

# User Service Assistant: An End-to-End Reactive QoS Architecture

Björn Landfeldt

Aruna Seneviratne

University of Technology Sydney

Christophe Diot

INRIA Sophia Antipolis

**Keywords :** Reactive QoS, Architecture, End-to-end,  
User services, Internet, Adaptation, Monitoring.

## Abstract

*The rapid deployment of interactive and multimedia applications, and the increased mobility of computers leads to the need of new technical solutions in computing systems. Today, the Internet comprises a heterogeneous set of networks, with very different characteristics, especially considering the increased usage of wireless networks. Even the end systems are architecturally very different, and these factors combined lead to unreliable and unpredictable performance of networked applications. One of the problems today is the question of how to manage resources and thus provide users with control over the behavior of applications, known as Quality Of Service (QoS) management. Much work has previously been done within this area, but all proposed schemes have one thing in common, a high level of complexity, which so far have prevented any of them to be fully implemented. In this paper we propose a new approach towards QoS management, in a framework we call USA, User Service Assistant. We believe that a QoS management framework should work independent of applications and available resources, and that it should react to user input rather than trying to predict the users perception of quality. We have focused on the feasibility of implementing USA, and thus we have been able to realize it with an experimental application. This paper firstly lays out the framework, describes the differences between USA and the schemes proposed today, and then describes the implementation of the framework. After that we describe the extensions to the implementation we propose to evaluate next, in order to fully assess the model. Finally we discuss the framework and its implications, and conclude on our work.*

## 1. Introduction

Networked, general-purpose workstations are increasingly being used to run distributed multimedia applications. These systems will be based on general-purpose operating systems and interconnected using a heterogeneous mix of communication networks. One of the features of this environment is its variability. The variability stems from the general-purpose operating systems not providing the necessary real time capabilities, and the best-effort services provided by the underlying communication networks.

There has been a large body of work, which address various aspects of operating system support for distributed multimedia systems. However, as discussed in [4], the type and the level of support provided by these operating systems at present is neither suitable nor sufficient for supporting the multimedia applications. The new generation of networks being developed for both wide areas well as local areas such as B-ISDN [8] and 100VG AnyLAN will provide mechanisms for reserving bandwidth. As the exact requirements of multimedia applications are difficult to predict, reservations have to be based on worst case assumptions, i.e. maximum requirements instead of minimum requirements, as is the case in CBR or Premium service [12]. This results in very inefficient use of the available resources. In addition, wireless networks are gaining widespread use, and there will be a large installed base of Ethernet and Fast Ethernet networks in which reservations are not possible. As a result, even if the resource requirements could be accurately predicted, it will not be possible to provide universal performance guarantees.

Therefore these environments will provide variable levels of service which will dynamically change, even during a single session. This has resulted in the development of a number of Quality of Service (QoS) Management architectures[1, 3, 7, 15, 19]. QoS management architectures attempt to balance system resource requirements and presentation quality of multimedia applications. This is done by specifying an acceptable range of presentation quality (QoS levels), and determining the resource requirements for the acceptable QoS levels (QoS mapping), and then choosing the most appropriate level for the currently available resources. Whenever the operating conditions change, if the resource requirements of the minimum acceptable QoS level are still within the specified bounds, the manager renegotiates the QoS levels.

The above QoS management systems are inherently complex, making them extremely difficult to implement. Firstly, the determination of a generic range of acceptable QoS levels is virtually impossible. Secondly, the QoS mapping is extremely complex, due to the heterogeneity of the system. For example the CPU requirements of an application will be dependent on the speed of the processor apart from other factors. Thirdly, as eluded to earlier, reservation of resources end-to-end will

require specialist support from the system components which is unlikely to be available end-to-end. Fourthly, the QoS re-negotiation is at best cumbersome. Finally, use of the QoS management service requires the design and/or modification of all existing applications. As a result, to date, there has not been a single full implementation of any QoS management system.

We believe that for a QoS management architecture to be practical it must essentially be simple and useable in general purpose environments. We believe firstly that much of the complexity of the QoS architectures to date arise as a result of them being predictive, i.e. attempting to specify QoS levels and degradation paths to be used with re-negotiation. Secondly, we believe that the impracticality of these architectures stems from them relying on predictions. The bursty nature of multimedia applications make reservation unsuitable. Thus we believe that by making the QoS architecture reactive, and making the application adaptive it will be possible to both substantially reduce the complexity and make QoS management systems useable. In this paper we present a QoS management framework based on these two principles, which we call User Service Assistant (USA), and describe an experimental implementation. The main features of our framework are :

- It does not involve the estimation or prediction of resource requirements, thus eliminating the need for QoS mapping
- It requires no re-design and/or modification of existing applications,
- It provides assistance in negotiating the required level of service up-front when the application is launched, and assists when the user requests it. This removes the inherently complex re-negotiation functions.
- It does not make any decisions, thereby eliminating the requirement of determining thresholds. Instead it provides information to enable users to make informed decisions.

USA works similar to a travel agent. It gets a user request, looks at different alternatives, presents them to the user and sets up a session according to the requirements.

The rest of the paper is organized as follows. The next section presents USA, and section 3 describes the infrastructure necessary to support USA. Then in section 4 we describe the details of the prototype implementation of USA which provides assistance in service level selection, network service reservation, monitoring, and interruption/information in case of user dissatisfaction. Section 5 describes the intended extensions to USA, and section 6 provides some concluding remarks about USA.

## 2. User QoS architecture

The architecture of USA is schematically shown in figure 1. It plays a central role in the execution of distributed applications. Despite this, it is still possible to execute an application without the involvement of USA. This, however, will lead to an uncontrolled use of local and network resources.

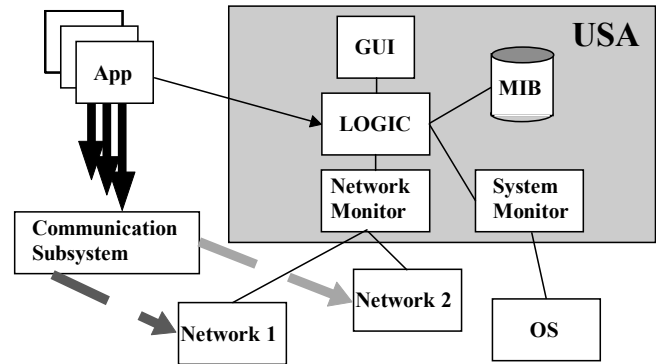


Figure 1 The architecture of USA

This architecture must be considered in the context of the Internet differentiated services [12] where the semantic of the service class is simple and very flexible. However, it could be implemented with any integrated service network as long as the semantic of the service classes can be clearly defined. This is facilitated by the following characteristics of USA:

- USA is an independent user level module which relies on monitoring capabilities. Therefore it can be implemented under any operating system, and used with any communication sub-system, network and application. Furthermore it does not require changes to the local environment. However, the efficiency of USA will depend on the support provided by the system components.
- USA only assists the user with negotiation of the transmission characteristics prior to the transfer of data. It can propose various transmission options (including different networks) out of which the user can select one depending on personal criteria such as importance of the session and price.
- Where possible, USA interfaces with all the elements involved in the execution of the distributed application. The basic operation is however based around a local MIB (made of application and network specific information) to propose to the user possible operational scenarios.
- Once the negotiation is done and the required conditions are set, USA moves to the background. It maintains a history file for each negotiation in order to be able to help the user in case of him/her being dissatisfied. This history file contains the description of the negotiated parameters, as well as the environment characteristics at the time the session started.
- It provides a simple interface, the “unsatisfied” button, which can be used by the user to express his/her dissatisfaction. When the button is pushed, USA is re-activated and uses local and network resource monitoring tools to help the user in :

- understanding what may have caused the change in the perceived level of quality,

- providing a set of possible actions that may alleviate the problem, and
- eventually re-starting a new session with a new set of parameters (no dynamic re-negotiation of the conditions of the current session).

Thus the analogy of a travel agent. When someone decides to travel, s/he goes with a budget and some travel plans to a travel agent who proposes various solutions, and who also makes reservations according to the customers requirements. The requirements are generally simple, with primary consideration of date of travel, cost and level of comfort (class), and secondary considerations of time of arrival at the destination. Criteria are not dependent of the network or of the application (e.g. If the destination is reachable by boat, train, and airplane, the notion of "class" is common to all alternatives). The travel agent negotiates with the traveler, makes the reservation, but does not provide any guarantees. Finally, if a problem occurs while traveling, the passenger complains to the service provider (airline, airport, etc.), and has to adapt.

Similarly, an independent QoS agent will help any potential user to select the right service (including the route to be followed and the resources to be used) at the time of starting the session. It is the responsibility of each user to decide when the level of service is not satisfactory, and to either stop or negotiate a new service.

This architecture is particularly adapted to group communication, where each participant will have different requirements (constrained by its capabilities) because the networks, the end systems, and the users themselves are heterogeneous.

Thus the philosophy behind USA is opposite to that of the RSVP[3] signaling protocol, where it is up to the application to make the decision of resource reservation, and where applications have to be modified to be RSVP compliant. We believe that all resource negotiations have to be controlled by the user since it is impossible for the application to have any knowledge of the user's perception of quality.

## 2.1 Assistance during negotiation

When a user wants to participate in a video-conference session, s/he might not necessarily have any prior knowledge of what application to use (VIC or IVS), and certainly have no knowledge of the different characteristics of the various networks his/her computer has access to. This will be even more true in "nomadic computing environments" where applications will not necessarily be available locally, and where access to multiple networks will be available (various ISPs, cable modem, mobile phones, satellites, etc.).

Consequently, the notion of service starts with the selection of an application, continues with the negotiation of the transmission characteristics, and should terminate with the transmission set-up. In our architecture, once USA and the user have agreed on the transmission setting, USA makes all the processing to start the application with the agreed conditions :

- It starts the application and connects it to the communication environment, in particular it selects the network access that has been chosen by the user, and may resolve all naming and addressing problems involved in setting up an IP connection over a heterogeneous link. If a group address has to be allocated, *the responsibility of the allocation falls on USA*. It also verifies the applications and sessions compatibility.
- It makes all resource reservations if such reservations are required and possible. Resource reservation can have various aspects and we are concentrating on:
  - local CPU scheduling,
  - memory reservation,
  - network resource reservation,
  - route selection.
- When necessary and possible, it starts local and network resource monitors in order to obtain the required information to provide assistance when the user is not satisfied with the level of service s/he is receiving. The monitoring aspect will be described in more details in a following section. We decided to monitor local and network resources only when the user expresses dissatisfaction as this approach is more efficient from a resource point of view. Historical information is not relevant in highly dynamic environments whereas the momentary information is. The momentary information is also sufficient to determine the cause of the problem.

This approach overcomes the problems associated with need for network signaling to be embedded within the application as in the case of RSVP which expects that the application itself issues all necessary RSVP packets to setup the network path. This behavior is not natural nor flexible as resource reservation is totally independent from applications, and that it is the users responsibility to decide whether resource reservation is required or not. In case resource reservation is required by the applications internal characteristics, the service assistant will be informed by the MIB and USA will inform the user of available options costs etc.

Moreover, our approach has an important advantage over schemes proposed to date: it is a central element of the communication architecture that can solve some problems more efficiently than any other element in the system. For example, this is typically the case for address allocation and for signalling in RSVP.

## 2.2 User dissatisfaction

In predictive QoS architectures that have been proposed, the user (or the application) has to specify threshold values that correspond to a "dissatisfaction" level. When the dissatisfaction level is reached, the QoS system automatically starts a re-negotiation procedure. In most of the architectures we have identified, the user is not informed nor involved in the re-

negotiation. We believe this is the wrong approach to quality of service management.

The quality perceived by the user continuously changes during a session. Consequently, it is the user who has to decide when the perceived quality is such that the quality is no longer acceptable, and notify the local environment. In USA, the dissatisfaction button provides the above mentioned user control. When the button is pushed, USA is re-activated, and looks at the resource monitors (local and network monitoring) to try to determine what may have caused the user to react. In a majority of cases, there will be a combination of factors that lead to the unacceptable level of service. To overcome this, USA compares the current conditions to the conditions at the start of the session using a history file. Depending on the changes in the conditions, USA determines the possible solutions, and informs the user of the available options. Then, it is the users responsibility to change the conditions for the application (for example close other applications to free memory or CPU), or to stop the session and re-start it with a new set of parameters.

As mentioned earlier, we believe that re-negotiation is the wrong approach to improve the level of quality. The reason for this is its implementation complexity, and because it is superfluous when the application is adaptive. Because of this we did not permit re-negotiation in our prototype implementation of USA.

Another advantage of the dissatisfaction button controlled by the user is that USA does not have to incorporate a complex mechanism to detect transient problems. It is impossible to predict a users changing tolerance of transients, i.e. when to start the re-negotiation, which depends on the users current mood as well as the current context. For instance, the quality of an audio stream in a users native language can typically have a lower signal to noise ratio than an audio stream in a second language. When trying to predict the threshold for user dissatisfaction the risk is that the manager would re-negotiate on a transient problem. This will not occur when the user is in control of the dissatisfaction threshold, and holds the decision of when to take action.

### 3. Monitoring

From the above description, it is clear that USA needs to carry out three main tasks:

- Determine the state of the system,
- Negotiate the transmission characteristics, and
- Inform the user of problems and provide possible solutions.

This requires a facility to monitor the system resources the application is using, well defined signaling interface for negotiating the transmission characteristics, and the logic necessary to determine the cause and provide a solution to the problem. As the user will be decision making, and the new generation networks will provide the APIs necessary for negotiating the transmission characteristic (e.g. Winsock 3,

RSVP), the major obstacle to implementing USA is the monitoring facility

A number of tools for monitoring system resource usage, and network characteristics exist today. In the case of system resource usage, existing tools such as Top or Xsysinfo (publicly available tools), provide sufficient information, and it is likely that developments such as those reported in [1] will become available in the future.

For monitoring the network subsystem, monitoring tools such as *ping* or *traceroute* have been available for a long time [17]. They however do not provide the required information. Furthermore, the more recent network monitoring tools that attempt to alleviate the problems of ping and *traceroute*, i.e. *Pathchar* [9] and the RTP protocol which provides the control packets RTCP [13], have inherent limitations that make their usage impossible on a large scale as required by USA.

In the case of RTCP, it was designed to limit its overhead to 5% of the total application bandwidth. This approach works well for small sessions, but it is not scaleable for large groups. Moreover, when the size of the group increases, the accuracy of the monitoring information decreases, as all the participants have to share the 5%. Another drawback of RTCP is that it does not guarantee that all the end systems in a session have the same monitoring information.

*Pathchar*, in contrast, computes per link information on the route between two end systems on the Internet. However, *Pathchar* needs to run for a certain period of time before it provides useful information. In addition, it is both CPU intensive and increases the network load. This makes the use of *Pathchar* unsuitable for dynamic environments, which use adaptive applications and have the most use for USA [7].

As a result, most of the adaptive applications and most of the QoS architectures, have developed their own embedded monitoring tool. This has resulted in redundant operations, and significant CPU and bandwidth overheads. An example of a well implemented networking tool is the Netflow tool designed by CISCO to support QoS management in their IOS software [6]. A centralized monitoring approach such as Netflow would have major advantages compared to distributed monitoring, as done by RTCP and *Pathchar*. Another appealing method would be to use a scheme similar to SNMP, in order to derive network characteristics. SNMP itself is unsuitable due to the fact that it often is turned off for security reasons.

Our opinion is that monitoring is one of the key elements of differentiated services or integrated services architectures. Thus, a centralized monitoring with a common semantic of parameters, which results in a minimum of CPU and bandwidth usage is necessary for future networks, as well as QoS management frameworks. Furthermore, the monitor(s) should have well defined interfaces, which will reduce the burden when developing adaptive applications. When such a monitoring tool would be available USA would make use of it. Otherwise USA would work equally well with any monitoring tool which provides the required information.

The purpose of this article is not to propose a monitoring tool. However, in order to validate USA we chose to implement our own tool for obtaining both the system and network information. The system monitor was implemented as a simple set of functions which return current CPU and memory usage. We also implemented a network monitoring tool called NECT, Network Characteristics determination Tool.

NECT is an extension of ping with added functions for determining bandwidth, delay and jitter. The tool sends ICMP echo packets, commonly referred to as probe packets, to the remote host. The choice of ICMP packets stems from the fact that many UDP echo ports on machines in the Internet are in fact turned off, so that ICMP is the most reliable tool.

In order to obtain bandwidth currently available to the user, NECT sends a burst of small packets thus emulating a small file transfer. The time of the transfer is recorded and knowing the size, the band width can be calculated. Single empty packets are then sent in order to obtain the round trip time of the data channel. This value is used to see if the delay has changed, and thereby affected the band width calculation.

The jitter is calculated as the ratio of the most deviating value to the median value, and gives an indication of the stability of the bandwidth and delay calculations.

NECT is in its initial stage of development and much research remains to be done. Further developments of NECT are discussed in section 5.

It is still difficult for a user to determine the bottleneck of the system if the network connection is downgraded at the same time as the CPU load or the memory usage is very high. Mapping of these parameters to determine the bottleneck is an open research issue, which we are currently investigating, and which is also discussed in section 5.

## 4. Implementation

After having described and justified the framework of USA, the first part of this section describes the different components of our prototype implementation of the model. The second part of this section then describes the operation.

The USA prototype is implemented under Red-Hat Linux and the majority of the code is written in JAVA for a number of reasons. Firstly JAVA allows for rapid development and rather a robust behavior. The part written in JAVA is also very much hardware independent, since the development environment we use is Suns JDK, which is a well defined platform for all hardware types for which a Virtual Machine exist. Also, almost every new computer sold today, has a Virtual Machine included, in a web-browser or in the OS. This makes the porting and installation of the software much easier. There are currently some concerns about the efficiency of running the code, but since USA is idle to a very large extent, thus having no adverse effects. For example, the average CPU usage on a Pentium 150 is currently in the order of a few percent. Currently there are two parts of the prototype implemented in C, the Network Switch

and NECT. Both these modules use low level programming not possible to implement in JAVA for security reasons.

### 4.1 Components of the prototype implementation of USA

The USA prototype consists of different components and modules. The core of the implementation is a GUI, an Engine, a Network Manager, a Network Monitor and a MIB, as shown in figure 2. Network specific modules hook on to the Network Manager via a well defined API.

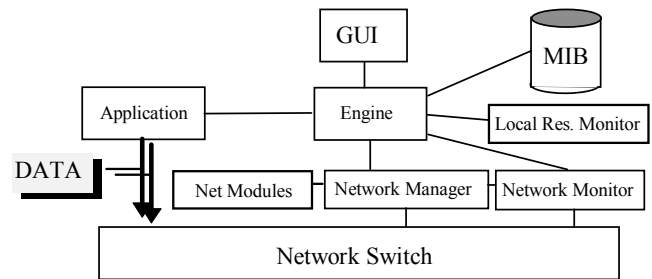


Figure 2 The prototype implementation of USA

These modules provides the Network Manager with network specific information used for performing tasks such as probing, resource reservation etc. When the Network Manager wants to perform such a task it actually calls the module, which in turn contains the necessary logic for carrying out the task. The modularity makes it easy to add specific information on how to manage new network types, to the Network Manager.

#### 4.1.1 The GUI

The user interface is implemented with one thought in mind, it is supposed to be user friendly i.e. easy to use and understand. The central part of the GUI is the service tab in the main window shown in figure 3. The tab shows which networks that are currently available and which sessions that are currently registered with the manager. The sessions become registered when the applications are started through USA, see section 4.2 Starting a session.

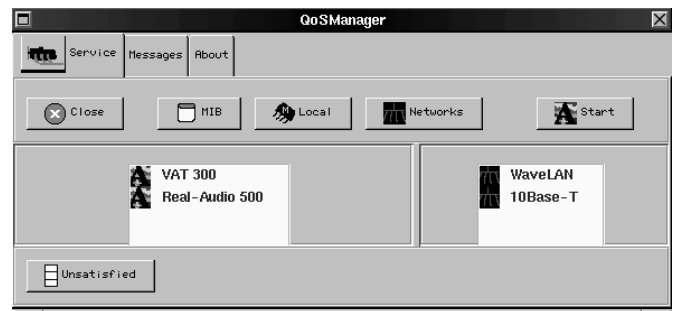


Figure 3 The application main window service tab.

From this tab, the user can select to start a new session with an application by pressing the “Start” button, to look at system information by pressing the “Local” or the “Networks” buttons, to look at the contents of the MIB by pressing the “MIB” button and to express dissatisfaction by pressing the “Unsatisfied”

button. The other different windows and tabs will be discussed later on in section 4.2 Starting a session.

#### 4.1.2 The Engine

The Engine is the central part of the USA framework. Its primary function is to map user input to the Network Manager, and to keep a record of the registered applications and their session parameters. When the user launches an application through USA, the Engine reads the application specific parameters from the MIB, maps available networks into descending order of suitability for that application, suggests a configuration to the user, and waits for user input to set up the session. The Engine is also the manager of the resource monitors, Xsysinfo for local resources and NECT for network resources. In this stage of development, the engine also reacts when the user expresses dissatisfaction through the GUI, by launching network probing, and thereafter presenting the user with information about the current status of the different networks. The next stage of development will be able to map local and network resource information into suggestions of action, and present these to the user, as described in section 5. The ability to provide the user with a suggestion of course of action when he/she is dissatisfied is a fundamental part of USA.

#### 4.1.3 The Network Manager

The Network manager has got four interfaces, to the Engine, Network Modules, NECT and to the Network Switch. It is responsible for providing the Engine with information on currently available networks and to relay information about chosen network for a session, to the Network Switch. It also has a well defined API for the network specific modules which are loaded into the manager in order to provide network specific behavior. The ATM module for instance, would provide information on how to negotiate resource reservation when starting up a session, and the different modules have got information of network specific parameters for determining the current status using NECT. When determining network characteristics, the Network Manager plays the central role of using the Network Switch for switching between the available networks, for each network starting a NECT session, and then reporting back to the Engine the results of each probe. The modularity of the Network Manager allows for easy integration of different levels of service, another key concept of USA.

#### 4.1.4 The Network Switch

The network switch is the entity within USA responsible for routing data streams in-between two hosts over a certain network link. The switch updates the routing table for a connection to a remote host, before starting the transmission, thus setting the routing of the session. The switch can be thought of as a de-multiplexer for which the control word is the remote IP-address. Furthermore the Network Switch is responsible for updating the Network Manager when there is a change in network availability.

#### 4.1.5 The MIB

In order to provide the engine with characteristics about the network resources and available applications, a MIB has been implemented. It contains the following information about networks:

- Network Name
- Maximum bandwidth
- Typical delay
- Typical Bit Error Rate
- Cost involved with using the network and
- Reservation capability

The MIB contains the following information about available applications:

- Application name
- Application parameters
- Application characteristics (adaptive non-adaptive etc.)
- Statistics from previous sessions

When an application is launched, the engine looks at the application and network specific values and orders the networks in a descending order of suitability, according to how well the parameters match. The statistics from previous sessions is used to classify available networks such as being PERFECT, SUITABLE or BAD for an application. These parameters are solely used by the engine in order to make an estimate of the required resources for an application, and from this make a suggestion of a suitable configuration to the user. Consider the case when there is a choice from a 9,6 kbps GSM link and a 2 Mbps satellite link to run a video conference over. If the user indicates that the session is a low priority session and therefore cost is the most important factor, USA should still not recommend the GSM link as the most suitable network connection for the session, even if it is less costly than the satellite link. It is instead up to the user do discard the recommendation from USA and manually select the GSM link for the session, as described in section 4.2.

Note that these parameters are not in any way compulsory but rather advisory, for the engine to make a suggestion of configuration to the user. In USA there are no musts in resources, but rather advice to the user of a suitable configuration.

## 4.2 Starting a session

When USA is first launched, the main window shown in figure 3, which correspond to the service tab is shown, displaying information about the available networks in the right list of the service tab.

The main window has got three tabs, the service tab shown in figure 3, the message tab where messages to the user are

displayed, and a tab with information about the application. The service tab contains buttons for closing the application, browsing the MIB, starting local and network resource monitoring, starting up an application and for indicating dissatisfaction with the operation of an application. The local and network resource monitoring buttons start stand alone versions of the monitoring tools in order to provide the user with concurrent information if s/he is curious, without being unhappy with the sessions performance, and thereby avoiding suggestions on how to upgrade the performance of the session.

When the user starts an application by clicking on the Start button, the Application Start dialog in figure 4 is shown. This dialog enables the user to select which application to launch and to specify application specific parameters, i.e. extra command line options such as port number, host image resolution etc.

Since the application is unmodified and does not provide any information about the remote host, the user has to fill in this parameter so that end-to-end probing of available networks can take place. Furthermore the user can select a priority level for the application, so that mapping can be done to class of service. We have decided to use three different priority levels , low, mid and high which can be interpreted as:

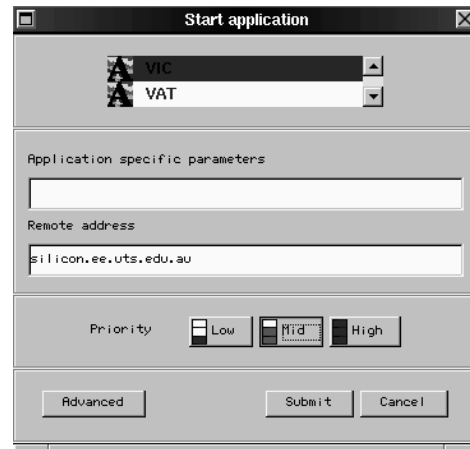
- Lowest class: Best effort, lowest possible cost and performance unimportant to the user.
- Medium class: Statistical service, the cost of the service is higher and the performance is better. This class could be realized with a priority queue in intermediate routers such as the 1 bit QoS Traffic Control implemented at INRIA [11], the assured service defined by Clark [20], or by controlled load [18] depending on availability.
- High class: Guaranteed Highest cost allowed and guaranteed service. If guaranteed service is not allowed use best possible service. This class could for instance be realized with guaranteed resource reservation [16], or yet a higher level priority queue in intermediate routers.

The two classes controlled load and guaranteed service have been defined in the IETF IntServ working group [4]. Guaranteed service assures that a packet will be delivered within a certain amount of time, and that it will not be discarded due to network congestion. Controlled load offers a nice degradation of transmission parameters according to the reservations made, but does not guarantee the service parameters in case the network becomes overloaded. It should be noted that USA is conceptually totally independent of the way the three classes of service are provided by the underlying network, and that the three classes of service could equally well be mapped to ATM classes of service, as to those specified by IntServ.

Once the application start dialog has been filled in, the user is here given two choices of how to proceed. The first choice is to click the “Submit” button in which case the application will make the mapping of network parameters to application parameters, and automatically generate a suggested configuration to the user. In this case, the “Suggestion dialog” in figure 5 is displayed, where the user is either given the choice of

accepting the proposed configuration or can choose to cancel in which case the application starting procedure will have to be restarted. The Application start dialog will however remember the latest input parameters in order to speed up the reentering of parameters for the request. The user can also click the “Advanced” button, in which case the “Advanced dialog” shown in figure 6 is displayed. Here the user is given the option of manually selecting a network for the application to use.

In either case the network selection and priority is sent to the Network Manager, which sets up the possible resource



**Figure 4 The Application Start Dialog**

reservation and tells the Network Switch to route the data stream over the chosen network. Finally the application is added to the left hand list in the service tab (see figure. 3) and USA goes into an idle state, waiting for user input. This entire procedure is the registration phase, where the session, through launching the application, becomes registered with the manager.



**Figure 5 The Suggested Configuration Dialog**

Choose Network				
Network	Bandwidth	Delay	BER	Cost
10Base-T	10000000 bit/s	1 ms	100000000	\$0.1 /min
WaveLAN	2000000 bit/s	5 ms	1000000	\$0.3 /min

**Figure 6 The Advanced Dialog**

If the user is dissatisfied with the behavior of an application, he/she can press the “Unsatisfied” button in the service tab to start collecting information about the current available networks. In this case the Network Manager will use the Network Switch to route NECT packets to the remote host over the different available networks, thereby probing all possible paths. The results of the network probes are then presented to the user, who can make a decision of how to proceed.

### 4.3 Application transparency

The implementation of USA allows for any application to be registered with it. The application is simply added to the MIB, with its specific initial parameters and USA can then launch it. The current implementation has implemented support for VIC, VAT, an MPEG Player and Real Audio/Video. The only requirement of the application is that it does not change the remote host address during a session, or else the network probing will become useless. Applications would have to be modified to tell USA in case the remote address has changed, for the network monitoring tool to work properly. Consequently, applications such as web-browsers do not work well unaltered, with the current model.

## 5. Extensions and future work

There are some necessary extensions to USA that we intend to investigate and implement. This section describes our intended future work, and the areas we intend to investigate further. As this is future work no technical solutions are available, and the results from our investigations might differ considerably from our intentions.

### 5.1 Mapping of resource monitoring information

One of the key features of USA is the monitoring of both local and network resources. The current implementation only provides the raw monitoring information from the monitoring tools to the user, who has to analyze the results, and from them draw his/her own conclusions. USA is meant to be very user friendly and assist the user in making decisions about the configuration of sessions, and therefore it is most important that it has the ability to interpret the obtained monitoring information and present possible solutions to the user. Mapping resource requirements, lack of resources and its effect on applications is therefore one of the natural extensions to USA, and will be investigated.

## 5.2 Monitoring with NECT

The first prototype of NECT is not very optimized for collecting network data as it was not our premiere goal to develop a new network monitoring tool. The statistical model has to be improved and mapping of obtained monitoring information to a more “user friendly” interface will be implemented.

In order to make NECT scalable and in order to reduce consumed bandwidth, a distributed monitoring scheme needs to be investigated. The scheme will build on the different characteristics of fixed and wireless networks, and will therefore split up the monitoring in a centralized fixed part and a decentralized wireless part. NECT is conceptually closer to mroute or ping, than Netflow or Pathchar, since the data sizes involved are relatively small.

### 5.3 Resource re-negotiation

Re-negotiation of resources have proven very difficult to implement. Mapping of new requirements is very difficult, and as a result no such framework exists today. In some cases however, re-negotiation can be implemented without cumbersome mappings. Upgrading traffic from best effort to priority traffic using priority buffer schemes for instance, involves a minimum of work and is quite feasible to implement. The simplicity stems from the priority queue model itself, where there is no real negotiation involved, but rather an upgrading of the priority of the traffic to a “higher priority best effort”, i.e. no reservations or guarantees.

Consequently, in the future USA will not allow any changes to reservations made for a session, but the user will be proposed to free resources through closing other applications or data streams, or to make use of application or protocol adaptation capabilities, if permission for doing so is granted. More radical changes will require the user to stop the current session and to start a new one.

### 5.4 Priority class dependent reaction

USA allows for three different priority classes to be set to sessions according to user request. One important issue is what these different classes will mean for USA and the user. Different priorities can be interpreted in different ways, depending on the situation. The difference in interpretation applies to both the user and to USA depending on the availability of resources. A high priority can mean to suggest an expensive network solution to the user, but in case only a best effort network is available, it might mean, close other lower priority applications competing with this one for resources. This issue will be thoroughly investigated and combined with the mapping of resource monitoring information, to form a framework for proposed action to the user. It is important for the notion of class at the user level to be universally understood, and to be independent of the underlying networks. Such a universal understanding of the class of service definition will certainly take time to investigate, and requires experimenting with USA like tools.

## 5.5 CPU Scheduling

One issue in local resource management is CPU scheduling. This is a way of making a guaranteed reservation of CPU processing time for a certain process. This enables the user to specify limits for usage of local resources for a certain application, thus controlling this possible resource bottleneck. Such a scheduler has been implemented under Linux at UTS [1], and will be integrated into our model. The scheduler filters input from both the user and the application in order to schedule the reserved CPU time sharing between processes, thus becoming adaptive. USA will be able to keep information in the MIB about performance parameters of the applications, and reserving local resources accordingly. There would be a need for a tool that securely determines the bottleneck of the system, in order not to make wrong reservations with the scheduler see sect 5.1.

## 5.6 Remote negotiation

The current implementation of USA works only within a local scope. Assistance is provided based on local information found in the MIB. It is our intention to extend a future version of USA with assistance depending on remote entities involved in the session, i.e.:

- Allocating group addresses to a group session
- Checking to see if other participants are available
- Evaluation of possible paths end-to-end instead of making a network access choice solely based on local considerations and
- Eventually making end-to-end resource reservations using signaling protocols.

These evolutionary stages are indeed the most complex of the USA model, but they are highly interesting to investigate, in order to provide the user with a better assistance.

## 6. Conclusion and Discussion

In this paper we have presented USA, a framework for QoS management. USA differs from the QoS management schemes proposed to date, by being reactive rather than predictive, and by its simple non layered structure which makes it easy to implement. USA focuses on providing assistance in decision making to the user, rather than automating the entire process of resource reservation and re-negotiation. We believe that the user has to make the final decision about the requested level of service and the cost involved simply because it is impossible to predict, especially so the users momentary preferences. USA is designed to be platform and resource independent, i.e. it is transparent to the local system and network resources.

Through the prototype we have shown that USA is possible to implement and that it is viable. This realization of USA is highly configurable, to suit different platforms. The high level components are written in JAVA and the low level components are written in C. This allows USA to be moved between different platforms which provide different levels of service, and

a different set of applications. The local resource monitor for instance, will not be available under Microsoft Windows, but this does not prevent USA from working, it rather limits its performance. USA builds heavily on resource monitoring and interpretation of the obtained monitoring results. Therefore it is of the uttermost importance that the monitoring tools used are suitable and can deliver accurate information. We have implemented a network monitoring tool called NECT, to provide us with networking information, and we intend to extend this tool considerably to make it more suitable for our purposes.

We intend to extend USA according to section 5, in order to fully evaluate the model. The extensions are necessary in order for USA to be a fully fledged implementation, and to provide the maximum possible level of service to the user.

## Related Work

Over the past few years numerous projects have addressed issues associated with providing end-to-end quality of service, and developed architectures. They all have very similar layered architectures, and logic necessary to re-negotiate and re-configure the session using some predefined degradation/upgrade paths. For example, the Lancaster QoS Architecture [7] provides a layered framework which is based on the concepts of the ISO reference model, with re-negotiation at different levels. A QoS maintenance plane contains the logic necessary to initially configure the communication subsystem and where necessary, specify the course of action to be taken in response to signals it receives from set of monitors using a set the threshold values. The University of Pierre Marie Curie architecture [3] has the same features as the Lancaster model: four layers and the logic necessary to configure the communication system to provide the level of service of requested by the application and react to signals from the monitors. In contrast to horizontal layering, the Architecture based on XRM proposed by [1], uses vertical layering (planes). However, again this model uses a predictive paradigm with two planes combining to provide the re-negotiation and re-configuration functionality. Our approach fundamentally differs from these, and the other related approaches as it is non layered and uses a reactive as opposed to a predictive paradigm.

To our knowledge, the only other QoS management architecture that is non layered is the QoS Broker proposed by the researchers at the University of Pennsylvania [24]. However, it is based on a predictive paradigm as it attempts to re-negotiate and re-configure in accordance with a prescribed policy.

## Acknowledgments

Björn Landfeldt and Aruna Seneviratne, would like to acknowledge the financial support from Ericsson Australia (EPA), and the Australian Research Council.

## References

- [1] C. Aurrecochea, A. Campbell, and L. Hauw, "A Review of Quality of Service Architectures", ACM Multimedia Systems Journal, November 1995.
- [2] G. Beaton, "Feedback-based QoS Management Scheme", HIPPARCH Workshop, Uppsala, June 1997.
- [3] L. Besse, L. Dairaine, L. Fedouai, W. Tawbi, and K. Thai, "Towards an Architecture for Distributed Multimedia Application Support", Proc. International Conference Multimedia Computing Systems, Boston, May 1994.
- [4] J.C. Bolot "Characterizing End-to-End Packet Delay and Loss in the Internet", Journal of High-Speed Networks, Vol. 2, no. 3, pp. 305-323 December 1993.
- [5] R. Braden et al. "Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification ", IETF RFC 2205)
- [6] R. Braden et al. "Integrated Services in the Internet Architecture: an Overview", RFC 1633, ISI, MIT and PARC, June 1994.
- [7] A. Campbell, G. Coulson, and D. Hutchison, "A Quality of Service Architecture", ACM Computer Communication Review, April 1994.
- [8] R. L. Carter et al. "Measuring Bottleneck Link Speed, in Packet Switched Networks", Performance Evaluation, Vol. 27&28, 1996.
- [9] <http://www.cisco.com>
- [10] C. Diot, "Adaptive Applications and QoS Guarantees", IEEE MMNet, Aizu, Japan, September 1995.
- [11] M. De Prycker, "Asynchronous Transfer Mode: Solution for Broadband ISDN", Prentice Hall, 1995.
- [12] V. Jacobson "pathchar - A Tool to Infer Characteristics of Internet Paths", <ftp://ee.lbl.gov/pathchar>, 1997
- [13] S. Keshav "Packet-Pair Flow Control", IEEE/ACM Transactions on Networking, February 1995.
- [14] M. May et al. "1-Bit Schemes for Service Discrimination in the Internet: Analysis and Evaluation", INRIA Research Report, August 1997.
- [15] K. Nahrstedt and J. M. Smith, "The QoS Broker", IEEE Multimedia, Vol. 12(1), Spring, 1995.
- [16] K. Nichols, V. Jacobson and L. Zhang, "A Two-bit Differentiated Services Architecture for the Internet", IETF Internet draft November 1997.
- [17] H. Schulzrinne et al. "RTP: A Transport Protocol for Real-Time Applications", IETF RFC 1889, January 1996
- [18] H. Schulzrinne, "Operating System Issues for Continuous Media", Multimedia Systems", Volume 2, 1996.
- [19] S. Seshan et al. "SPAND: Shared Passive Network Performance Discovery", In Proc 1st Usenix Symposium on Internet Technologies and Systems (USITS '97) Monterey, CA December 1997.
- [20] S. Shenker et al. "Specification of Guaranteed Quality of Service", draft-ietf-intserv-guaranteed-svc-06.txt, August 1996.
- [21] R. Stevens "TCP/IP Illustrated, the Protocols, Addison Wesley
- [22] J. Wroclawski, "Specification of the Controlled-Load Network Element Service", IETF Internet draft, November 1996.
- [23] A. Vogel, et al. "Distributed Multimedia and QoS: A Survey", IEEE Multimedia, Vol. 2, No. 1, Summer 1995.
- [24] David Clark and John Wroclawski "An Approach to Service Allocation in the Internet", IETF Internet draft July, 1997