QoSJava: An End-to-End QoS Solution*

Xiaohui Huang, Yu Lin, Wendong Wang, Shiduan Cheng

State Key Lab of Networking and Switching
Beijing University of Posts and Telecommunications
Beijing, P.R.China, 100876
{hxiaohui, linyu, wdwang, chsd}@bupt.edu.cn

Abstract. Incompatibility of different QoS (Quality of Service) mechanisms and heterogeneity of different vendors' network devices are the major obstacles for providing end-to-end QoS in IP network. Inspired by Java, we propose an end-to-end QoS solution in this paper, i.e. QoSJava, which decouples QoS requirements from network details. By QoS Mechanisms Adapter and Device Driver, which act as "Java Virtual Machine", QoSJava enables interoperation between different QoS mechanisms and cooperation of dissimilar network devices. A prototype of QoSJava has been implemented, and the experimental results prove that network devices can be configured automatically to provide an end-to-end QoS. Moreover, QoSJava is not only compatible with current QoS mechanisms and devices, but open to new QoS solutions and advanced devices in the future.

1 Introduction & Motivation

Today network becomes a necessity in most people's daily life. People use network to do shopping, watch movies, make phone calls, read news, play games and so on. And naturally, they require current network infrastructures to transform from providing mere connectivity to a wider range of tangible and flexible network services with QoS. However, current traffic of various services is carried by IP network, which only provides best effort transmission. Therefore, QoS provisioning in IP network has been a hot topic in recent years.

Many researchers concentrate on this problem and have proposed a great deal of solutions. Among them, IntServ [1], DiffServ [2] and MPLS [3] are well-known. Moreover, many projects brought forward innovative solutions. CADENUS [4], TEQUILA [5] and AQUILA [6], which are part of Euro Commission's IST (Information Society Technologies) projects, have implemented architectures to provide QoS in IP network. They are independent between each other and provide solutions for IP QoS.

* This work was supported by the National Basic Research Program of China (Grant No. 2003CB314806 and 2006CB701306), the National Natural Science Foundation of China (No. 90204003 and 60472067) and the National 863 Program of China (No.2003AA121220)

However, none of the QoS solutions proposed is in use. The current network is still a best effort IP network. Though some network regions are equipped with routers with MPLS capabilities, to establish LSP for each micro-flow is impractical. End-to-end QoS is still far away from the ultimate goal.

QoS will bring profits for Service Providers (SP) without any question. But why does the situation remain the same? When investigating the large scale network, the essential reason can be found out. Current network infrastructure is divided into several domains and belongs to different Network Providers (NP), who purchase network devices with diverse capabilities in light of their budget, and adopt different QoS mechanisms based on the devices. Noticeably, dissimilar QoS mechanisms are not compatible with each other. Thus all the aforementioned projects [4, 5, 6] assume that a unique QoS mechanism is deployed in the whole network, which makes them impractical in the real environment. Though mapping mechanisms enable the interoperation of two different QoS mechanisms [7, 8], we argue that developing mapping mechanisms between all QoS mechanism pairs is impractical, especially as more and more new QoS mechanisms appear in the future. In addition, devices of different vendors have disparate command sets. When QoS mechanisms need to be changed, instead of issuing an order to do batch modification, network administrator has to log in each router and modify the configuration one by one, which increases the operational cost. In a word, a major obstacle for providing end-to-end QoS in IP network is the heterogeneity of network devices and QoS mechanisms.

QoSJava is proposed in this paper to solve the problem. Our solution is named QoSJava only because the idea comes from Java. Providing e2e QoS in current network has some similarities with programming in distributed environment. Programming in distributed environment should consider the portability of the program and the heterogeneity of the runtime environment. Analogously, providing e2e QoS in heterogeneous IP network should adapts to various QoS mechanisms and devices. As we known, Java is a powerful language for distributed network environment. After compiled, Java programs run on Java Virtual Machine (JVM) implemented on a particular platform. JVM plays a central role in making Java portable. It provides a layer of abstraction between the compiled Java program and the underlying hardware platform and operating system. Thus Java can conceal the heterogeneity of runtime environment and gain great success.

Inspired by Java, we propose QoSJava and believe it is a desirable solution for QoS provisioning in IP network. Different from other QoS solutions [1-3, 4-6], QoSJava can provide QoS for heterogeneous IP network, without assuming that the network is deployed with the same QoS mechanism. QoSJava achieve this goal by QoS Mechanism Adapter plus Device Driver, which accomplish the similar functions as JVM. They provide an abstraction layer between the application and the heterogeneous environment. Analogous to Java, QoSJava firstly translates user's QoS requirement to a stream of "bytecodes", i.e. deployment task specification. After that, QoS Mechanisms Adapter translates the deployment task specification into a script of instructions. Then the script is fed into Device Drive, which interprets each instruction in the script into a series of commands corresponding to the network devices. Finally, the commands are executed on the devices and configuration is actually completed. Thus QoSJava can migrate to arbitrary networks with different

QoS mechanisms and devices of different vendors, as a result provides an end-to-end QoS.

The rest of the paper is organized as follows: Section 2 describes the detail of QoSJava framework, especially the QoS Mechanism Adapter and Device Driver, which is the major contribution of this paper. The deployment of QoSJava is given in section 3. Then our implementation of QoSJava and the experimental results are presented in section 4. The paper is concluded in Section 5 with the future work.

2 QoSJava

The framework of QoSJava consists of two parts, the user part and the administrator part, as illustrated in Fig.1. The user part resides on the left of the dash, dealing with the whole process from user submitting QoS requirement to network devices being configured. The administrator part situates on the right side, which is for network administrator to initialize the network, monitor network performance and execute high level configuration task. QoS Mechanism Adapter and Device Driver traverse two parts, acting as the “Virtual Machine”. In this paper, we focus on the user part, since it lays the foundation of end-to-end QoS provisioning.

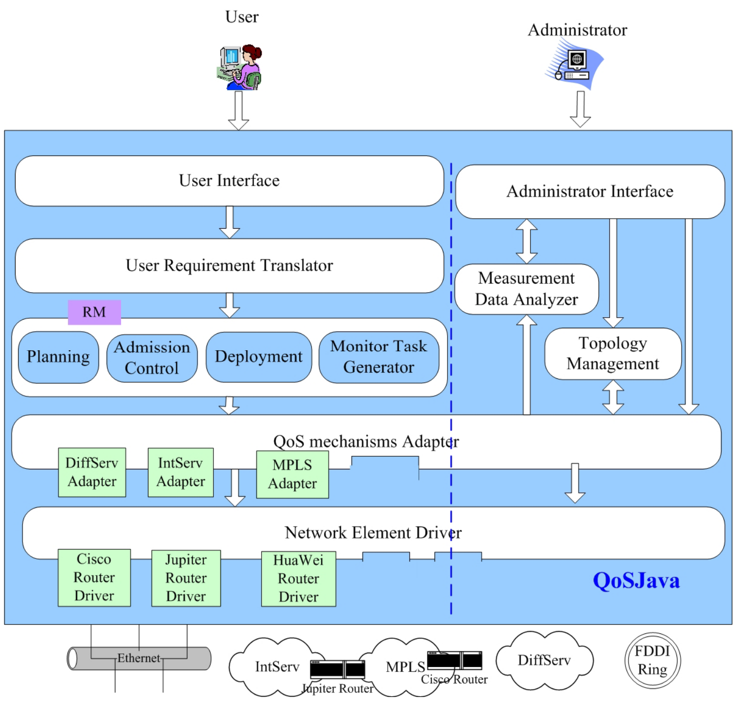


Fig. 1. QoSJava Architecture

2.1 User Interface

User Interface (UI) is the entrance for end users. As the front-end, UI is responsible for receiving user's requirement, delivering it to User Requirement Translator and returning the admission result to the user. In Java, the software function is coded in Java language. Similarly, in QoSJava, user's QoS requirement is expressed by QoS requirement description language, which can be SLA (Service Level Agreement), xml specification or any other standard format. The implementation of UI subjects to no constraints, so developer can choose any suitable technologies to realize it. SLA is adopted in our prototype to express user's requirement, and the UI is presented to end users as a web service.

2.2 User Requirement Translator

Not all end users are network experts, thus they always express their QoS requirements in a simple way. In addition, due to the different implementations of User Interface, various expression patterns exist. Therefore, a layer is needed to be inserted between User Interface and the execution logic to extract the technical parameters reflecting user's actual requirements, and denote them in a consolidated way. User Requirement Translator (URT) is such a layer in QoSJava. Based on the policies provided by Policy Server, URT analyzes user's expectation and educes the following tuple to describe a specific QoS requirement:

$$QoSReq \triangleq (SrcIP, DesIP, BW, Class, Delay, LossRate, Jitter, StartTime, EndTime)$$

$$q_{e2e} \in QoSReq$$

The parameters contained in the tuple can be extended as needed. At present, the following items are defined: Source IP Address ($SrcIP$), Destination IP Address ($DesIP$), Bandwidth required (BW), Traffic class of the service ($Class$), end-to-end delay ($Delay$), end-to-end packet loss rate ($LossRate$), end-to-end jitter ($Jitter$), the time when the contract begins to take effect ($StartTime$), and the time when the contract begins to expire ($EndTime$).

2.3 Resource Manager

Resource Manager (RM) has a logical view of its corresponding domain's physical network, including network topology, the state and the available resource of each network device. Compared to the whole network, a domain has fewer network devices, which makes the domain oriented resource management practical. RM obtains network information from a network management system developed by ourselves. After the information of network devices (mainly routers) is collected, RM does calculations for resource planning and management. RM maintains a resource database to record the resource information of the domain where it resides. According to the q_{e2e} tuple specifying user's QoS requirement, RM enforces admission control and generates corresponding monitoring tasks.

Routers are the most important components of IP network, hence router resource gives a reflection of network resource. Router resource correlating to QoS can be abstracted into the following tuple:

$$Res \triangleq (RouterID, DomainID, RT, IfNum, If_1, If_2, ..., If_{IfNum})$$

In which $\overset{IfNum}{\underset{i=1}{\forall}} If_i \in If$

$$If \triangleq (BW, Buffer, Priority, Bucket, NextHop)$$

Current tuple has the following items: Identity of Router ($RouterID$), which is one IP of the router. Identity of the domain where the router is situated ($DomainID$), Routing Table (RT), Number of the Interfaces in the router ($IfNum$), the detail of each router interface ($If_1, If_2, ..., If_{IfNum}$), Bandwidth of the interface (BW), Buffer size of the interface ($Buffer$), Scheduling priority ($Priority$), Bucket Size ($Bucket$), and the router to which the interface connects ($NextHop$).

Router information can be obtained from network management system. The items contained in tuple Res can be extended as needed, and corresponding interfaces should be added to network management system to retrieve the required information.

Based on the resource information collected, network planning is done at first. Planning is coarse-grained, which can improve resource utilization instead of reaching the optimal resource assignment. In fact, there is no solution for optimal resource utilization in Internet due to its complex traffic pattern. Planning calculates the resource matrixes for Gold, Silver and Bronze services, which are analogous to EF, AF and BE aggregates in DiffServ [2]. Resource matrixes set the stage for admission control process. The fundamental idea of planning is to locate the bottleneck of the network, and distribute its bandwidth to the aggregate flows which share the link. Please refer to [11] for the detail algorithm. The resource matrixes produced by planning are R^{Gold} , R^{Silver} , and R^{Bronze} . They are $n \times n$ matrixes, in which n is the number of edge routers in the domain. The semantic of the element in the matrix is explained below. Take $r_{i.j}^{Gold}$ in matrix R^{Gold} as an example, it represents the available resource for Gold Service between ER_i and ER_j . It is defined by the following tuple, among which $SrcER$ and $DesER$ are the IP addresses of Ingress edge router and Egress edge router separately. Other parameters have the same meaning as in tuple $QoSReq$ and If .

$$r_{i.j}^{Gold} \triangleq (SrcER, DesER, BW, Class, Buffer, Priority, Bucket)$$

Since User's QoS requirement q_{e2e} may involve multiple domains adopting different QoS mechanisms, Admission Control component (AC) firstly decomposes q_{e2e} into several QoS requirements $q_i \in QoSReq$ ($i = 1,2,...,m$) based on domains' capabilities, and sends them to the AC of domain i ($i = 1,2,...,m$) for admission. m is the total number of domains along the end-to-end path, and q_i corresponds to domain i . The decomposition algorithm is presented in our previous work [10].

After decomposition, AC translates each QoS requirement q_i into resource requirement for domain i . Function f maps QoS requirement to resource requirement.

$f : q_i \rightarrow r_i^{out}$

In which $r_i^{out} \triangleq (SrcER, DesER, BW, Class, Buffer, Priority, Bucket)$

According to resource requirement r_i^{out} and the admission policies provided by the Policy Server, AC consults the resource database for corresponding resource matrix, and determines whether the user's requirement can be admitted. If resource of all domains along the end-to-end path is sufficient, admission is successful, or a failure notification will be returned with the failed reason to guide the user's renegotiation process.

If admission turns out to be successful, AC subtracts the resource assigned from the available resource database. A monitoring task T_i is also generated by Monitor Task Generator to perform QoS surveillance during the service operation time. The parameters of T_i are not given here for the space constraint, Please refer to [9].

After user's requirement is admitted and monitoring task is generated, Deployment component creates deployment task for lower layers. The deployment task specification is the "bytecode" of QoSJava, designating how much resource should be assigned for QoS provisioning and how to execute monitoring task for QoS guarantee. The specification can be written as an xml document. It can also be written as a configuration file with APIs (Application Program Interface) provided by lower layers. In our implementation, QoS Mechanisms Adapter provides a series of APIs for Deployment component. Deployment component can use these APIs to issue orders, such as resource assignment and monitoring task enforcement.

2.4 QoS Mechanisms Adapter

Different QoS mechanisms have dissimilar resource management patterns and QoS provisioning approaches. In IntServ, resource should be reserved in all routers along the end-to-end path. DiffServ classifies traffic at the edge and specifies packets' PHB, i.e. EF, AF and BE. As for MPLS, it establishes LSP and sticks labels to packets at the network entrance. In addition, dissimilar QoS mechanisms behave differently in traffic monitoring. The purpose of QoS Mechanisms Adapter (QMA) is to conceal their heterogeneity and provides a unified interface for Resource Manager.

QoS Mechanisms Adapter should perform at least two operations. One is to interpret resource assignment task r_i^{out} , and the other is to interpret monitoring task T_i . Both r_i^{out} and T_i are designated in the Deployment Task Specification. Based on the QoS mechanism adopted in the domain, QMA translates the deployment task specification to a script containing a series of instructions provided by Device Driver. The adapting scheme is as follows:

$$QoSAdapter(r_i^{out}) = \begin{cases} Configuration\ of\ all\ Routers\ along\ the\ path & IntServ \\ Configuration\ of\ Edge\ Rounters & DiffServ \\ Establish\ LSP\ between\ Routers & MPLS \\ To\ be\ extended & other\ QoS\ mechanisms \end{cases} \qquad (1)$$

$$QoSAdapter(T_i) = \begin{cases} \text{Monitor all Routers along the Path} & \text{IntServ} \\ \text{Monitor Ingress Router and Egress Router} & \text{DiffServ} \\ \text{Monitor entrance and exit of } LSP & \text{MPLS} \\ \text{To be extended} & \text{Other QoS Mechanisms} \end{cases} \quad (2)$$

In IntServ, QMA needs to translate the Deployment Task Specification into the configuration of all routers located in the domain along the end-to-end path. In DiffServ, QMA translates the specification to the configuration of Ingress router and Egress Router, designating traffic class (EF/AF/BE), queue priority, packet dropping scheme, and etc. In MPLS, the specification is translated into label distribution, LSP establishment and monitoring.

Formula (1) and (2) only give the semantics of QMA's result. In the implementation, the result produced by QMA is an execution script with instruction sequence. An instruction encapsulates a series of commands of network devices and can perform more advanced task than a single command. An execution script example is given below. It describes a scenario in which domain D_1 adopts IntServ. Thus in D_1, resource reservation and monitoring task deployment should be done in all routers along the end-to-end path.

```
[INTSERV_QOSCONFIG]
#ResvRes<Domain D1 , IP of Router 1, r_i^out tuple>
#DeployMonTask<Domain D1 , IP of Router1, T_i tuple>
…
#ResvRes<Domain D1 , IP of Router N, r_i^out tuple>
#DeployMonTask<Domain D1 , IP of Router N, T_i tuple>
```

When a new QoS mechanism appears, new adapting module can be added to QoSJava by extending current execution scripts or adding new execution scripts. Therefore new QoS mechanisms can merge into QoSJava without violating the existing QoS mechanisms. Thanks to QMA, variety of QoS mechanisms could be coexistent in the network to provide an end-to-end QoS.

2.5 Device Driver

While QoS Mechanisms Adapter conceals the heterogeneity of QoS mechanisms, Device Driver (DD) makes the difference of network devices transparent. Due to router vendors' different strategies, their products have disparate command sets. DD in our prototype can adapt to command sets of major router vendors including Cisco, Juniper and HuaWei.

DD provides instructions for QMA, and is responsible for interpreting each instruction into commands according to the devices' types in the domain. Instructions describe advanced tasks to be performed, such as resource reservation and monitoring task deployment. Completion of such tasks involves a sequence of commands to be executed in the router. Upper layer can issue high level orders using the instructions, and DD translates the order into a series of commands correspondently. Thus DD can realize automatic configuration of network devices, and administrators don't have to manually modify routers' configuration one by one.

Our prototype provides more than 20 instructions, categorized into QoS provision, monitor task deployment, data collection, router configuration/control, and network management. The instructions for network management encapsulate SNMP commands. An instruction example which is interpreted to commands of Cisco Router (2600, 3600 and 7200 series) is given below.

```
#MODIFY_SERVICECLASS <19>
@TELNETCONN <1>
******
Enable
******
Config terminal
policy-map p-in-<19>
class <17>
police cir <10> bc <11> pir <12> be <13> conform-action set-
dscp-transmit <14> exceed-action drop
violate-action drop
@TELNETDISC
```

Note: <N> means the Nth formal parameter of the instruction.

When the script with this instruction (#MODIFY_SERVICECLASS) is executed by Device Driver, the instruction will be interpreted into a sequence of commands, completing the task of service class modification. First the API of telnet package provided by operating system is used to telnet to the router and establishes a connection (@TELNETCONN). Then password (******) is transmitted to the router. After authentication, administrator's priority would be upgraded using command "enable" and password needs to be input again. Service class is modified in succession. The command "police" sets the parameters including committed information rate (cir), confirm burst (bc), peak information rate (pir), exceed burst (be) and the dscp value attached to packets whose rates are less than cir (set-dscp-transmit). It also indicates that all packets whose rates are greater than cir will be dropped. When the task is completed, it disconnects from the router (@TELNETDISC). These commands will be executed in batch, avoiding administrator's interference.

3 Deployment of QoSJava

QoSJava is deployed in each domain of the network and communicates in a distributed manner. It can be hosted by server farm or just a computer with powerful computation capability. A deployment example is given in Fig. 2, in which QoSJava is hosted by server farm. Components of QoSJava are hosted in separate servers.

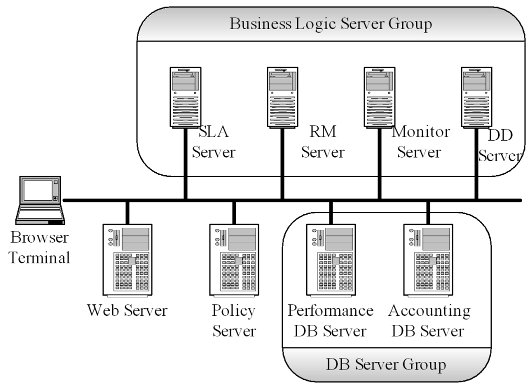


Fig. 2. QoSJava Deployment

4 Implementation & Experimental Results

A prototype of QoSJava is implemented in the National 863 project of China, whose purpose is to establish a carrier-class IP network and provide the QoS as in telecommunication network. QoSJava is the essential part of the project. Fig. 3 presents our testbed, which consists of five domains with different QoS mechanisms including DiffServ and MPLS, and consists of different network devices from Cisco, Juniper and HuaWei. We deploy more than 20 routers in the testbed. Some of them are omitted in Fig. 3 to improve visibility. A management system is also implemented as an affiliated system to monitor the performance perceived by end users [13].

In our experiment, user subscribes his SLA by a web page. Fig. 4 gives a demonstration when the user subscribes a VoIP Service. The technical parameters specified in SLA are translated into resource requirement and admitted by Admission Control component. Once the SLA is admitted, QoS mechanisms Adapter plus Device Driver configures the network devices according to the QoS mechanisms and devices series in the domain.

RouterTest instrument of Agilent and Iperf [12] are used as traffic generators. Routertest generates 256kb UDP packets at the rate of 171.24Mb/s, flooding link 172.16.4.0 to produce a congestion situation. Iperf [12] is an open source tool for network performance measurement. It injects packets in router 11.11.11.11 and congests link/interface 172.16.12.0. The link utilization of a router interface in congestion situation is illustrated in Fig. 5 in terms of CPU utilization, bandwidth utilization and packet loss rate.

Fig. 6 compares the performance of Audio service when the user subscribes to Gold Service and Bronze Service separately. Mobile nodes and correspondent nodes of VoIP service situate in WLANs (Wireless LAN) and connect to the testbed through APs (Access Point). In Fig. 6, from top to bottom, the five diagrams illustrate delay, jitter, packet loss rate, goodput, and network element load. The following statistics are obtained from the curves: packet loss rate is much less in Gold Service, approximate 2.6%, compared to 40% average loss rate in Bronze Service. The delay and jitter are very small in Gold Service, but they increase significantly in Bronze Service when congestion occurs. Some spikes appear in the curves of Gold Service because of the noise in the wireless link. The quality of voice is excellent in Gold Service. But when carried on Bronze Service aggregate, there is obvious incontinuity in the speech.

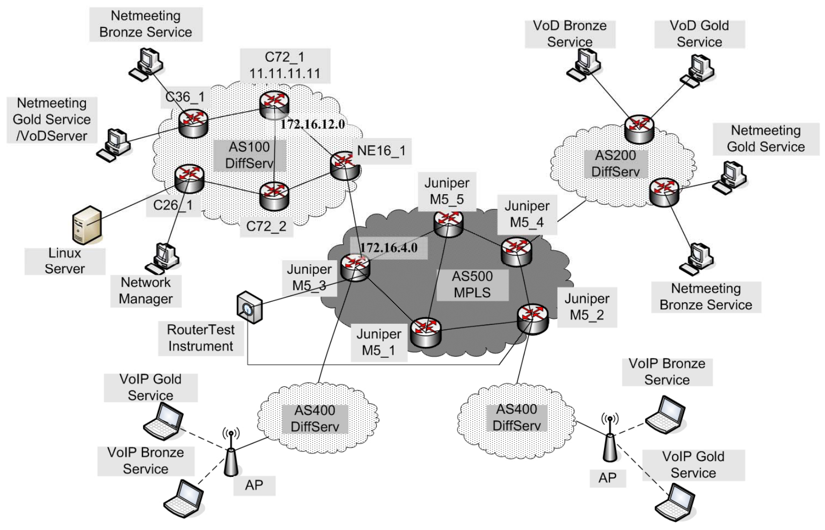


Fig. 3. Testbed

Fig. 7 depicts the performance of Video service in Silver and Bronze aggregate. Before the background traffic is generated, their performances are almost the same. But after the traffic is injected into the network, the curves show that the Video performance of Silver Service is much better than that of Bronze Service. Fig. 8 and 9 present the image of a movie, one with QoS (Silver Service) and the other without (Bronze Service). When congestion happens, the distinction of their performance is obvious. Experiments are also conducted for other services including video conference (Netmeeting), on-line games and ftp service. The results are omitted due to the space constraint.

The experiments prove that, even in the network with heterogeneous QoS mechanisms and network devices, QoSJava does deliver differentiated quality of service.

5 Conclusion & Future Work

QoSJava can conceal the heterogeneity of different QoS mechanisms and devices. Network devices from different vendors such as Cisco, Juniper and HuaWei can be managed automatically. Moreover, QoSJava is compatible with new QoS solutions and advanced devices. QoS is provided by software implementation and current network needs little modification. Therefore the network can evolve smoothly and legacy investments are preserved. QoSJava is an open and stable QoS management architecture. It is independent of the evolvement of network technology, QoS mechanism and application implementation. Consequently, it can adapt to new service requirements in the future.

When it is put into large scale use, performance and security issues should be considered carefully. Security mechanisms such as digital signature and encryption will be added to our prototype. We also think of adding an Access Server to deal with huge number of concurrent requests to improve the performance. These issues will be studied in the future work.

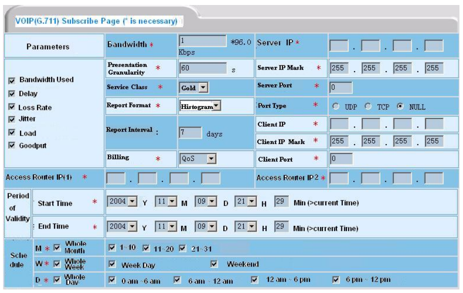
Acknowledgement

We would like to thank Xirong Que, Li Jiao, Yidong Cui, Huirong Tian, JunFeng Xiao for their intelligent idea. And we also want to thank all members of QoSA project for their hard work on QoSJava implementation.

References

1. Braden, R., Clark, D. and Shenker, S.: Integrated Services in the Internet Architecture: an Overview, Internet RFC 1633, June 1994
2. D. Grossman: New Terminology and Clarifications for Diffserv, RFC 3260, April 2002
3. E. Rosen, A. Viswanathan and R. Callon: Multiprotocol Label Switching Architecture, RFC3031, January 2001
4. CADENUS Project Consortium, Deliverable D1.2, End-user services in the Premium IP: Models, Architectures and Provisioning Scenarios, http://www.cadenus.org, November 2001
5. TEQUILA Project Consortium, Deliverable D1.1, Functional Architecture Definition and Top Level Design, http://www.ist-tequila.org, September 2000
6. AQUILA Project Consortium, Deliverable D1201, System Architecture and Specification for the first trial, http://www.ist-aquila.org, June 2000
7. Y. Bernet, P. Ford, R. Yavatkar, F. Baker, L. Zhang, M. Speer, R. Braden, B. Davie, J. Wroclawski, E. Felstaine: A Framework for Integrated Services Operation over Diffserv Networks, Internet RFC 2998, November 2000
8. F. Le Faucheur, L. Wu, B. Davie, S. Davari, P.Vaananen, R. Krishnan, P. Cheval, J. Heinanen: Multi-Protocol Label Switching Support of Differentiated Services, Internet RFC 3270, May 2002
9. Xiaohui Huang, Yu Lin, Wendong Wang, Xirong Que, Shiduan Cheng, Li Jiao, Yidong Cui: QoSjava: An Open and Scalable Architecture Decoupling QoS Requirements from QoS Techniques, draft-bupt-qosjava-arch-02.txt, http://www.ietf.org/internet-drafts/draft-bupt-qosjava-arch-02.txt
10. Xiaohui Huang, Yu Lin, Wendong Wang, Shiduan Cheng: PDB-Based SLS Decomposition in Heterogeneous IP Network, Proceedings of 2004 IEEE International Workshop on IP Operations & Management
11. Xiaohui Huang, Wendong Wang, Yu Lin, Shiduan Cheng: Resource Manager in Heterogeneous IP Network, Proceeding of International Conference on Communication and Information, 2005, to appear

12. Iperf, University of Illinois, http://dast.nlanr.net/Projects/Iperf/
13. Junfeng Xiao, Yidong Cui, Wendong Wang, Shiduan Cheng, A Service Level Specification (SLS) Monitoring System in Multiple Services IP Network, High technology Letters, ISSN 1002-0470, published by Executive Office of the Journal, Institute of Scientific and Technical Information of China, to appear.


VOIP(G.711) Subscribe Page (* is necessary)

Parameters	Bandwidth *	1 Kbps *96.0	Server IP *	. . .
☑ Bandwidth Used ☑ Delay ☑ Loss Rate ☑ Jitter ☑ Load ☑ Goodput	Presentation Granularity *	60 s	Server IP Mask *	255 . 255 . 255 . 255
	Service Class *	Gold	Server Port *	0
	Report Format *	Histogram	Port Type *	○ UDP ○ TCP ● NULL
	Report Interval :	7 days	Client IP *	. . .
			Client IP Mask *	255 . 255 . 255 . 255
	Billing *	QoS	Client Port *	0
Access Router IP(1) *	. . .		Access Router IP2 *	. . .

Period of Validity	Start Time *	2004 Y 11 M 09 D 21 H 29 Min (>current Time)
	End Time *	2004 Y 11 M 09 D 21 H 29 Min (>current Time)
Schedule	M * ☑ Whole Month	☑ 1~10 ☑ 11~20 ☑ 21~31
	W * ☑ Whole Week	☑ Week Day ☑ Weekend
	D * ☑ Whole Day	☑ 0 am ~6 am ☑ 6 am ~ 12 am ☑ 12 am ~ 6 pm ☑ 6 pm ~ 12 pm

Fig. 4. User Interface for signing contract

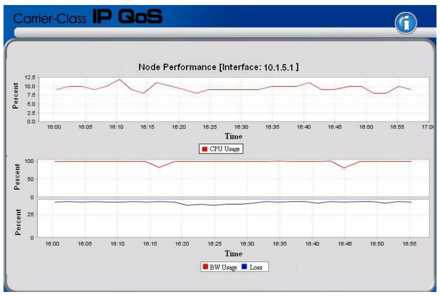


Fig. 5. Congestion Link Utilization

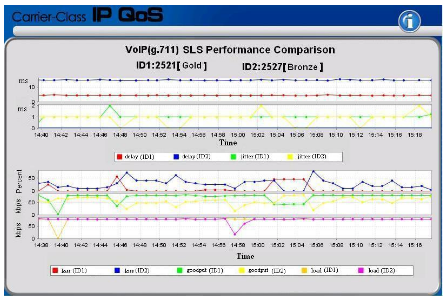


Fig. 6. Audio Performance Comparison

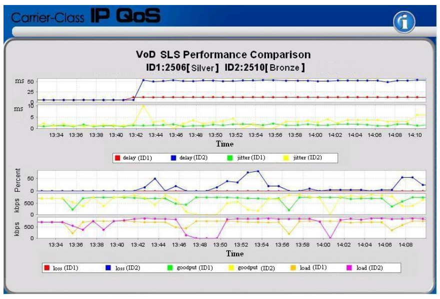


Fig. 7. Video Performance Comparison

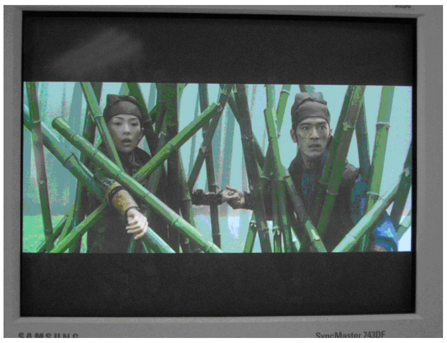


Fig. 8. Video with QoS in congestion

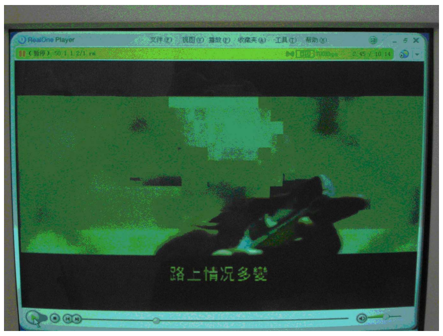


Fig. 9. Video without QoS in congestion