net/Projects/Iperf`
14. Fabini, J., et al.: Generic Access Network Emulation for NGN Testbeds. In: Proceedings of the 4th International Conference on Testbeds and Research Infrastructure for the Development of Networks & Communities (TRIDENTCOM 2008), Innsbruck, Austria, Paper ID 3135 (2008)