THE BRAIN QUALITY OF SERVICE ARCHITECTURE FOR ADAPTABLE SERVICES WITH MOBILITY SUPPORT

Georg Neureiter\(^1\), Louise Burness\(^2\), Andreas Kassler\(^3\), Piyush Khengar\(^4\), Ernő Kovacs\(^5\), Davide Mandato\(^5\), Jukka Manner\(^6\), Tomàs Robles\(^7\), Hector Velayos\(^8\)

ABSTRACT

Next generation IP networks and applications will have to address the increasingly important challenges of wireless access, mobility management, the provision of quality of service (QoS), and multimedia issues. These problems form the basis of the research within the EU financed BRAIN (Broadband Radio Access for IP based Networks) project. The project is developing a novel architecture that will be able to deal with the extreme QoS violations that are likely to occur during a running session that is exposed to the radio access environment. The core of this architecture supports different types of applications. It inherits and develops from the traditional Internet approach, but incorporates aspects of a modern flexible QoS middleware solution.

The given problem is addressed in a comprehensive, modular, and open manner, by providing different APIs to different types of applications. It provides powerful functions to application programmers, but does not assume that lower level functionality must be hidden from the application programmer. It encompasses a variety of objects, APIs, end-system mechanisms and protocols to cope with the dynamic variation in mobility management and QoS. This solution will provide applications with more predictable services and allow applications to react in a pre-determined way to QoS violations.

INTRODUCTION

This paper presents the BRAIN End Terminal Architecture (BRENTA), a quality of service architecture that is designed to provide seamless service over IP networks with mobility support and wireless access subnetworks. It is modelled using a top down approach, by defining likely usage scenarios on top and deriving the user requirements from them. Premium services will be available in hot spot areas with maximum quality. When moving out of those areas, the user expects a controlled and predictable degradation of the quality of the service received. However the user will control this degradation by specifying a set of high-level QoS parameters for each service.

These parameters are the main input for the BRAIN QoS architecture. The architecture specifies how these parameters are mapped to application and network QoS parameters, providing end-to-end QoS. To meet the user’s expectations, cooperation is needed between the application providing the services and the network elements, including the mobile terminal.

The proposed architecture does not specify how the network provides QoS to the data transport, but enables any combination of IntServ [4], DiffServ [2] or any other QoS technology like MPLS [15] to be used. Applications will be able to negotiate network resources, and will also be able adapt to the resources available. Several mechanisms already exist for resource adaptation over fixed networks, and so these are hereby integrated to facilitate the handling of the huge variations of resource availability in wireless networks.

This paper is focused on issues above the transport-layer. First, the general concepts of BRAIN and its related works are described, followed by the BRAIN End Terminal Architecture, BRENTA. BRENTA supports middleware functionality, which provides quality of service support for applications. It is an open component design based on brokers to allow distributed QoS management. Five application programming interfaces are specified to allow any kind of application to receive the desired level of QoS support from the system.
RELATED WORK

Work on the architecture of systems that provide quality of service in networks with mobility support, is still in an embryonic phase. The current work on different architectures can be divided into two groups: IETF (Internet Engineering Task Force) registered proposals (protocols for QoS and mobility support), and proprietary (typically distributed object based) proposals.

Within the Internet community, QoS is studied at both the network/transport layers and the session/application layers. The session layer protocols such as SIP [3], RTSP [6] and RTP [7] operate independently from the network and so are unaffected by mobility supporting protocols. Such protocol frameworks are currently used to provide some level of QoS for applications. However, it is likely that these protocols will find it difficult to provide the quality of service required for future interactive multimedia applications in a wireless mobile environment, where the changes to the network QoS could be huge. Thus there is a clear requirement for some form of basic network or transport layer QoS functionality. As argued in [9] however, this will not remove the need for session layer quality - rather this lower layer functionality should be minimized to prevent unwanted interactions with the application adaptivity.

At the transport layer, three different architectures exist. The Differentiated Services [2] architecture is aiming to deliver scalable service differentiation in the Internet. This is based on a simple model where traffic entering a network is classified and possibly conditioned at the boundaries of the network. The strengths of the architecture are its simplicity and excellent scalability. Its weaknesses include its lack of an associated signalling protocol for admission control and error reporting. Additionally, it assumes that large volumes of traffic are aggregated, and it is not clear how well DS would work at the edge of a resource-limited network. The Integrated Services architecture [4] and its associated signalling protocol, RSVP, [5] provide guaranteed QoS negotiation and reservation for data flows. The primary drawbacks of this approach are its lack of scalability, the need to refresh state regularly using precious wireless bandwidth, the need to re-negotiate resources as a result of terminal mobility and the associated problems with mobile IP tunnels. However, work is ongoing regarding these issues, for example, a proposal called Mobile RSVP (MRSVP) [14]) tries to solve the problems of mobility with RSVP reservations. The Internet Architecture Board has also noticed the issues related to IntServ and DiffServ and mobile environments with a new draft [14]. The third protocol in this family is MPLS [15]. It combines layer 2 switching with layer 3 routing in a new fashion, called label switching. Label switching provides improvements in the packet forwarding process by simplifying the processing and creating an environment that can support controlled QoS. The main implementations nowadays rely on Integrated Services for label signalling and therefore the drawbacks mentioned above are also associated with it.

The above protocols (once fully developed) will provide a means of supporting to some extent, real-time multimedia applications. However, many feel that this support is insufficient for many distributed multimedia systems. One problem is that these often use a distributed object model of the world [11], assume an underlying ATM network, or aim to meet QoS requirements for the terminal rather than the end user [13]. Another issue is that these applications require more explicit co-ordination of both the network and computer resources [8]. A further requirement is to provide some greater level of programming support to hide some of the complexity of QoS provision [10]. A number of architectures have been proposed to address these problems. Typical weaknesses of these approaches are a lack of independence from the networking infrastructure, and conversely, there is also a tendency not to utilise the functionality provided by the developing IP based protocols (both networking and session) - see for example [12] which also includes a survey of a number of systems. Ideally a QoS architecture should provide a smooth transition from the protocol implementations to the distributed objects through middleware, to give suitable abstractions of the underlying infrastructure. This is achieved in [11], where the authors address QoS issues with the problems of mobility, by using introspection to enable the application to gather information about the underlying communications link. However, the solution is solely for CORBA applications, and is yet to integrate the computer hardware management functionality. The approaches mentioned above (e.g. IETF protocol frameworks) are limited to closed environments. The BRAIN approach aims to provide a whole design – able to interchange protocols as required.

THE BRAIN APPLICATIONS

Due to the nature of next generation communication networks using different kind of wireless access and added mobility, applications will have to react rapidly to variations in resource availability. To cope with temporarily unavailable network resources, multimedia applications have to be elastic in adapting media representations without excessively sacrificing the perceived quality of service.

To address both mobility and QoS issues, two alternative but complementary architecture solutions have been identified. The first approach purely leverages existing protocols and components defined (or being defined) by the IETF, and tries to provide the necessary extensions to them. The choice of this organization is due to the fact the BRAIN architecture

9 The ability of an entity to analyse and understand its own internal properties and behaviour
is IP-centric and so therefore no modifications of existing applications are needed. The second approach
presumes instead the availability of some middleware, which is providing the major functionality for dealing
with mobility and QoS issues, as well as offering several Application Programming Interface (API)
functionalities for to-be-developed applications. Both viewpoints are therefore synthesized in a modular
fashion, indicated by different types of application
classes in the architecture.

There are likely to be lots of variations and developments at the lower levels, and the lower layer
protocols will only provide a certain level of QoS that needs to be enhanced for many applications. Hence the
need for a middleware layer to provide suitable abstraction from the networking layers and facilitate
session layer QoS processing.

Mobile terminals moving into regions with low signal
quality or handing-off to new access points, may
violate the QoS contract with the network, which can
cause the frequent dropping of connections. This
requires QoS adaptation and even re-negotiation. It was
always considered that the extension of the QoS
paradigm to the end user would be a hard and complex
task. All these conditions require the applications to be
adaptive in a sense that applications have to react to
varying resource availability inside the network and the
dori systems. In order to simplify the programming of
mobile broadband applications and to allow for support of
dynamic QoS changes, these active adaptation
mechanisms should be hidden to application
programmers. The idea of shifting adaptation
mechanisms from the application level to a flexible
middleware featuring QoS functions, will thereby
result in simplified application development for mobile
environments.

The goal of the BRAIN End Terminal Architecture is
to allow any kind of application to get the desired level of
support from the system in other open environments
like the Internet.

Legacy applications (Type A)

Type A applications typically run over standard
TCP/IP (+UDP) stacks and do not address QoS issues,
but can transparently achieve the benefits of QoS
guarantees over wireless links, e.g. through some
configuration tool like a DS marker. These tools can be
accessed through a set of GUIs (each addressing a
given level of user expertise). If no QoS is provisioned,
the given legacy application will still be able to operate
as usual, but without QoS support. An example of such
an application is a commercial, off-the-shelf Web-
browser, where QoS issues span from simply
improving responsiveness in the download of web
pages, to achieving high fidelity multimedia content
delivery.

Self-contained QoS-aware applications (Type B)

Type B applications directly manage QoS and mobility
related functionality, without any external support.

These applications can use various session layer
protocols (e.g. SIP, H.323), and deal with QoS issues
via IntServ and/or DS. Additionally, these applications
will typically include RTP/RTCP/RTSP functionality.
These applications are likely to be specialized
applications, written by skilled application
programmers, who know how to directly deal with
issues like e.g. QoS violations. An example of such
applications would be commercial off-the-shelf and/or
upcoming QoS-enhanced Voice over IP client
applications.

QoS-aware applications based on a component model
(Type C)

Type C applications are able to adapt to QoS violations
by themselves, but can rely on basic lower level
functionality offered by components. Therefore, the
development of this type of application can be eased
considerably. This is a category of applications not yet
commercially available. Upcoming products will be
based on existing component based platforms, e.g.
Microsoft’s DCOM, OMG’s CORBA, or Sun’s
Enterprise Java Beans.

Support of QoS-aware applications based on external
QoS handling functionality (Type D)

Type D applications typically are not designed to deal
directly with QoS violation issues, though they are
QoS-aware. Therefore these applications need some
form of intelligence, provided by external components
that hide QoS and mobility handling issues from them.
Furthermore, application programmers can even use
high level QoS-languages for the provision of QoS.
This paradigm significantly eases the programming of
distributed multimedia applications, e.g. multimedia
information kiosks that subscribe to an adaptable, QoS
aware video streaming service.

THE PROPOSED BRAIN END TERMINAL
ARCHITECTURE (BRENTA)

In order to support the aforementioned types of
applications, the proposed architecture needs to be
modular, open and configurable/flexible. Modularity
guarantees that existing applications can be
immediately used as is, whereas more complex
middleware solutions can be gracefully introduced
later, as soon as they become available. Openness
broadens the scope of the proposed architecture,
taking into account interoperability issues with other
architectural solutions (e.g. active networks).
Flexibility is needed to cope with different media types
by, e.g. supporting downloadable codecs. In addition,
the interfaces have to be well defined and standardised
so that QoS enabling components may enhance the
system by downloading them from a server during
runtime.

10 Examples of components are video grabbers, data
compressors, packetizers, etc.
Applications will interface with a QoS and mobility-aware protocol stack through a set of interfaces, each addressing one of the aforementioned application types.

**Interface Type 0**

Legacy applications (like web browsers) access IP services by directly interacting with the classical (neither QoS- nor mobility-aware) transport layer (Application Type A).

**Interface Type A**

Legacy applications (Application Type A) may eventually use the services provided by the QoS-Enabled Transport Interface\(^\text{11}\), also called the BRAIN Service Interface (SI). Since most legacy applications do not feature any QoS support, an optional external control panel would allow users to configure/setup and monitor the QoS parameters that this tool\(^\text{12}\) would provide.

---

\(^{11}\) This interface, described as a Service Interface for openness reasons, may be e.g. socket-based with QoS support.

\(^{12}\) This tool is an add-on and does not interact with the application.
QoS Management Plane
In order to provide applications with QoS and mobility support (according to the various types of applications described above) a set of entities have been identified below.

Component
A pre-fabricated, customisable software entity providing meaningful services through a published interface\textsuperscript{13}. Each component monitors its most important QoS parameters and implements means for adaptation.

Chain Coordinator (ChC)
The ChC manages one or many component chains (each associated with a flow) on behalf of either the application (Interface C) or an external QoS Broker (Interface D), in order to guarantee flow synchronisation within the tolerances requested by the user. Furthermore, this entity concentrates on QoS events being generated by the monitors associated with each component, in order to provide the QoS Broker with refined and concise information.

Component Coordinator (CC)
Provides applications with a generic framework for managing software components. More specifically, the CC deals with component and component-chain lifecycles (retrieval, deployment, activation, management, deactivation, and disposal). The retrieval and deployment phases can even include mechanisms for selecting and downloading SW components from remote repositories. To this extent, the CC provides an abstraction of the physical platform actually used.

QoS Broker
This is the centralized intelligent entity that governs at the highest level, all the QoS and mobility mechanisms on behalf of the applications \cite{IST BRAIN} on the terminal device. This entity ensures that enough resources are available to accommodate a given applications requirements at connection establishment time, both locally and remotely\textsuperscript{14}. In particular, the QoS Broker maps QoS parameters across the various components. Afterwards, during the connection lifetime, the QoS Broker monitors the connection quality, and reacts to any degraded conditions (e.g. QoS violations), by rearranging multimedia component "chains" (through the CC) and/or performing fine QoS parameter tuning\textsuperscript{15}. The QoS Broker can be a component by itself and as such it can be managed through the CC. Humans through a proper GUI, addressing various levels of user expertise can directly access the QoS Broker.

Resource Managers (NM, CM and MM)
Perform flow control mechanisms (e.g. flow policing, shaping, coordination, etc.) \cite{IST BRAIN}. Each resource manager is typically associated with one flow and a specific type of resource: NM (Network QoS Manager), CM (CPU QoS Manager) and MM (Memory QoS Manager). The resource managers can be components, manageable through the CC.

Session Manager (SM)
This is a software component that abstracts any session-layer detail, and co-ordinates multiple peer-to-peer associations within a given session.

Resource Controllers (RC)
These entities represent the finer grained control over local resources (e.g. the CRC controls the CPU Scheduler, the MRC controls paging, and the PSC controls the Network Packet Queue Scheduler, thereby acting as a service provider for the BRAIN Service Interface). For each resource there exists exactly one RC that controls admission for it, manages its reservation, allows dynamic negotiation for the resource and performs adaptation. A black box definition of said RC boxes is hereby proposed, which specifies a set of interfaces - one for each RC box - in order to achieve hardware and software platform independence.

Network Resource Booker (NRB)
Provides QoS provision, like DS and RSVP. A DS marker can be used to mark packets belonging to a given flow with each indicating the class of core network traffic. This would improve the perceived performance of legacy applications. For example, the DS marker may mark all IP-packets sent from a standard Web-browser to indicate low latency traffic class. This would result in superior web-browsing performance transparently to the standard browser. In addition, as an RSVP daemon, the NRB can analogously be used for setting up network paths with the required QoS level. The NRB can either be accessed directly by humans, via a specific GUI, or programmatically.

CONCLUSION
Within this paper we have presented a QoS architecture that provides support for adaptable services and mobility. The QoS architecture is being developed for the IST project BRAIN, that focuses on broadband radio access for IP based networks. Nevertheless, a generic QoS architecture has been provided that may be used in any networking host by making abstractions from the underlying network. Clearly, in wireless environments, the heterogeneity of both the end-systems and the characteristics of the access network is an important issue that needs to be addressed. For that,\textsuperscript{13} Examples of components are third party downloadable Java-Beans, DCOM-objects, or CORBA-objects.
\textsuperscript{14} This implies a negotiation process with peer QoS Brokers. Fallback mechanisms (broker-enabled applications communicating with other types of applications - like type A, B, or C) shall be taken into account as well.
\textsuperscript{15} This may require renegotiations with peer-Brokers.
the concept of adaptable services has been introduced, which are provided to service consumers of the QoS architecture. The user may wish to specify preferences and the system reacts accordingly. Adaptable services are supported by components that manipulate streams at the end-system or inside the network. Whenever QoS violations occur, the application is informed and may perform the appropriate actions (eventually requesting service user assistance). Mobility management is achieved by the interworking of adaptable services and specific components of the QoS architecture. We see the main benefit in providing pre-fabricated components that handle QoS and mobility, as well as adaptation mechanisms. The QoS broker component is placed on top, which co-ordinates and orchestrates local and remote resources, by using resource management specific components. In addition the QoS broker may invoke media scaling/processing elements in the network (i.e. filters) to adapt media streams for wireless access network characteristics.

The main design point of the architecture is its modularity that enables support for all kinds of application (i.e. legacy as well as special VoIP clients). This fosters interoperability, fast time-to-market developments, and a common look-and-feel across various QoS-related man-machine interfaces. Therefore, a pure VoIP client (using SIP/RTP) that manages adaptation and QoS handling via e.g. RSVP, would be supported. The hereby-proposed architecture addresses these issues in a comprehensive, modular, and open manner, by providing different APIs to different types of application. Interoperability between applications that use pure IETF approach and applications that use BRAIN provided components to manage QoS, mobility and adaptation by themselves, could be achieved using proxy style techniques. However, these issues are for future work, as well as detailed work on the QoS management and QoS negotiation protocols.

FUTURE WORK

The QoS architecture proposed in this paper will be integrated into a general architecture for the BRAIN follow-up network (BRAIN-II). In this architecture, other services and requirements are under study. When that work is finished, BRAIN-II will provide a network for hot-spot areas that includes QoS and mobility management features. We still see several issues that are open for future research:

- integration of other requirements such as mobility and strong security issues. Combination of all the requirements must be transparent for the applications;

- dynamic inclusion of ad-hoc networks (e.g. provided by personal area networks based on Bluetooth);

- interworking of different access networks based on self-organisation principles.

In BRAIN-II, practical work will require the creation of a test bed, where applications and services will be integrated with real access networks. This will provide the framework for defining experiments where QoS solutions will be validated and tuned.

REFERENCE


ACKNOWLEDGEMENT

This work has been performed in the framework of the IST project IST-1999-10050 BRAIN, which is partly funded by the European Union. The authors would like to acknowledge the contributions of their colleagues.