Market Managed Multi-service Internet

M3I

European Fifth Framework Project IST-1999-11429

Deliverable 3
Pricing Mechanisms Pt II
Price Reaction Design

The M3I Consortium

Hewlett-Packard Ltd, Bristol UK (Coordinator)
BT Research, Ipswich GB
Eidgenössische Technische Hochschule, Zürich CH
Darmstadt University of Technology, Darmstadt DE
Telenor, Oslo N
Athens University of Economics and Business, Athens GR

© Copyright 2000, the Members of the M3I Consortium

For more information on this document or the M3I project, please contact:

Hewlett-Packard Ltd,
European Projects Office,
Filton Road,
Stoke Gifford,
BRISTOL BS34 8QZ,
UK
Phone: (+44) 117-312-8631
Fax: (+44) 117-312-9285
E-mail: sandy_johnstone@hp.com

© Copyright 2000, the Members of the M3I Consortium

Document Control

Title: Pricing Mechanisms PtII; Price Reaction Design
Type: Public Deliverable
Editor: Bob Briscoe
E-mail: Bob.Briscoe@bt.com

Origin: BT Research
Wk Package: 5.3
Doc ID: pr_react_des_1_0.fm

AMENDMENT HISTORY

Version	Date	Author	Description/Comments
V 0.0	7 May 2000	Bob Briscoe	First draft
V 0.1	9 Jul 2000	Bob Briscoe	Added AUEB Intelligent Agent discussion; Updated for consistency with M3I Architecture; Added explanation of types of reactoin table; Added comments following review at modelling workshop.
V 1.0	10 Jul 2000	Bob Briscoe	First Issue

Legal Notices
The information in this document is subject to change without notice.
The Members of the M3I Consortium make no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Members of the M3I Consortium shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

© Copyright 2000, the Members of the M3I Consortium

Table of Contents

(This page intentionally blank)

© Copyright 2000, the Members of the M3I Consortium

1 Introduction

Some applications only make sense within a very tightly bounded range of quality of service (QoS) from the network. Others are far more adaptive. For the former type of application, it is relatively easy to determine their QoS requirements. This document primarily concerns how to determine the QoS requirement of the latter type of application, given a tariff for network quality of service. We also cover how an application might describe its policy for determining QoS with respect to price. This description can then be used as policy for another entity controlling QoS, whether a middleware function on the same host, or a remote application being communicated with through a protocol.

The RSVP protocol [7] was developed to succinctly describe QoS requirements and transport them to routers. Therefore, it makes sense to re-use the content of the protocol, whether or not the transport features are required to communicate with routers on the path used by the application. The rest of the document focuses on how to determine such a QoS requirement given pricing and the new protocol elements that will be required to describe price sensitivity to QoS, rather than just QoS itself.

After giving further motivation for this work in Section 2, the body of this document is essentially in two parts. The first deals with an overview of what QoS control policies might look like, how they might be created, what type of information they contain, how they are adapted, how they are approximated, etc. The second half presents a software framework to allow QoS control policies to be applied to traffic flows in the Internet, allowing flexibility at the same time as efficiency. The document ends by enumerating limitations and further work, then drawing conclusions.

2 Motivation

A strong requirement of network service is to offer a predictable service at predictable charges [2], [3]. However, this assumes the customer (or the application she is using) can predict her own needs, as it is difficult to guarantee a predictable response to a vague request. Proposed QoS solutions favour applications and organisations that can both predict what traffic they will send or receive and predict where they intend to send it to, or receive it from. Where prediction is difficult, it is typical to simply leave a margin of error by over-estimating future demands. Current attempts to deploy the differentiated services (diffserv) architecture [4] take this pragmatic approach, often using prediction periods of weeks or months. However, in a competitive market, if over-estimation is known to be the rule, providers will tend not to deploy as much resource as customers predict they will need, instead over-booking as requirements are aggregated in order to cut costs at the risk of occasionally failing to be able to meet demand. Also, of course, customer over-estimates may occasionally be insufficient.

Although the integrated services architecture (intserv) [5] requires prediction of QoS requirements per-flow, rather than per customer contract, one can still see the same pressures at work. To illustrate the point, one only has to consider what reservation would be made by a participant in a network game or distributed simulation [8]. The game may involve long periods with not much network activity, followed by sudden flurries of data, all of which requires low latency delivery. Further, it may be impossible to predict which hosts will be interacted with until it is too late. Thus, any reservation made by participants is likely to either be far too optimistic, or far too pessimistic. Further, there will be pressure for over-booking by the provider, if the actual usage of many such reservations is being monitored.

© Copyright 2000, the Members of the M3I Consortium

Worse still, network providers have to predict the likely future level of customer predictions when sizing their physical network installations. Clearly, there is margin for error here too.

An approach that the M3I project [6] is interested in investigating is to supplement prediction of QoS with flexibility in the price domain. Thus, when predictions turn out to have been wrong, the price can be raised to prioritise available capacity for those most willing to pay. Once this approach is available to supplement prediction, the M3I project is also interested to investigate how far dynamic pricing can replace the need for prediction. The project plans to investigate whether it is possible to simultaneously satisfy a customer's demand for price stability by interposing some form of risk broking function or role. It is also interesting to speculate what new applications may be encouraged by a network that offers service to customers without the need for prediction.

Given regular variation in price might be used to signal congestion, it may become impractical to separate the price reaction function from the QoS control function. This would require continual messages between the two in order to continually adapt the QoS requirement as the price varied. This motivates the need for an application to be able to describe how it would adapt QoS as the price varies. This price reaction policy can then be used within the QoS control function.

It may be possible to describe this QoS control policy to routers on the path but it is not clear how a customer can trust its network provider to execute her own buying policy. It seems more feasible for QoS control to be moved closer to the customer, perhaps on the customer host, or in a proxy of the customer. These are respectively the dynamic price handler (DPH) and the guaranteed service provider (GSP) scenarios referred to in the M3I requirements specification [2]). These approaches have the disadvantage that price feedback from within the network is delayed, compared to if it could be acted on at the point of congestion (the router). However, among their advantages are greater scope for innovation without requiring standardisation and a removal of dependence on the route through the network (if routers aren't managing QoS control there is no longer a need to identify which routers are within the scope of the communication). This frees the network to re-route if advantageous. The pros and cons of all three approaches will be weighed in the M3I project.

3 QoS control policies and price reaction policies

We propose that price-based QoS control should be separated into two parts: **price reaction** and **QoS control**. The aim of the price reaction function is to produce a policy for the QoS controller. The Price reaction function is a high level, flexible module that must be able to adapt to novel provider tariffs and out of the ordinary user requirements. The QoS controller is separated out from this, as it must directly control the flow of network traffic and therefore must sit low in the communications stack, preferably in the kernel (or equivalent) of the operating system. Thus, the QoS controller will most likely be fairly standardised so that there are likely to only be a few types of QoS control policy. For instance, there may be one type like a token bucket which tends to keep the rate around a mean. Another type might control an adaptive transmission window, like TCP's, but perhaps allowing different multipliers in the rates of increase and decrease of the window. A QoS control policy is simply the set of parameters that controls one of these algorithms.

We will discuss price reaction policies later in this section. But first, we must work back from the goal, i.e. the end result of the QoS controller's behaviour that the price reaction function ultimately controls. The output from a QoS controller is either a specification of quality of

© Copyright 2000, the Members of the M3I Consortium

service, or actual control of the rate of a flow of traffic. Which type of control is exerted depends solely on the type of input to the QoS controller (polymorphism). This is necessary because QoS signalling on the Internet is carried in packets that must be distinguished from regular data packets.

Figure 1b)i) shows the behaviour o f a QoS controller if the input is a QoS specification, I_{q0} - the output is another QoS specification, I_{q1}, that has been set by the QoS control policy, P_q. Figure 1b)ii) shows the behaviour if the input is a stream of packets, I_p - the output is a modified stream of packets, that conforms to the QoS policy of the controller, P_q. Figure 1a) shows the abstract case of invocations of network service, I, being processed by the QoS controller, irrespective of their type, which gives the high level view of both behaviours

.

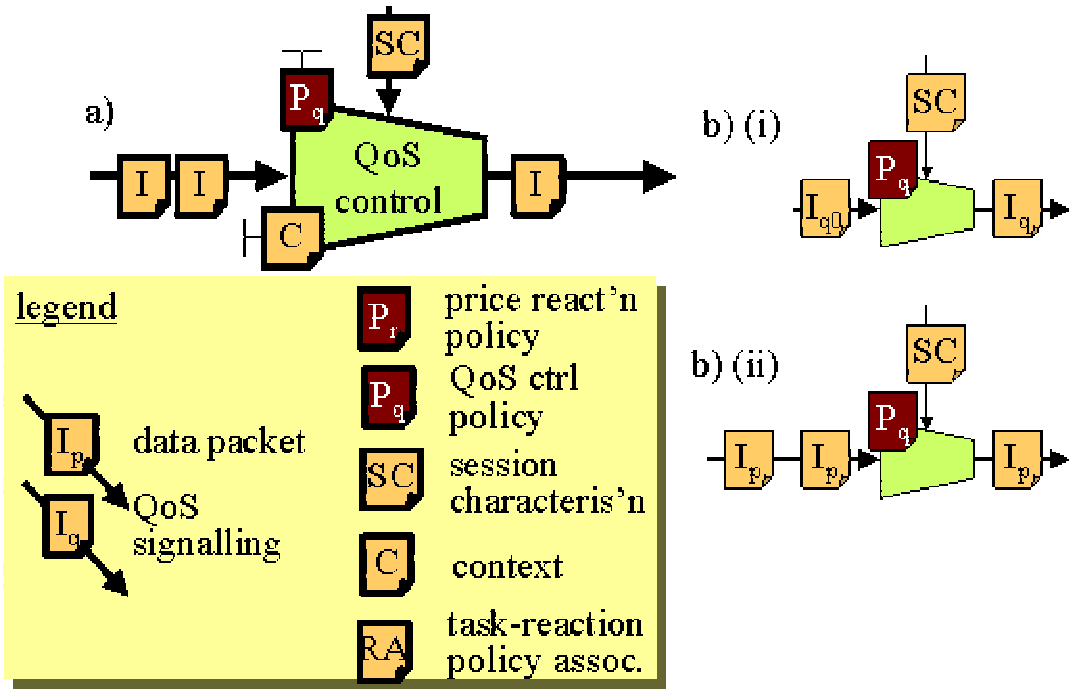

Figure 1: QoS controller polymorphism

Having identified what a QoS control policy controls, we must now clarify that, in general, a QoS control policy is multi-dimensional, as also discussed in Shenker's seminal work [1].

3.1 QoS mapping

A QoS control policy has to embody a mapping between user or application conceptions of QoS and network conceptions (e.g. RSVP takes a network view of QoS, being, effectively, the parameters required to set up a token bucket, which is the typical router mechanism used to police QoS). We believe that useful intermediary concepts from an application programmer's point of view will be:

- bandwidth (x)
- burstiness (dx/dt)
- responsiveness ($r = 1/latency$)
- jitter (dr/dt)
- delivery probability ($d = 1 - drop\_probability$)

Note that we have transformed two of the classic measures of QoS (latency and drop-probability) to ensure all our measures increase as QoS gets 'better'.

© Copyright 2000, the Members of the M3I Consortium

The notion of effective bandwidth [9] is used to embody both bandwidth and burstiness, as it provides a measure of both peak and mean bandwidth requirements. However, it still fails to encompass all the above measures. An opposite problem is that all the above measures are not independent. Introducing resource to improve one (e.g. bandwidth) improves others (e.g. delivery probability, responsiveness and usually jitter). However, there is not a linear relation between the improvements, hence the necessity to consider all these dimensions.

The nature of the mapping between user and network notions of QoS is for further investigation, primarily through a review of the likely copious literature in this area. We rule wider aspects of network QoS, such as availability, out of scope.

3.2 Approaches to price controlled QoS

Assuming we are now clear on how we expect to control QoS, we can now venture into how a QoS control policy might be generated from pricing and charging information.

In the case of dynamic pricing for network services there is a strong need for automated mechanisms that will be able to handle the varying prices on behalf of customers and ‘buy’ the desired amount of resources according to the user's utility function. The knowledge of the utility function of the customer would be adequate for such a mechanism to make the optimal choices, but it is extremely difficult to be expressed in a simple and consistent way. It depends on her elasticity, her total budget, her perception of QoS, and various emotional and unpredicted factors that are strongly related to her personality and the specific task. It is impossible for a price reaction mechanism to know, in the case of a video on demand (VoD) service for example, what is the favourite scene of a user, and if, for instance, there is congestion during that scene, whether it should offer a lot of money relative to the rest of the scenes in order to achieve the best possible QoS.

Current research activity on the design of automated price reaction mechanisms is focused on the approximation of customer’s utility function based mainly on some user input (budget, QoS sensitivity, recorded user choices) and the specific application characteristics. The general model is that the price reaction function ‘listens’ to the varying prices per unit of network resources used (p) and the QoS received (taking one dimension, bandwidth x) and decides either the new willingness to pay per unit of time (w) of the customer or the resources (QoS) that will be requested each time (x), where $w = p*x$.

Network services, as most of the services in real life, are constituted of several levels, each of which may be differently priced, but the user is always interested in the whole service delivered. In the case of the Internet, the basic service offered by the network is the delivery of packets. Above that, many services could be offered in terms of a flow, a collection of flows, etc. The different levels of a service’s provision could be mapped to the corresponding utility acquired by customers. So according to service levels, as an initial approximation we could identify two basic levels of customer utility for the network service although are undoubtedly more. The first, short-term level corresponds to the utility that the customer acquires from the volume of data received (bandwidth) per unit of time. The second, longer-term, level of utility corresponds to the overall utility that a user obtains with the termination of a flow (e.g. a video stream). Of course there could be even more levels as for example the utility of the overall QoS received by a number of flows (e.g. a video conference), etc.

Taking the utility of bandwidth first, as we stated in the introduction, the quality requirements of some applications are very insensitive to the price of the quality of the network service (QoS), while others are relatively sensitive. One can characterise the utility function with

© Copyright 2000, the Members of the M3I Consortium

respect to QoS of any user of a price insensitive application as close to a step function [1] (probably with slightly rounded off corners). On the other hand, adaptive applications and tasks can be achieved with a broad range of QoS. They therefore typically have a more gradually increasing utility function, although probably still generally sigmoid in shape with a working range of QoS over which they are most sensitive (the marginal utility is greatest). See Figure 2 for some examples. Thus, determination of a QoS specification for non-adaptive applications is straightforward, while that of an adaptive application is more involved.

A number of approaches to the problem of determining a QoS specification from the price of bandwidth have been proposed:

- An approach targeted at non-adaptive applications, based on per session sharing of a communications budget
- Approaches targeted at the adaptive application space, based around the theoretical ideal that could be attained if the customer's utility function for QoS were known:
  - A configurable enterprise agent that creates an approximation of the utility function from the configuration
  - A variation on the above, where the configuration can be created using a more declarative approach
  - A intelligent agent that learns an approximation of the utility function

These approaches therefore cover a spectrum starting from the prescriptive, moving to the declarative, and ultimately, a learning approach.

3.2.1 Non-adaptive applications

The simpler case implemented as 'QoteS' [15] is where the input consists of:

- a fixed QoS specification
- the tariff for such a QoS specification
- a budget to achieve the desired QoS
- a policy setting whether cost or quality is paramount if both requirements cannot be achieved simultaneously

In this base case, if requirements conflict but QoS is paramount, the user is asked to confirm it is OK to exceed the budget. If, on the other hand, budget is paramount, the QoS is degraded pro-rata to fit the budget. In the QoteS implementation, QoS varies slowly in response to changing cost and excess or deficit budget is traded off during the adaptation using an additive increase, multiplicative decrease algorithm internally. Cost might change due to changing price for reserved capacity, or due to variation in traffic volume under the reservation if volume is a chargeable parameter. This will be the case if the 'abc' tariff is applied [9] (Ch. 2), so-called because $charge = aT + bV + c$, where a, b and c are constants, T is the duration of the QoS reservation and V is the volume of traffic.

3.2.2 Adaptive applications

A more sophisticated (indeed idealistic) set of policy inputs would allow required QoS to be optimised with respect to the utility function of the user. This would require the following inputs:

- a set (vector) of utility functions, each with respect to a dimension, i, of QoS, $U([Q_i])$
- the equivalent vector of prices for each QoS dimension, $[p_i]$

Such inputs are shown in Figure 2, leading to an output specification of QoS, $[Q_i]$. Here the current price of the QoS dimension in question is assumed linear, and the balance between utility and charge is optimised at the point where the marginal utility matches the price [10].

© Copyright 2000, the Members of the M3I Consortium

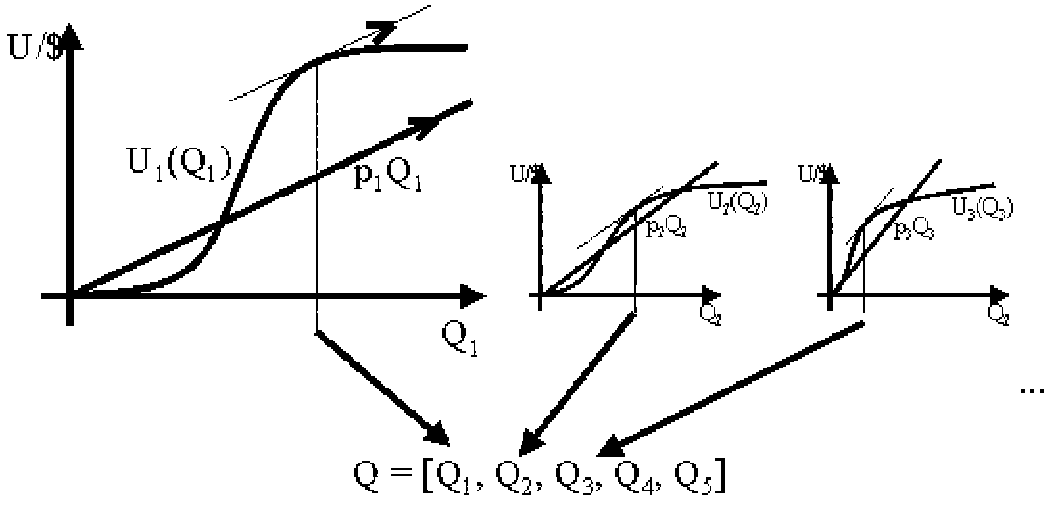

Figure 2: Inside QoS control policies

One step higher still would be to take the declarative, task-oriented approach, simply describing the task at hand (context) and enable this to lead to a QoS control policy (perhaps by looking it up in some directory of previously stored QoS control policies). Having retrieved one (or failing that been presented with an interface to define or refine one), this could then form the input to one of the previous two cases.

However, such approaches are (probably unattainable) ideals. In practice, modelling an inherently human characteristic like price sensitivity to QoS is a very inexact science. It would be overkill to build software to support such an exact model of such an inexact reality. Therefore one of the main goals of the work is to define pragmatic approximations and show that they are reasonable. We believe we have the following scope for approximations:

- prioritisation of multiple quality dimensions
- definition of the context of the task at hand
- characterisation of each utility function
- re-use of similar utility functions

Firstly, although Figure 2 shows QoS with multiple dimensions, we conjecture that, for any one task, one dimension will typically dominate price sensitivity, while the others can effectively be considered fixed

Secondly, there is good evidence that QoS price sensitivity is task rather than application specific [3], but there are clearly a large, if not infinite number of possible types of task. Therefore, in practice, whenever a QoS control policy isn't available or known for a certain task, it would seem reasonable to find a closest match as long as the user can intervene to tailor a new policy to the task in hand if the one chosen isn't appropriate. BT's first attempts to implement such a directory are shown at the top left of Figure 3, where hierarchies of services, users, applications and tasks allow a task to inherit its policy from the closest parent in the hierarchy, as highlighted bottom left of Figure 3.

Thirdly, we need a technique to approximate to the utility curve for any particular task. In this document we present one conjectured extension of an existing approach and another existing approach that will be used and evaluated in this project.

- 'QoteS' [17], shown in Figure 3, is conjectured to be able to estimate a set of utility fitness curves categorized according to the application selected and user configured bounds (budget and QoS).

© Copyright 2000, the Members of the M3I Consortium

- The 'Intelligent Agent' [18], estimates the short-term utility mentioned above based on observations of user choices.

QoteS

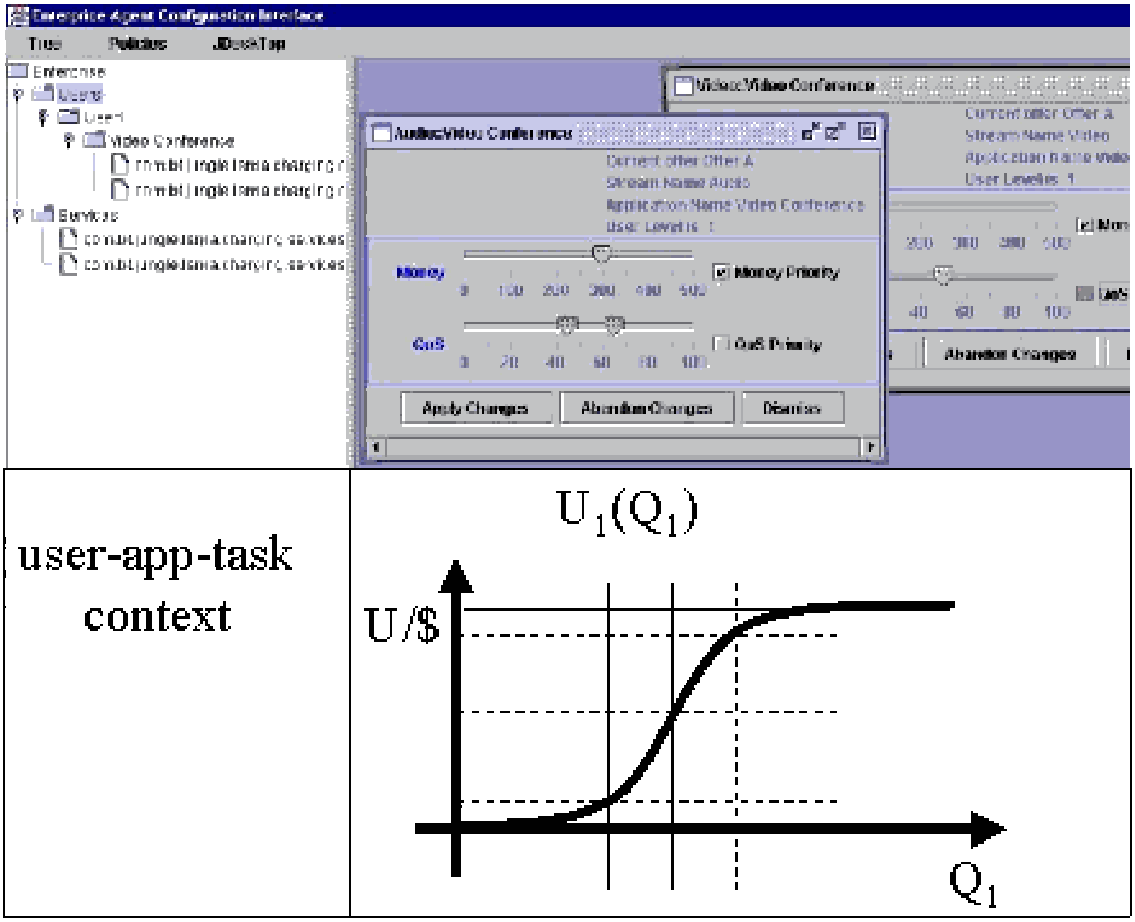

Figure 3: Setting QoS control policy

The bottom right hand side of Figure 3 shows how we might allow a customer to characterise her utility function underneath the more familiar user interface elements shown at the top right of Figure 3. This is a graphically edited adaptation of the real QoteS user interface built for the earlier, non-adaptive case. The top slider in Figure 3 allows the customer to set the maximum she is willing to pay for any QoS for a particular task context. The bottom sliders allow her to set both the minimum QoS acceptable for the task and the target QoS she would expect to be happy with. The graph bottom right of Figure 3 shows these settings as three solid straight lines. The imputed utility curve is initially assumed to be a symmetric sigmoid curve about the target QoS, perhaps modelled on a cumulative normal distribution curve. The shape of the curve is then tied down by three points and the two asymptotes by making assumptions on the number of 'standard deviations' away from the 'mean' that the utility crosses the minimum QoS bound. The lower point is the intersection of the minimum QoS and some proportion of the maximum utility (say the 5 percentile). The upper point is the intersection between the 95 percentile (say) and an assumed upper bound of high QoS sensitivity equidistant from the target QoS to the lower bound, but above it. It is further assumed that, if these assumptions prove incorrect, the customer will be able to conceptualise how to adjust the sliders incrementally to achieve the desired effect.

© Copyright 2000, the Members of the M3I Consortium

Intelligent Agent

Figure 4 shows the demand curve and the algorithm that the Intelligent agent uses to construct it, in order to estimate the utility function.

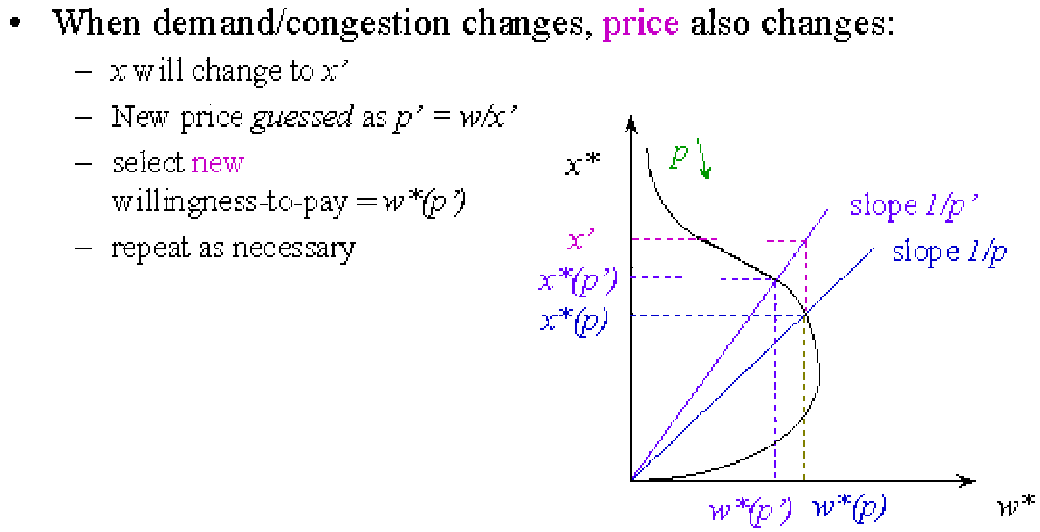

Figure 4: Intelligent agent algorithm

The Intelligent Agent records the points (w, x), which correspond to the user's choices under different network states (different prices per unit of bandwidth, where price $p = w/x$). After having collected sufficient number of points, it fits a decreasing curve in the p,x axis applying antitone regression (Figure 5) assuming concavity of the utility function $u(x)$.

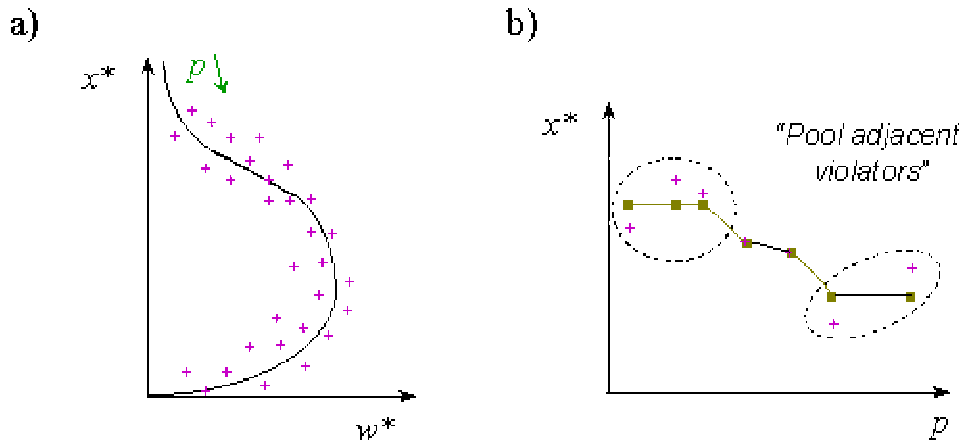

Figure 5: Filtering out 'noise' from measurements

In [18] it is shown how non-concave utility functions could be approximated by concave ones extending the applicability of the Intelligent Agent to non-concave utility functions as well.

Intelligent Agent was designed and implemented in an ATM context (for priced ABR connections) but it could easily be applied to elastic services in general, such as TCP/IP.

It replaces the user in choosing the willingness-to-pay, adaptively to the network state (which is expressed by the corresponding price per unit).

Each user is assumed to select his willingness-to-pay so as to maximize his net benefit, namely the difference of the utility acquired by the QoS received minus the willingness-to-pay. The user observes QoS (e.g. video quality), 'evaluates' utility, and adjusts w until she is

© Copyright 2000, the Members of the M3I Consortium

satisfied. Intelligent Agent records the corresponding points (measuring the bandwidth x received) until a sufficient number of points are recorded.

Simulations have shown that the Intelligent Agent works very close to the optimal choices but the fact that it only takes into account the short-scale aspects of user's utility poses some limitations to its use in a 'real' environment where users value the long-term aspects of utility much more highly than the short-term ones.

We now return to the fourth and final item in our list of possible approximations.

Fourthly, we conjecture that as budget varies an individual's utility curve will merely vary in scale, not shape for a certain task. Further we conjecture that the remaining budget available for communications will scale the utility curve in a predictable fashion. Indeed, it is likely that the utilities of different people for the same task will be similar, but scaled by individual wealth. If true this would allow default characterisations of utility curves to be deployed with applications, which could then simply be scaled to the declared communications budget of the user.

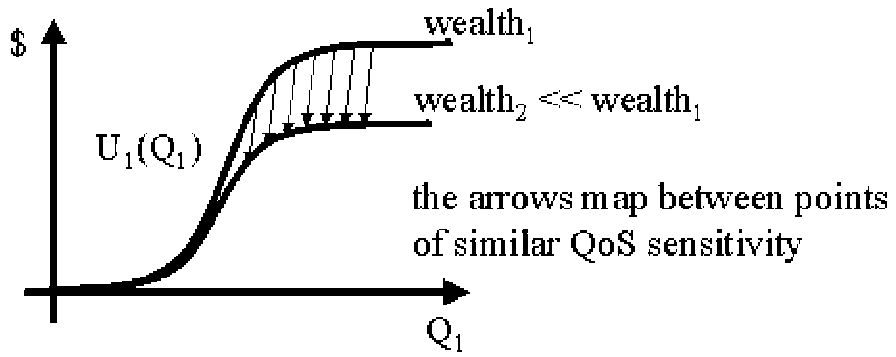

Figure 6: Conjectured effect of wealth on utility

Further we conjecture that relative wealth of an individual will primarily scale their utility vertically (Figure 6). In effect, increasing wealth or budget, simply scales the value the individual puts on money. However, there is likely to be a small effect in the QoS axis, where a richer individual can afford to be more discerning in their quality choices. The figure shows how we assume the shapes of two utility curves scale. They only differ in terms of the individual's wealth. The curve is conjectured to move mostly downwards but also slightly to the left as wealth decreases. The arrows join points of equal slope on the curves, representing equal price sensitivity to QoS (marginal utility wrt. quality). The point is that, once a utility curves for a task has its shape defined, we expect it to be relatively easy to extend this to a family of utility curves for related tasks and individuals in systematic ways.

In this way, we expect that it will also be possible to systematically infer a utility curve in one context, from that in another. For instance, if a volume discount is promised to a group on condition that a target spend is reached, it should be possible to estimate the utility curve for the context where the discount is reached and for that where it isn't. Then all that is necessary in terms of group communications is to occasionally be advised of the probability that the group will reach the target, rather than all sharing detailed usage statistics.

3.2.3 Further work

Combine the two approaches

The fact that the Intelligent Agent can adapt its learned utility curve according to new-recorded pairs, gives us the intuition to combine the two approaches described for

© Copyright 2000, the Members of the M3I Consortium

estimating the utility function. The new agent could make its first decisions according to the pre-computed curve fitness for the specific application with the budget and QoS constraints set by the user for the specific task (QoteS) and user could help the adaptation of pre-computed utility curves towards his own, ‘correcting’ the agent choices instead of starting from scratch. The fact that the Intelligent Agent must first record a significant number of user choices makes it somehow difficult to be used by inexperienced users.

Application specific intelligence

Another interesting feature that one could add to the price reaction algorithm is the knowledge of some specific application characteristics that may ‘help’ the agent in order to make the best choices concerning the long-term aspects of utility. For example, for a VoD application, the price reaction algorithm could take into account the assumption that users prefer to have a single long degradation of video quality rather than many short ones. Having such information, a price reaction algorithm could - in case of congestion and the need to allow quality degradation (because for example the price was too high) - keep for some time the low quality situation in order to ‘save’ some money for future ‘difficult’ situations.

Effectively such a feature would be equivalent to giving the user control over the setting of the kappa time constant in the feedback equations. QoteS has a user interface element to vary the frequency with which agent calculations are updated, which could form the basis of this setting

Assumptions about the future

The most challenging research area in the field of price reaction mechanisms is the ability to predict future situations concerning the network state (price, congestion) and the application behaviour (bursts, etc.).

This knowledge will enable the agent to take into account long-term aspects of utility in a more efficient way. For example, an agent having the information that a period of high prices is always followed by a period of low ones, could avoid being conservative in the ‘peak’ periods, as far as the budget is concerned, and allow some ‘over-spending’ with the hope that later it will make up for it.

In order to make such assumptions about the future one should identify the crucial events that are involved in the decision making of the price reaction algorithm and calculate the possibility that they will occur in the future based on past observations (the longer the past experience the more the confidence in the predictions made).

© Copyright 2000, the Members of the M3I Consortium

4 Types of reaction

A QoS control policy should implicitly clarify the type of reaction required by a QoS controller. This should be inferable from whichever QoS dimension is dominant. Different types of reaction are given in Table 1.

data volume	sender	sender proxy	network	receiver proxy	receiver
preserve	delay	buffer	buffer	buffer	buffer/f/b
	delay	mark	mark	f/b mark	f/b mark
	re-xmt	drop	drop	f/b nack	f/b nack
reduce	-	drop	drop	suppress nack	suppress nack
	recode	mark	mark	f/b mark	f/b mark
	recode	drop	drop	f/b nack	f/b nack
	-	-	(f/b nack)	f/b drop layer	f/b drop layer

Table 1: Types of QoS control reaction

There are two main classes of reaction, depending on the application (first column). Either the volume of data to be transmitted must be preserved, or it can be reduced in the presence of congestion. The types of reaction available also depend on where the QoS controller is in the data path: whether sender or receiver or whether somewhere in between acting on behalf of either, or the network itself. Each of the second to last columns shows what choices of reaction are available to the different parties potentially present on a path. Each row represents a consistent set of reactions which will work together to produce the desired change in QoS:

1 Any party can absorb data into a buffer for at least a short while to sit out temporary congestion conditions. Even the receiver can do this if the host processor or disk is ‘congested’. Of course the receiver also has to feed back some indication that a reaction is necessary. Delay of this feedback should cause the sender to slow down. The best place to delay data is at the sender, which can stretch all the data that must be sent over a longer time period. A typical application of this case is a file transfer.
2 The sender can delay sending future data if it is receiving congestion marking feedback, instead of delayed acknowledgement as in the previous case.
3 If the network or sender proxy has to drop a packet, the parties downstream can feedback an indication that data is missing. The sender then has to re-transmit, as all data must be preserved in this category. Note, we use the term ‘nack’ for negative acknowledgement. Strictly, this has a specific meaning where a nack is only sent when data is missing. We use the wider meaning to include not sending an acknowledgement, or acknowledging the next data received to imply there is a gap.
4 If it is legitimate not to receive all sent data, a receiver can simply not feedback anything if sufficient but not all data has arrived. This is a typical case used for many real-time Internet applications (but with longer-term feedback of receiver reports to cause longer term adaptation).
5 As with network marking above, this case is identical, except the sender can reduce the amount of data it sends, typically by using a lower quality encoding of a real-time stream.

© Copyright 2000, the Members of the M3I Consortium

6 This case is similar to the previous one, except drop is necessary to indicate congestion, instead of marking.
7 The final case is only possible with IP multicast. The sender side need do nothing except always send data in ‘layers’ that can be re-constituted at the receiver, the higher layers cumulatively adding more quality. If each layer is sent to a different multicast address (channel), individual receivers can ‘tune’ in or out of channels depending on the congestion each experiences on their own leg of the multicast tree. The term ‘drop layer’ is used to imply ‘tuning out’ of a ‘channel’.

In all cases, the protocol agreed makes explicit what the sender should do when sent congestion feedback by the receiver. However, it must always be remembered that the sender always has the option to react in its own best interest, rather than to keep to the protocol.

5 Example scenarios

Having discussed what should be in QoS control policies and in price reaction policies, and the various ways QoS can be controlled, we now move on to present a framework within which software to implement these policies can sit.

As in the M3I architecture [11], the following examples first describe scenarios with a price reaction service in *operation* then scenarios of how to *configure* such services. The various service building blocks used and the symbols for them are defined in the M3I architecture. Each is labelled with its name. Those labelled with an abbreviation use the legend given earlier in Figure 1. Only relevant service building blocks are shown in order to remove unnecessary detail. ‘Relevant’, is taken to mean ‘related to the scope of the flows in question’. How the flow scopes are classified is outside the focus of this discussion, and therefore left unshown for clarity.

As described in Section 2, as the ability to predict decreases, tightness of control has to increase. Thus price reaction only becomes important in scenarios where prediction is difficult or becomes incorrect over time (as likely to be due to the actions of others as to any misestimation). And, the tighter price control has to be, the closer it has to move to the end systems to avoid continual messaging to adapt the policy. This tendency can be seen in the following list of scenarios:

- fixed traffic profile per customer (diffserv)
- fixed traffic profile per flow, but can adapt (intserv)
- congestion charge reaction by proxy (e.g. risk broker [2])
- congestion charge reaction middleware on host
- congestion charge reaction by application

Moving from top to bottom of these scenarios increases the level of application control to take account of unexpected events. Conversely the level of over-estimation expected reduces. In the diffserv scenario, a traffic control agreement will remain for weeks or months, so price reaction hardly needs automation, being primarily a human activity. Moving to intserv, the traffic must be predicted for the duration of a flow. If the expected traffic is underestimated, the contract can be re-negotiated at the risk of being denied. Therefore price reaction is primarily important at flow set up. During transmission, the flow is at least protected from the vagaries of the actions of others. The next three scenarios assume that unpredictability has been exposed in the form of dynamic pricing. This caters for both the unpredictability of the customer and of everyone else crossing her path through the Internet. The third scenario uses a proxy function on a remote host, while the second

© Copyright 2000, the Members of the M3I Consortium

scenario uses a middleware function on the customer's own host. Therefore, in both cases, it can be assumed that the customer (or application writer) can predict traffic to some degree in order to give the proxy or middleware a policy to work with. However, if this function is on the local host, this policy can be adapted as circumstances change with only minor overhead. Whereas if the function is at a proxy, more prediction leeway will have to be allowed, as adaptation of the policy consumes more overhead. The final scenario is included to highlight that sometimes no-one can predict what will happen next. Here, the application has to take full control of QoS and reaction to its pricing, as there is no scope for describing the policy to adopt in case of all possible future events. A network game is a typical example of this scenario.

On this spectrum of tightness of price reaction control, the first is of little interest in terms of price reaction design, while the last is very difficult to design for. Therefore we shall focus on scenarios like the middle three. However, we will bear in mind, that the final (full application control) scenario will require functions in the application similar to those we design for more generic use. Thus we focus on generic design issues, whether for deployment in the network, host middleware, or within applications themselves.

6 Scenarios in operation

Figure 7 shows the relatively simple case of intserv charged purely by duration. The sender's ISP charges the sender for PATH messages (for their potential to cause a later reservation), while the receiver's ISP charges for successful RESV messages (which do set up a reservation). An alternative scenario (not shown) might have the sender charged for successful appearance of a RESV from the receiver(s), but the scenario in the figure also makes commercial sense. Also note that the receiver may choose to reserve less than or equal to the PATH message.

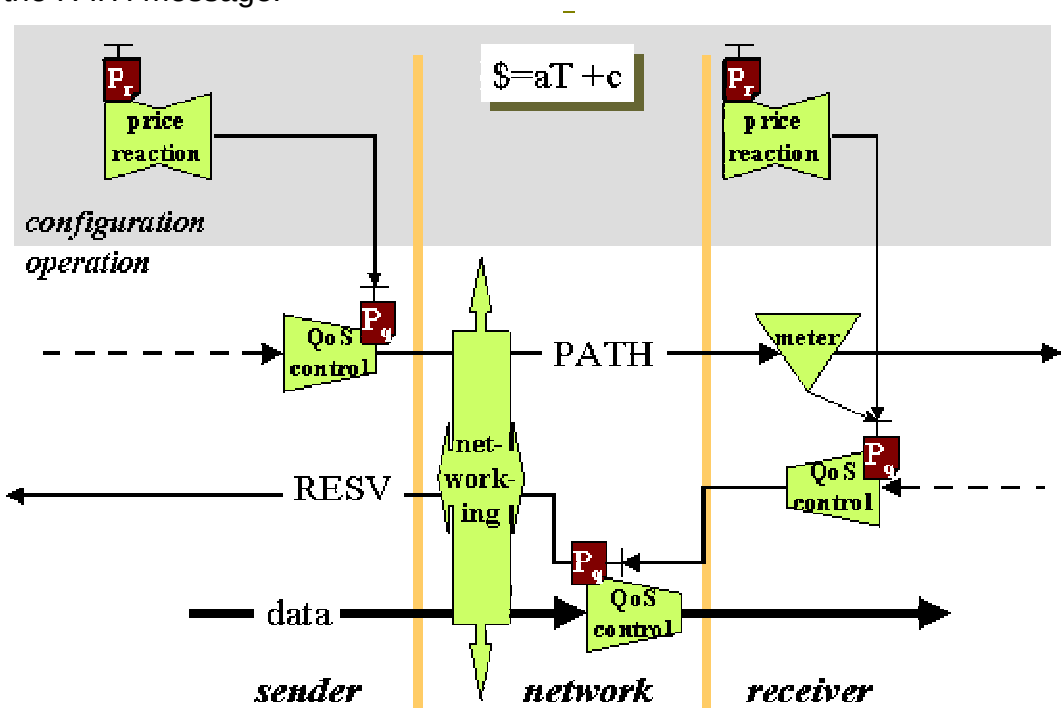

Figure 7: Duration charged intserv

The figure actually covers two scenarios, one without the dashed arrows and the other with. Without the dashed arrows, the QoS controllers shown are meant to depict those in the end

© Copyright 2000, the Members of the M3I Consortium

sender and receiver applications. These would typically take required QoS from the logic of the application or some directory of associations between task and QoS.

With the dashed arrows, the QoS controllers represent intermediaries in the path between sender and receiver, acting as proxies for each. An example of a proxy scenario might be some QoS control middleware on the host acting on behalf of an application to manage spending. Alternatively, a proxy might even be a separate organisation (the M3I GSP scenario [2]). Thus a proxy might take the required QoS from the upstream application and degrade it to keep to a budget, based on the QoS control policy rules.

The operational phases is shown in the bottom half of the figure, with the configuration phase shown on the shaded background above.

Whether the scenario involves proxies or not, the price reaction policies could include all or none of the complexity described in Section 3 in order to end up with a QoS specification for the QoS controllers. Just to briefly re-iterate, the simple case would be to input the required QoS directly to the policy, which would then merely have to take account of available budget. The more complex cases involve inputting approximations to utility functions for the context at hand. These are then used to determine the QoS specification, taking fuller account of the cost implications. In all cases, the price reaction policy function requires access to the tariff currently in use.

Whatever, once the QoS policy is determined, it is fed down to the QoS controllers and stays fixed in this scenario, throughout the session. Therefore, during operation, this scenario is extremely simple - all the complexity is in the configuration phase.

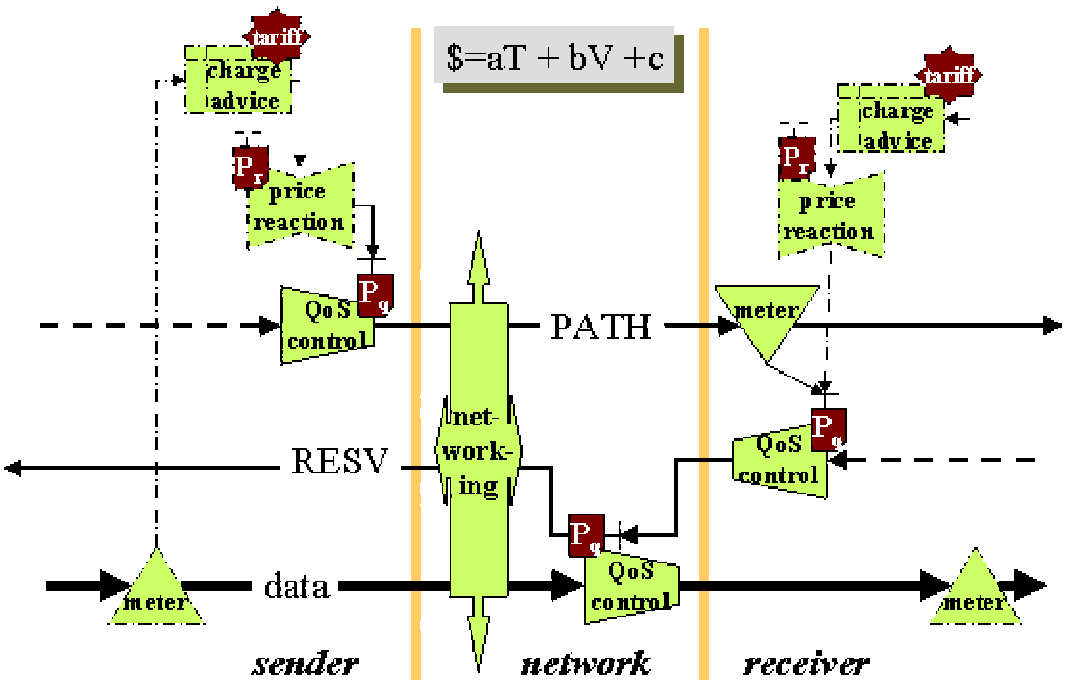

Figure 8: 'abc' charged intserv

Note that the receiver's QoS control policy uses the PATH message as an indication of the required QoS. The policy given to the QoS controller can then allow it to adapt if the PATH messages change, without requiring a new policy. This would be essential if we were dealing with the real receiving application, but even if it were a proxy, receiving a hint from the receiving application of the desired QoS, it may still be interested in listening to the maximum QoS possible, in case the budget allowed the QoS policy to allow the QoS

© Copyright 2000, the Members of the M3I Consortium

controller to upgrade QoS from that which the receiver initially believed was necessary, as opposed to only ever down-grading it.

The above assume the traditional intserv notion that the sender knows the QoS specification inherently, while the receiver learns it through RSVP PATH messages. See the discussion in the M3I architecture [11] on a generalisation of this architecture to a position where neither knows the QoS specification *a priori*, until informed through session control.

Price reaction becomes more interesting in a scenario such as that in Figure 8, where intserv is still the QoS architecture, but it is charged using the 'abc' tariff described in Section 3. This requires the addition of the chain dotted elements compared to the previous scenario (where the tariff was only aT+c, rather than aT+bV+c). Now, there is price back pressure on the actual volume of traffic sent under the reservation as well as the level of reservation itself. The motivations for this are described in [9]. This requires metering of data volume at both sender and receiver. The session characterisation from the meter (data volume in a certain duration) is then fed into the charge advice function, which uses the tariff to inform the price reaction function of the current charge for the volume charging element of the tariff. It can then take this into account when determining its overall reaction to both prices in the tariff, much as before, but with two degrees of freedom.

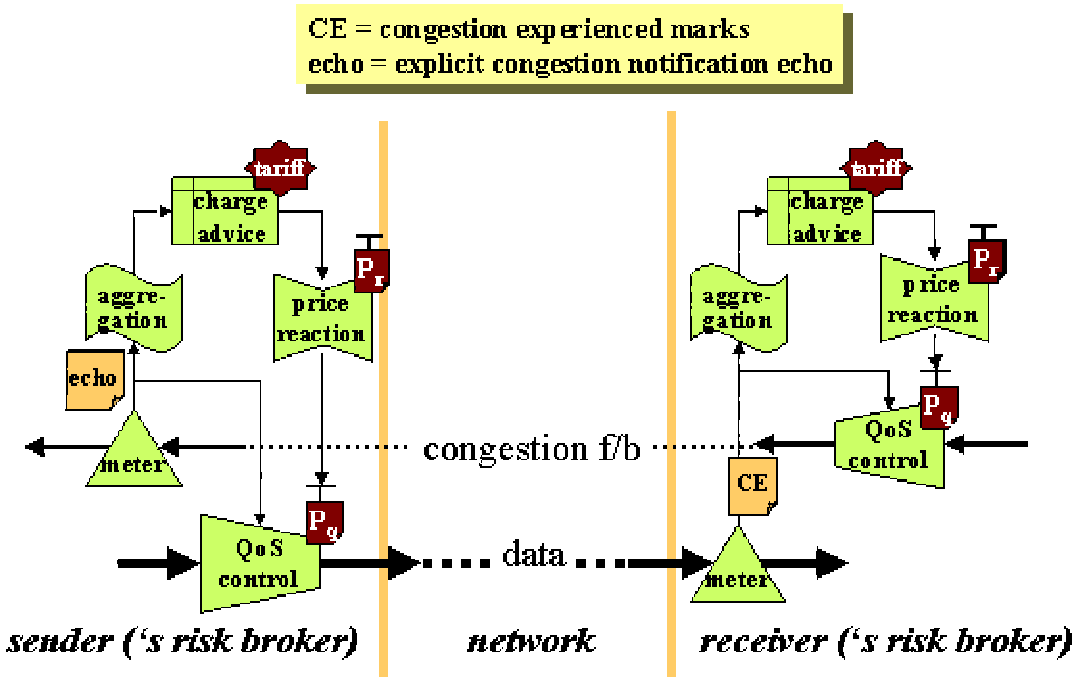

Figure 9: Congestion charging sender and receiver

Moving to the third and fourth scenarios, we now consider congestion charging, shown in Figure 9. Here, it will be noticed, there is no dependence between routing and QoS control in the network, as all QoS control for the flow in question is determined fully by the end-systems (or their proxies). Of course, there is effectively a QoS controller at every router determining the congestion marking rate of all flows, but this is not specific to this flow, and is therefore left unshown. This marking is denoted by 'CE' which implies setting of the experimental 'congestion experienced' bit [12] in each packet. As congestion approaches on any router in the path, the rate of marking increases. For this scenario, we assume the receiver's ISP charges a tiny amount for each mark. We then assume the sender is charged by the receiver for any marks fed back (verifiable on longer time-scales by the receiver's ISP). The CE marking rate in the data is measured at the receiver and fed directly into its

© Copyright 2000, the Members of the M3I Consortium

QoS controller, which directly controls the amount of congestion feedback to the sender. On longer time scales, this rate is aggregated and fed into a (rudimentary) charge advice function, which sets the current longer term price of the marking rate as input to the QoS control policy. This allows the QoS control policy to calculate the current willingness to pay and hence optimum QoS given the receiver's utility function.

The feedback behaviour of ECN is only defined for TCP, which is effectively in the 'preserve all data' class of reactions in Table 1. For applications that can tolerate never sending some potential data, we have to assume how the ECN feedback protocol will work. For this scenario, let us assume that the receiver only indicates congestion when it wishes the sender to reduce its rate. It charges the sender whenever it does this. The receiver's ISP can verify the total level of congestion marking in the session at the receiver, so that the sender can be assured the receiver isn't trying it on (assuming the receiver and its ISP aren't colluding). The functions at the sender, are then identical to those at the receiver, in terms of how it meters the congestion feedback rate (rather than marking rate) and controls its data sending rate.

Note that we have only discussed charges by the receiver's network provider (whether paid by sender or receiver). The sender's network provider may charge on any other basis, but cannot charge based on congestion itself, as this information is not always available to it. Thus the sender's charging strategy should be treated as orthogonal to congestion charging.

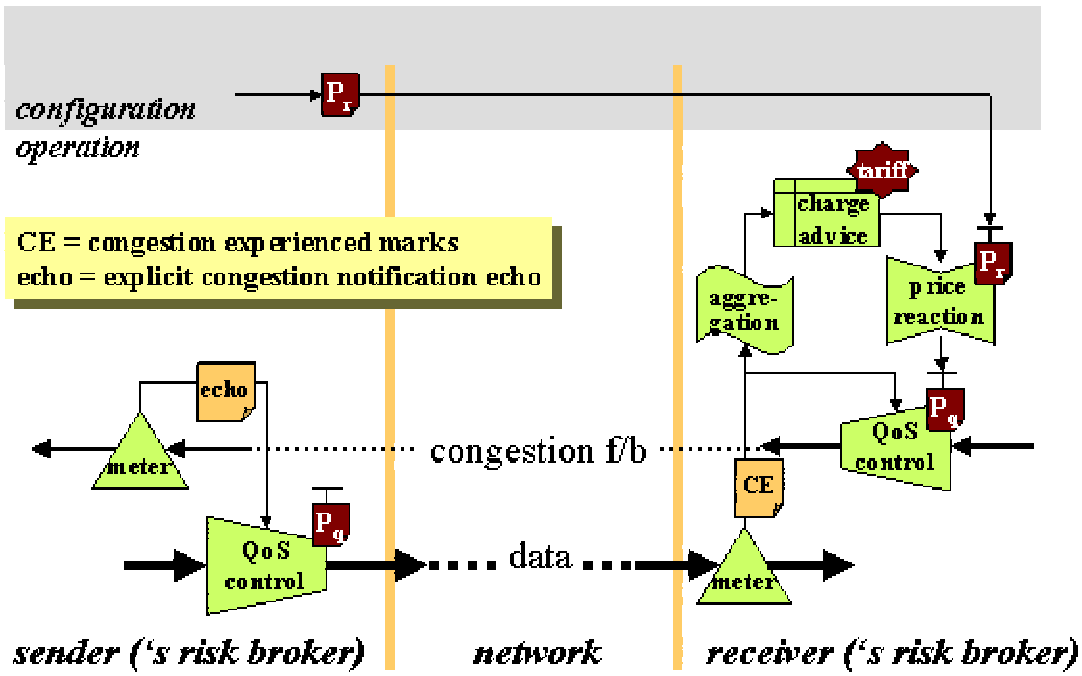

Figure 10: Congestion charging receiver directly, sender indirectly

A simpler alternative would be for the QoS control policies at sender and receiver to belong to whoever is paying. If, for instance, the sender is paying for both ends, it might include its QoS control policy in the session description of the session set-up message (Figure 10 - perhaps using SIP [13]). The receiver would then use the sender's policy to determine its QoS control feedback. The sender's policy might set an upper limit on the cost, which the receiver might even choose to improve on if it doesn't need the QoS the sender is allowing (assuming in this case that the two parties are friendly).

© Copyright 2000, the Members of the M3I Consortium

6.1 Scenarios for set up

As promised, we now briefly describe the context within which the above service scenarios operate, in terms of how they are created and configured. Figure 11 is similar to the generic QoS control configuration figure in the M3I architecture [11], which should be referred to for a general description of how a QoS control policy is found in the first place. In that description, we left undefined what application was driving the process. In Figure 11 we have chosen to show a buying agent separate from the application setting the QoS control policy. Our motivation is to enable price-controlled QoS for applications without the application having to be modified. Certainly we would hope to avoid any need for applications to understand pricing issues, but there is also the possibility that QoS can be added to applications through client middleware using session control to add the necessary information.

The buying agent sets the price reaction policy by finding the context in which a particular flow is being used by an application, looking this up in its database of policies, then applying the relevant policy to the QoS controller. Context might typically be discovered through interrogating session description information from the session control function. The buying agent would deal with generation of new policies where an appropriate one cannot be found (perhaps using an interface similar to that in Figure 3, or using the Intelligent Agent to teach it the new policy). The other addition in this figure to those objects discussed in the M3I architecture is a query to an object holding a record of the current budget the customer wishes to use, in order to scale the utility function it uses to the budgetary context. The ability to control QoS by price, without major alterations to applications is one of our primary requirements. However, it is unclear how we can find the context in which an application is using the network without the application revealing this explicitly. We can fall back on adding context information into a session description, as above, but this is not always available. Such a capability is very application and operating system specific, even to the extent that the application name is sometimes not available directly.

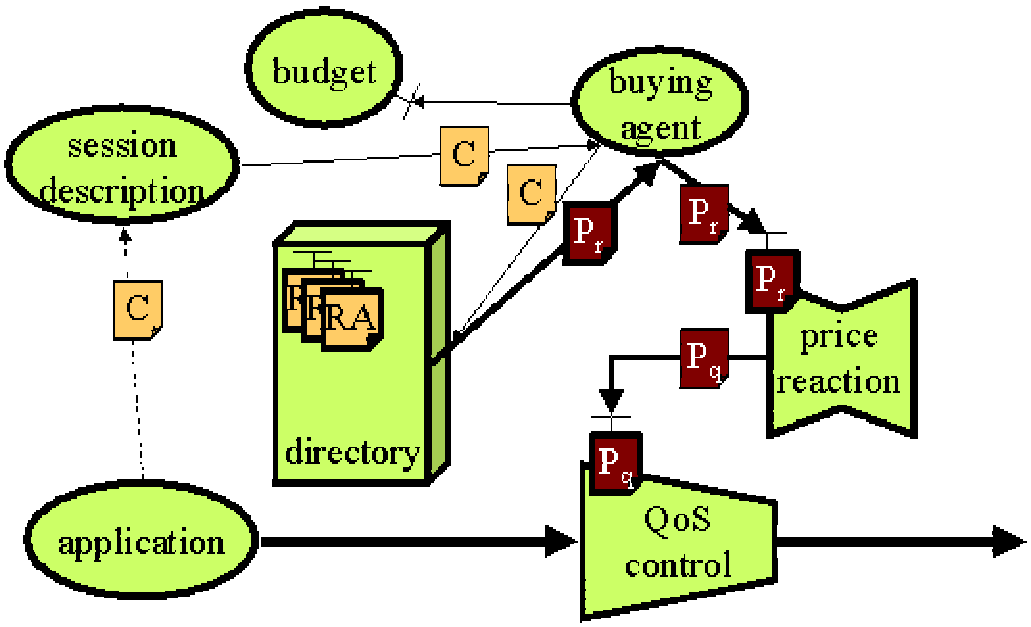

Figure 11: Price reaction in context

© Copyright 2000, the Members of the M3I Consortium

We are clear, however, on the importance of abiding by the overriding principles of application involvement. Even if the application is not modified to manage QoS, it is necessary to insist that it should at least delegate control of its quality to another entity, such as a QoS controller, or a buying agent and QoS controller combined, but only for 'non-functional' communication quality.

We define non-functional communication quality as that which can be controlled by policy, because its requirements can to some extent be predicted in advance. On the other hand, functional communication quality is directly under the control of the application, and it makes no sense for another entity to override it. Any price reaction for functional quality must be through making the user or application directly aware of the cost of her or its actions, so that she or it may react according to rules that will be completely application or user-specific.

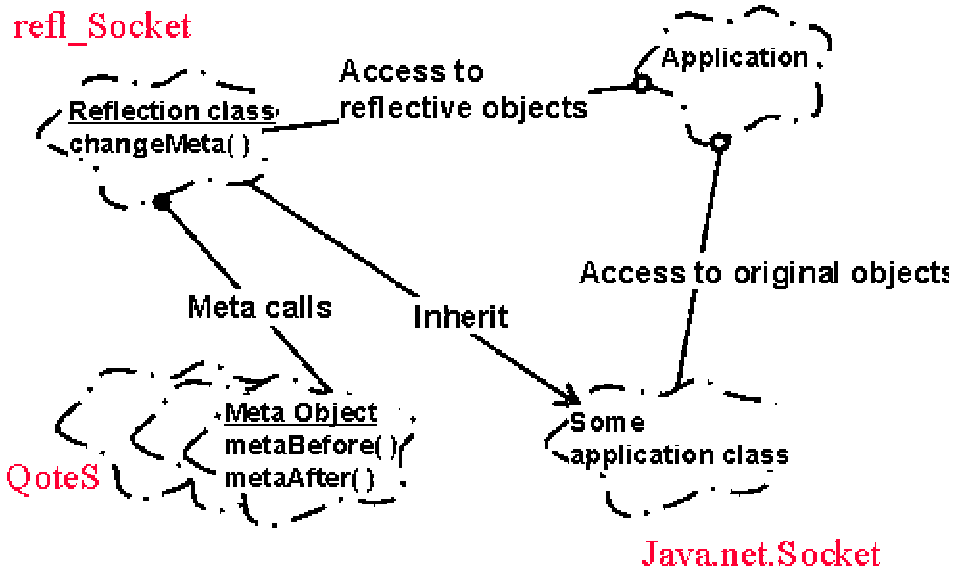

Figure 12: The meta-object design pattern

In BT's earlier work [14] use of the meta-object design pattern is recommended to achieve delegation of control for non-functional QoS. This requires a minimal decision by the application programmer to open a differently named socket from normal if she wishes to delegate QoS control to some other entity for that socket. All parameters remain the same as for a regular socket because the reflective socket inherits its behaviour from the regular socket. The details are shown in Figure 12, which is taken from that work. Essentially, the application is edited to open such sockets through a point of indirection. This point of indirection can then be controlled independently, to determine what happens on opening and subsequently closing a socket with delegated QoS control. In our example implementation of a receiver, a reservation was set up on creation of the socket, and torn down on closure, but only if the presence of PATH messages was detected from the sender. Thus, the context was supplied by the sender in this case. This still leaves open the question of how context is shared in other scenarios, if session control information is not available.

Thus the general principle is that another agent shouldn't take control of QoS away from an application unsolicited. This is because, the application must remain in control, otherwise, future versions of the application which include some QoS control could well conflict with the attempts of another agent to control QoS. The only exception to this, is if the user becomes involved.

© Copyright 2000, the Members of the M3I Consortium

The user is entitled to delegate control over any application's network QoS to any other agent if she so desires, and can obviously supply context if necessary. Obviously, the user is then responsible for any conflicts with the application. Similarly, as already said, the user is responsible for price reaction for functional actions taken directly by herself. User charge advice feedback is designed into the system for this purpose.

7 Limitations & Further Work

There is a view, being expressed from the economic modelling task of the M3I project, that modelling the utility of the network service is futile as soon as the customer is buying a bundle that merely includes the network service as one part. This discussion is obviously fundamental to the work described in this paper, and will continue until resolution.

The approximations conjectured in the QoteS implementation shown are definitely the subject of further research and validation by experiments as they involve taking considerable liberties with unproven assumptions (and Java GUI APIs!).

In general, it is recognised that this area of research is new and fragile, with a lot more work required to establish even the correctness of the basic assumptions, such as the conceptualisation of customer utility of network QoS.

8 Conclusions

Background work by BT and AUEB in the field of controlling network quality of service through dynamic and not so dynamic pricing has been described. A software framework consistent with the M3I architecture has been presented which will allow these price controlled QoS algorithms to be configured into a working system while it is running, to help enable a market managed multi-service Internet.

Acknowledgements

Konstantinos Damianakis, Jérôme Tassel, Mike Rizzo, Kennedy Cheng & Francesco Manganotti (BT); Panayotis Antoniadis & George Stamoulis (AUEB).

References

[1] Scott Shenker (Xerox PARC), "Fundamental Design Issues for the Future Internet", IEEE Journal on Selected Areas in Communications, 1995. URL: <http://www.spp.umich.edu/spp/courses/744/docs/FundamentalDesign-shenker.pdf>

[2] Ragnar Andreassen (ed) (Telenor R&D), "Requirements specifications; Reference model", M3I Eu VthFramework Project IST-1999-11429, Deliverable 1, Jul 2000, <URL:http://www.m3i.org/private/>

[3] Bouch, A., Sasse, M., & DeMeer, H. G (UCL), "Of packets and people: A user-centred approach to Quality of Service", In Proc. IWQoS'00, May 2000, <URL:http://www.cs.ucl.ac.uk/staff/A.Bouch/42-171796908.ps>

[4] S. Blake (Torrent), D. Black (EMC), M. Carlson (Sun), E. Davies (Nortel), Z. Wang (Bell Labs Lucent), W. Weiss (Lucent), "An Architecture for Differentiated Services", IETF RFC 2475, Dec 1998 <URL:rfc2475.txt>

© Copyright 2000, the Members of the M3I Consortium

[5] R. Braden, D.Clark, S.Shenker, "Integrated Services in the Internet architecture: an overview", IETF RFC 1633, Jun 1994. <URL:http://www.isi.edu/div7/rsvp/pub.html>

[6] M3I project Web site <URL:http://www.m3i.org/>

[7] R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, S. Jamin, "Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification". IETF RFC 2205, Sep 1997, <URL:rfc2205.txt>

[8] Pete Bagnall, Bob Briscoe & Alan Poppitt (BT), “Taxonomy of Communications Requirements for Large-Scale Multicast Applications”, IETF RFC 2729, Dec 1999, <URL:rfc2729.txt>

[9] Dave Songhurst (Lyndewode) (ed), "Charging Communications Networks; From Theory to Practice", Pub: Elsevier, ISBN:0-444-50275-0

[10] {Ed: Suitable reference on optimisation of willingness to pay}

[11] Bob Briscoe (BT) (ed), “Architecture”, M3I Eu VthFramework Project IST-1999-11429, Deliverable 2, Jul 2000, <URL:http://www.m3i.org/private/>

[12] K. K. Ramakrishnan (AT&T Labs Research) and Sally Floyd (LBNL), "A Proposal to add Explicit Congestion Notification (ECN) to IP", IETF RFC 2481, Jan 1999 <URL:rfc2481.txt>

[13] M. Handley, H. Schulzrinne, E. Schooler, J. Rosenberg , " SIP: Session Initiation Protocol", IETF RFC 2543, Mar 1999, <URL:rfc2543.txt>

[14] Jérôme Tassel, Bob Briscoe, Alan Smith, (BT), “An End to End Price-Based QoS Control Component Using Reflective Java”, in Lecture Notes in Computer Science from the 4th COST237 workshop, pub. Springer-Verlag, Dec 1997, <URL:http://www.labs.bt.com/people/briscorj/papers.html#QoteS>

[15] Konstantinos Damianakis, Mike Rizzo, Jérôme Tassel & Bob Briscoe (BT), “QoS around the edge (QoteS), release III”, software (unpublished). Based on [14] & [17]

[16] Martin Karsten (TUD) (ed), "M3I Pricing Mechanisms (PM); Design" M3I Eu Vth Framework Project IST-1999-11429, Deliverable 3, Jun 2000, <URL:http://www.m3i.org/private/>

[17] Bob Briscoe, Mike Rizzo, Jérôme Tassel & Kostas Damianakis, (BT), "Lightweight Policing and Charging for Packet Networks", in Proc. IEEE OpenArch 2000, pp77-87, Tel Aviv, Israel (Mar 2000), <URL:http://www.labs.bt.com/people/briscorj/papers.html#e2char_oa00>

[18] C. Courcoubetis, G.D. Stamoulis, C. Manolakis, and F.P.Kelly, “An Intelligent Agent for Optimizing QoS-for-Money in Priced ABR Connections”, Presented: ICT’98 Porto Carras, Greece, 24 Jun 1998, To appear: Telecommunications Systems; Speical Issue on Network Economics.

Abbreviations

ABR	Available Bit Rate (an ATM service class)
API	Application programming interface
ATM	Asynchronous Transfer Mode

© Copyright 2000, the Members of the M3I Consortium

AUEB	Athens University of Economics and Business
BT	British Telecommunications plc
CE	Congestion Experienced (part of the ECN protocol)
diffserv	IETF differentiated services
DPH	Dynamic Price Handler
ECN	Explicit congestion notification
Echo	Echo of congestion experienced (part of the ECN protocol)
GSP	Guaranteed Service Provider
IETF	Internet Engineering Task Force
GUI	Graphical user interface
intserv	IETF integrated services
IP	Internet Protocol
ISP	Internet Service Provider
M3I	Market Managed Multi-service Internet
QoS	Quality of Service
QoteS	QoS around the edge
P_r	Price reaction policy
P_q	QoS control policy
PATH	An RSVP message from the data sender to routers on the path, and onward to the receiver
RESV	A RSVP message from the data receiver to routers on the path reserving resources
RSVP	Resource Reservation Protocol
TCP	Transmission Control Protocol
VoD	Video on Demand

© Copyright 2000, the Members of the M3I Consortium