Notes on a QoS information model for making adaptation decisions

Saleem N. Bhatti and Graham Knight

Computer Science Department, University College London, Gower Street, London WC1E 6BT, UK

Abstract: We present a network model that allows processing of QoS (quality of service) information about media flows to enable applications to make adaptation decisions. Our model is based around a multi-dimensional spatial representation that allows QoS information describing the flow constructions and QoS parameters – flow-states – to interact with a representation of the network QoS. The model produces reports about the compatibility between the flow-states and the network QoS, indicating which flow-states the network can currently support. The simple nature of the reports allows the application to make decisions, *dynamically*, on which flow-state it should use. The model is relatively lightweight and scaleable. We demonstrate the use of the model by simulation of a dynamically adaptive audio tool. Our work is ongoing.

Keywords: QoS (quality of service), QoS summarisation, adaptive applications, Internet applications

1. Introduction

In a best-effort network such as the Internet, there is no guarantee of the network QoS that a particular application instance might receive. The QoS may fluctuate due to routing effects or traffic interactions in the network [Pax97a] [Pax97b], or the application may be resident on a mobile host [Kat94]. There is currently great interest in making applications adaptable to changes in network QoS. This is of particular importance for real-time media flows or flows that are sensitive to QoS fluctuations. Much of the attention for this work has focused on mechanisms that allow adaptability of the media flow construction by scaling (e.g. [D+93]), filtering (e.g. [YGHS96]) or encoding techniques (e.g. [HSHW95] [AMV96]). An area that has received little attention is how applications can dynamically select the correct flow construction to match the available network QoS. The application must currently rely on the user to set the correct preferences to allow operation in a particular QoS regime. Although application specific mechanisms exist to allow some automatic adaptation (e.g. elastic buffering to combat jitter in audio tools), we would like to offer a more general model to allow applications to dynamically adjust their flow construction.

In this paper, we start by considering the interactions required between the user, the application and the network in deciding how flows should adapt (Section 2) and consider existing work (Section 3). We then present our model and a short analysis of its function (Section 4) followed by simulation of three scenarios showing an audio tool that uses our model (Section 5). We end with some short concluding remarks (Section 6).

2. The user, the application and the network

To allow adaptation, there must be interaction between the user, the application and the network. The network must supply information about the QoS that is being provided to the flow. The user must specify preferences that govern the behaviour of the application. It is then up to the application to decide how adaptation should take place based on both of these pieces of information. Our area of interest is shown by the dashed box in Figure 1. We think of the application as having well-defined **modes** that represent operating conditions for the application. Associated with the applications modes are media flow-states, **QoSStates**, that represent the operating conditions for flows. The **QoSSpace** is our network model and the QoSStates conceptually exist within the QoSSpace. The **QoSEngine** maps network state (derived from real QoS parameter measurements for the flow) into the **QoSSpace**. (We do not consider the QoSEngine in detail in this paper, but it does form part of our work.) The QoSSpace issues QoSReports that contain a **state compatibility value (SCV)** for each flow-state from the application. The application then combines the SCVs with other application specific information to make an adaptation decision using the **application adaptation function (AAF)**. We say more about

QoSStates, QoSSpace, QoSReports and SCVs in Section 4, and show an example of an AAF in Section 5. In Figure 1, we show only one flow but many are possible. The definition of a flow is application specific.

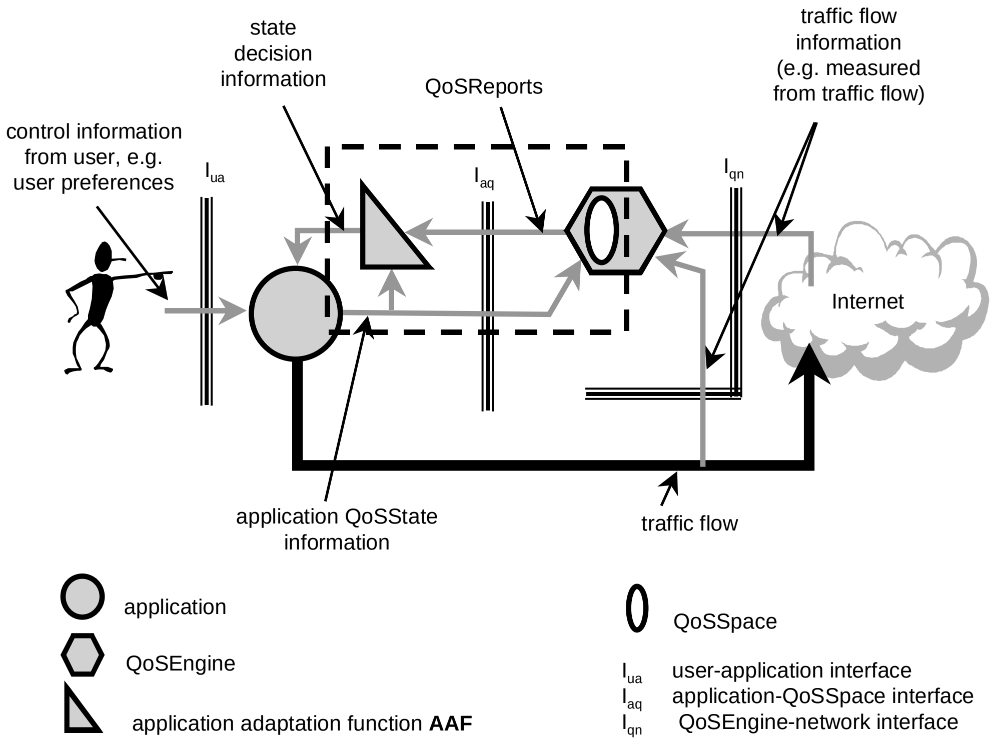

Figure 1: Interaction between the user, the application and the network

Our goal is to allow the application to make adaptation decisions in response to fluctuations in QoS seen by a flow, but the adaptation process should be under the control of the user.

2.1 Interactions between the user and the application

We do not consider in detail the mechanisms for interactions with the user but we discuss the requirements of the user in allowing the application to make automatic adaptation decisions.

There may be many instances of an application used by many different users. Each of the users will have their own preferences for the application. Any decision-making process for adaptation must allow the application to apply user preferences to the information about QoS being experienced by the flow. For example, with an audio tool with numerous encoding schemes, one user may specify that the tool “always uses the best audio quality possible” while another user may specify that the tool aim to “retain stability” in the face of network QoS fluctuations. The application may interpret these preferences as meaning the “encoding with the highest data rate requirement whenever possible” and “do not change flow-state too often”, respectively. So the adaptation decision is not only application specific but also application instance specific, and controlled by user preferences.

Typically, we would like user preferences to reflect the fact that users may have minimum technical knowledge. User **QoS preferences** may be expressed as an application specific enumeration such as “high”, “medium” or “low” quality. As the application will be making state changes, the user may wish to specify **QoS stability** criterion, i.e. do not change state more than once every minute. Note that the this last criterion is the user’s request for the application to not change to a “better” quality state too often, i.e. avoid state-flapping; if the QoS degrades, then we would expect the application to adapt as required.

Such heterogeneity means that the adaptation decision information reflects a per instance view. So the information used by the application to make adaptation decisions should:

- not unduly constrain the QoS mapping from user requirements to application capabilities
- be amenable to the mapping of the user QoS preferences and QoS stability criterion on a per application instance basis

2.2 Interactions between the application and the network

Before an application instance can make decisions about any changes in its behaviour, it needs to know what is “sensible” for its flows, i.e. what its current network connectivity can support. This requires some information from the network. Information about the network is typically expressed as values of QoS parameters such as delay, jitter, available capacity, loss, etc. This information may be received in a number of ways:

- via local mechanisms, e.g. from the communication stack on the host
- via application specific mechanisms, e.g. via proprietary signalling
- as control messages from the remote receivers of the flow, e.g. using RTP/RTCP [RTP]
- from network management tools, e.g. using SNMP [SNMPv2]

If we consider that “the application knows best”, then we must appreciate that there is likely to be no single “best” ubiquitous solution for getting information about the resources within the network. The “best” mechanism may depend on the network environment or the application’s function or both, but only the application (application designer) is in a position to make that assessment.

The application must also have some way of expressing its functional capability in terms of the construction of its flows, i.e. what states its flows can take. Typically, these will be performance bounds defined in terms of QoS parameter values for the states of a flow, e.g. minimum data rate required, maximum delay, etc. Note that as a definition of a flow is application specific, so is the timescale over which measurements of such performance bounds are measured. This means that our model must be capable of working in whatever timescales are specified by the application.

QoS parameters may include local (end-system) resources, not just network QoS measurments. For example, on mobile systems, measurements of battery power or host load [YL95] [FPM95] may also be used for making adaptation decisions.

So, our network model should be able to process information about QoS parameters, but:

- should not put any constraints on where that information (measurements) should come from
- should be able to accept a wide range of QoS parameter, whether end-system specific or network specific
- be able to cope with any timescale over which that information is measured

2.3 Making adaptation decisions

In Section 2.1, we saw that the user preferences introduce heterogeneity and that the information required to make an adaptation decision may reflect a per-application instance view of QoS. A mechanism is required that can give an indication of the ability of the network to support any of a number of flow-states that an application instance might take. In a distributed application, there may be other application-level signalling involved before adaptation can take place. Also, the application modes may be functions of other application specific information, so our model can not *make* a decision for the application, but offer *QoS summaries* – **QoSReports** – that represent a view of the relative compatibility of the network QoS and the application’s flow capability. However, in general we are not aware of the following application specific details:

- the nature in which information from a current QoSReport must be evaluated with information from previous QoSReports in order to make flow-state change decision
- the nature in which QoSReports must be evaluated with other (application specific) information in order to make flow-state change decision

There may be a statistical or temporal sense in which our QoSReports have meaning to the application with respect to its current mode or flow-states. The application may have synchronisation constraints between its application flows. Such matters can only be assessed by the application. So, we chose to separate the network-related flow-state information from other application specific information. We chose that the flow-states be expressed as **QoSStates**, flow-state information that is specific only to the QoS requirements and constraints of the individual application traffic flows. For example, an audio application may have several different audio encoding techniques and each would be a separate QoSState for that audio flow. Furthermore, as we do not know how the application will use the QoSReports from our model, we should simply issue QoSReports that are based on an assessment of the instantaneous QoS of the network – a snapshot in time. This is also consistent with our requirement for our model to work within the timescale constraints of the application.

Additionally, our model should aim to relieve the application from having to deal with raw QoS parameter measurements[1]. Our aim is to offer the application a method of assessing the need to make flow-state changes, so the granularity at which we report information from our model should be per-flow. The report of the per-flow state information needs to be a suitable summary of the QoS parameter measurements for a flow. In producing the summary, we must be careful not to lose the significance of any single QoS parameter that defines the flow state (e.g. a mean might “smooth out” a low or high QoS parameter value).

So, the QoSReports our model offers to the application are per-flow QoS summaries that:

- should be in a form that can be easily interpreted by a distributed application if required
- are derived form QoS parameter values, indicating the relative compatibility between the (instantaneous) network QoS and of an application’s QoSStates

3. QoS assurance and dynamic adaptation

For the provisioning of end-to-end QoS, the user/application is telling the network what is required and asks that the network should adapt/configure itself to comply to the application’s requirements, i.e. static adaptability. In enabling dynamic adaptability, the application is constantly (i.e. within application specific time scales) receiving information about the QoS that the network can offer, and then changing its flow-state to comply with the network capabilities.

3.1 QoS assurance

In [CAH96], there is an excellent description of the elements of a general QoS architecture for assuring QoS. With QoS assurance mechanisms, applications have to specify what they want the network to do for them, and the network tries to honour this request. The network itself may provide adaptability mechanisms [Cam97] to cope with fluctuating QoS or heterogeneity due to network architecture. [CAH96] suggests that (among other things) the application should be in control of the following information that passes to the network: QoS service-level (e.g. guaranteed-service, controlled-load service) and QoS management policy (how adaptation should take place when there are QoS fluctuations). These affect the way the application is integrated with the network (the separation of functionality between the network and the application), and how heterogeneity is supported (at the user, application and network access level).

QoS assurance mechanisms are based around the use of resource reservation and QoS re-negotiation to achieve the desired QoS for the application. Mechanisms such as RSVP [RSVP] are designed to provide resource reservations in the Internet. However, RSVP can fail [WGS97], and adaptability mechanisms in the network become impracticable and increasingly difficult to support in *ad hoc* or mobile network architectures. Additionally, where resource reservation and QoS re-negotiation are used, they should be available end-to-end for maximum utility. In today’s Internet there is very little resource reservation support and network components lack the programmability required to enable adaptability in the network. Furthermore, as such mechanisms do become available, they will not be deployed uniformly, and this will further compound the heterogeneity in the Internet. For many applications, such QoS assurance or adaptability mechanisms within the network may only be useful if they are available end-to-end. Where these mechanisms do exist, there is great value in being able to use them, but in the “standard” best-effort Internet, other mechanisms are required to address adaptability.

The QoS service-level required may be important to the individual users and may determine the cost of the service, e.g. guaranteed-load is a “better” service than controlled-load so may cost more. We argue that the service-level should not be specified by the application. The application should be prepared to be more flexible in its adaptation capability, leaving service-level selection to the user. Three reasons for this are:

1. the service-level may determine the cost of the service and users usually wish to control how much they pay
2. network heterogeneity, lack of resource reservation or network element failure may mean that a particular service-level is not available at a given time at a given point in the network
3. new, additional service-level definitions may be introduced that are more suitable (in terms functionality or cost) for use in a given situation (e.g. there are two relatively recent descriptions of **adaptive** service-level

[1] This does not preclude the use of such measurements for fine-tuning if mechanisms such as elastic buffering are in use, but we are working at a coarser level of flow granularity and over different timescales. Indeed, the value of the QoSParams produced by our QoSEngine may be more suitable for such use than raw values.

[CCH96] [LLB97]), and differentiated services may be set up that are domain/administration specific [DIFFSERV]

Additionally, if cost-based feedback is available from the network, then cost could be treated as a QoS parameter – although not related to the performance of the flow it acts as a defining constraint in the same way as, say, defining minimum data rate or maximum jitter for a flow. The value of making the cost explicit, as a QoS parameter, is that it highlights the importance of cost as a feedback control mechanism in future services [SCEH96].

3.2 Dynamic adaptation

QoS management policy will be subject to user preferences and application specific behaviour. Applications may find it useful to have a specification of the QoS management policy before the application starts operating. This would certainly be of value to the network for controlled allocation of resources, and makes sense in the context of trying to assure end-to-end to QoS. However, in our consideration of dynamic adaptability, the use of the application typically requires interaction with the user in order to determine its adaptation requirements, and these may not be known until after the application is running. In [CAH96], the QoS management policy:

> *captures the degree of QoS adaptation (continuous or discrete) that the flow can tolerate and the scaling actions to be taken in the event of violations to the contracted QoS [CCH94].*

We chose to make a separation between what “*the flow can tolerate*” and the “*scaling actions to be taken*”. We argue that the former is a property of the media and the latter is an application specific requirement that includes interaction with the user. Flow performance specifications can be used to indicate the flow-states that are possible for a flow and can be determined by the application designer. The action to be taken on fluctuations (“*violations*”) of QoS is a dynamic adaptation decision and cannot be determined by the application designer *a priori*. It is the difference between the application designer saying “I know what is sensible for the application” and the user saying “I know what is sensible for the application to do for me”. Ultimately the application’s functional constraints have the final say on which flow-state(s) is (are) functionally *possible*, but this should not dictate *how* the user would like the application to behave, i.e. how adaptation should take place. [LSD98] points out that the user requirements and the network QoS may change throughout the session and proposes that the user should be given the opportunity to make informed decisions about application adaptation. As an example, consider a remote teaching scenario and the requirements of the audio and video flows. When operating in lecture mode (main part of the teaching session), the conferencing application may tolerate relatively high delay and throughput is asymmetric, but during a question and answer session (at the end of the teaching session), low delay and jitter are required with symmetric throughput for flows.

The application must be able to assess the user preferences and available network QoS in order to make automatic and dynamic adaptation decisions.

4. The QoSSpace

Our concern is that we try and assess the overall relative *compatibility* between the flow QoSStates and the network QoS rather than try to evaluate the *equality* (or otherwise) of the absolute values of the QoS parameters for a flow-state and the measurements taken from the network. We would like our model to indicate *how well* the network QoS matches the requirements for any of an application’s flow-states.

4.1 QoSSpace, QoSParams and QoSStates

The **QoSSpace** is a multi-dimensional space in which flows conceptually exist, and into which the network QoS is mapped. Flows are represented by **QoSStates**, and each flow may have a set of QoSStates. The dimensions of the QoSSpace are represented by a set of **QoSParams**. This is depicted in Figure 2(a), which shows only three QoSParams, but any number of QoSParams are possible.

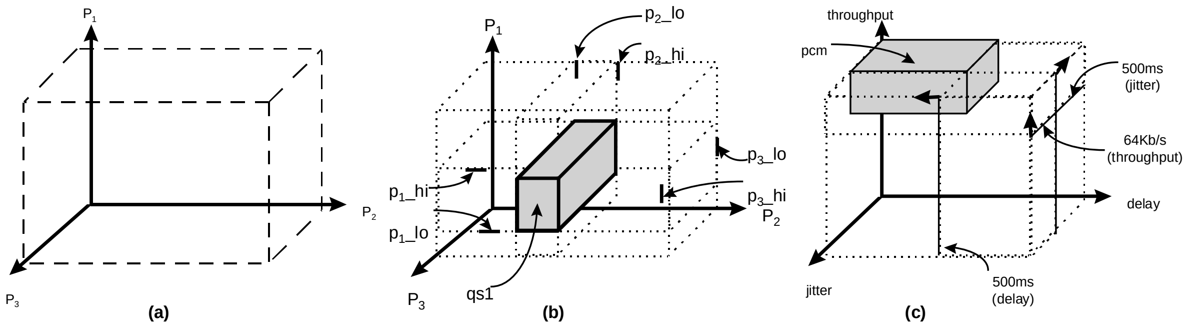

Figure 2: QoSSpace; (a) example with 3 QoSParams; (b) example QoSState; (c) a single QoSState for an audio tool

The QoSParams are variables that are representations of real QoS parameters, such as throughput, delay, jitter, etc. These are chosen to suit the application, i.e. the dimensions of the QoSSpace are application specific. QoSSpace effectively gives a snapshot in time, and the interval over which measurements are taken will be application specific. The network QoS is evaluated and mapped into the QoSSpace by some appropriate, application-specific mechanism. We chose to separate the abstraction of the QoSSpace from the mechanisms performing the mapping of the network QoS[2]. An example QoSState is shown in Figure 2(b). This spatial description effectively defines a region in QoSSpace in which the flow can operate. We define a QoSState for a fictitious flow in terms of the QoSParams P_1, P_2 and P_3. We use simple rectilinear, boundaries, which we call ***_hi*** and ***_lo***:

$$
\begin{aligned}
qs1 = \{&\langle P_1, p_1\_lo, p_1\_lo, p_1\_qlo, p_1\_qhi\rangle, \langle P_2, p_2\_lo, p_2\_lo, p_2\_qlo, p_2\_qhi\rangle,\\
&\langle P_3, p_3\_lo, p_3\_lo, p_3\_qlo, p_3\_qhi\rangle\}
\end{aligned}
$$

This statement identifies a QoSState called *qs1*, defined by tuples for the QoSParams P_1, P_2 and P_3. *qs1* consists of a set of tuples, each of which has the structure:

$$\langle id, \_lo, \_hi, \_qlo, \_qhi\rangle$$

where:

id	a name identifying the QoSParam
_lo	low threshold value of the QoSParam
_hi	high threshold value of the QoSParam
_qlo	QoSiState low threshold
_qhi	QoSiState high threshold

(We defer a description of QoSiStates until Section 4.4.) For any application flow:

- QoSStates for a flow need not be specified with the same number of QoSParams
- for any QoSParam tuple, either *_hi* or *_lo* threshold may be left undefined, but one must be present
- QoSStates may overlap

As an example of a QoSState for an (imaginary) audio tool, consider the definition of the QoSState *pcm* in Figure 2(c).

$$pcm = \{\langle throughput, 64, -, -, -\rangle, \langle delay, -, 500, -, -\rangle, \langle jitter, -, 500, -, -\rangle\}$$

This says that the state *pcm*, requires a minimum of 64Kb/s throughput, and can tolerate a maximum of 500ms delay and a maximum of 500ms jitter. Other QoSStates may be possible. Note that the model is not concerned with the semantics of the QoSParams, or any relationships between them – this is left for the application to control. Also, different QoSParams may exhibit strong correlation from the QoS parameters that they are derived from (e.g. on some networks, delay and throughput may be related – as delay goes up throughput goes down), and so it may be possible to reduce the number of dimensions. This, again, is an application level issue. The application may choose the complexity of the definition of the QoSStates, as required. However, the

[2] In our work, the QoSParam values for the network QoS, NetQoSState (see Section 4.2), are generated by the QoSEngine, but we do not describe this in detail here. Indeed, other application-specific, mechanisms may be used in place of the QoSEngine.

QoSSpace must be defined by the set of QoSParams that are the union of all the QoSParams for all the QoSStates for that flow.

4.2 NetQoSState

We now need a mapping of the network QoS into the QoSSpace. We achieve this by using QoSParams to describe the network QoS, **NetQoSState**. The NetQoSState will have the same dimensions as the QoSSpace. QoSParam values are estimates of the current value of the QoS parameter, based on real network measurements of QoS parameters for a flow. A simple interpretation of the NetQoSSpace would be to use the values of the QoSParams. This would translate to a point within the QoSSpace:

$$q = \{p_n\} \quad \forall\, n = 1 \ldots N,\, p_n \in P_n$$

i.e. $\{p_n\}$ are the values of the set of QoSParams $\{P_n\}$ that define the QoSSpace However, delay and noise effects mean that associated with the QoSparams is an *uncertainty* in our QoSparam estimations. As the network QoS fluctuates we may have different degrees of uncertainty. If the network is in a steady state, we may have a lesser degree of uncertainty than if the network is currently showing fluctuations in the QoS offered. Measures such as standard deviation are normally used to indicate such uncertainty. However such measures only have meaning in a statistical sense when we have some knowledge of the model of our traffic and/or the network. What we are actually after is an indication of the current *variability*, *v_p*, of the QoSParam. We discuss how we estimate a value for *v_p* later, but for now we see that in terms of our QoSSpace, we chose the mapping of the NetQoSState to be expressed as follows:

$$q = \{\langle P_1, p_1\_lo, p_1\_hi \rangle, \langle P_2, p_2\_lo, p_2\_hi \rangle, \langle P_3, p_3\_lo, p_3\_hi \rangle\}$$

where the *_lo* and *_hi* thresholds indicate the limits of our estimate of variability of the QoSParam. Note that this is a similar format as our expression for the QoSStates. However, the NetQoSState, must have both a *_lo* and *_hi* component for each QoSParam tuple.

4.3 State compatibility values

In terms of the QoSSpace, a description of the task for our model is relatively straightforward: we need to find when the region defined by the NetQoSState intersects with a region defined by a QoSState. If we can assign a meaningful value to this intersection, we can offer the application a **state compatibility value (SCV)**, a measure of the how well the current network QoS might support a particular QoSState. This compatibility value is a unitless number that is easy to use in other parts of the application.

Note that the definition of the QoSState and that for the NetQoSState suggest that we may be able to treat these state definitions as hyper-volumes. For example, we may chose to use the ratio:

$$\frac{\text{volume of overlap of NetQoSState with QoSState}}{\text{volume of NetQoSState}}$$

to evaluate a SCV for each QoSState. However, we choose not to do this. In definitions of QoSStates, our model allows use of different numbers of QoSParams in defining QoSStates for the same flow, resulting in different shaped volumes. In the evaluation of a volume, relative scaling by multiplication of values of N QoSParams may lead to a distortion when some values are particularly high or particularly low. Indeed, we need to process each QoSParam individually, and then offer some sensible summary to the application. So, we must first consider how we process individual QoSParam values.

For each QoSParam tuple, we can derive a **parameter compatibility value (PCV)**, that expresses the amount by which the value of a certain QoSParam from the NetQoSState falls within the operating region given by a particular tuple for a given QoSState. So, for a given tuple, T_p, from a QoSState, and the corresponding tuple, T_q, from the NetQoSState, for the same type of QoSParam, P:

$$\begin{aligned} T_p &= \langle P, p_p\_lo, p_p\_hi \rangle \\ T_q &= \langle P, p_q\_lo, p_q\_hi \rangle \\ PCV &= \mathrm{PCVF}(T_q, T_p) \end{aligned}$$

where **PCVF** is the **parameter compatibility value function**. The operation of this function is to assess the following statement:

$$\mathrm{PCVF}(T_q, T_p) = T_q \ \mathrm{WITHIN}\, T_p$$

where WITHIN is an operator that evaluates to a single number in the range [0, 1], for the ratio:

$$\frac{\text{intersection of length of } T_q \text{ and length of } T_p}{\text{length of } T_q}$$

The description of WITHIN is explained with the help of Figure 3. This shows the possible scenarios for evaluating WITHIN when T_q and T_p overlap. I is the length of the intersection of T_q with T_p. It is clear that the omission of either p_p_lo or p_p_hi from T_p (form the QoSState) poses no problem.

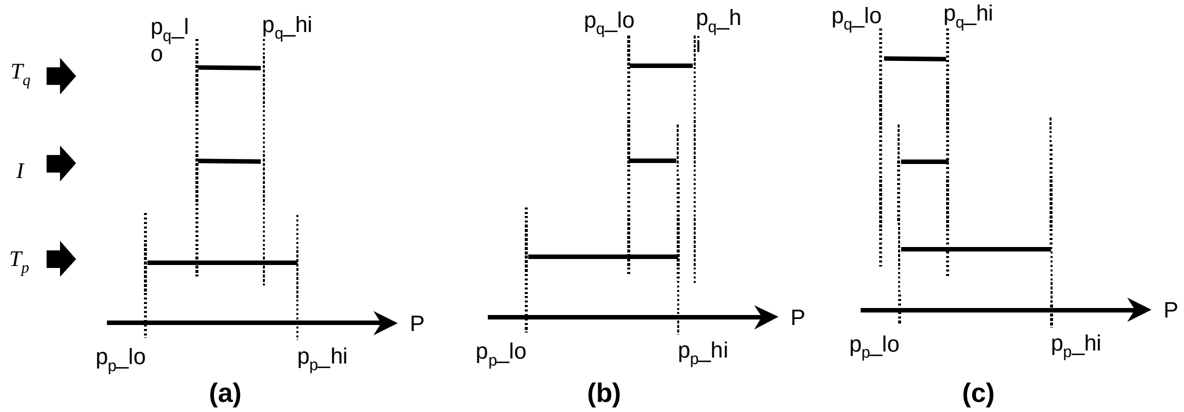

Figure 3: Scenarios for the evaluation of WITHIN

The PCVF has a simple algorithm:

$$\begin{aligned} L_q &= p_q\_hi - p_q\_lo \\ I &= \mathrm{MIN}(p_q\_hi, p_p\_hi) - \mathrm{MAX}(p_q\_lo, p_p\_lo) \\ PCV &= \mathrm{MAX}(0, I / L_q) \end{aligned}$$

The MIN and MAX functions in line 2 perform their usual operations, except that if either p_p_hi or p_p_lo are not defined, then p_q_hi or p_q_lo are used, respectively, as required. I takes the range [-∞, L_f]. When there is no intersection, I is negative, and we choose that PCV = 0, indicating "no compatibility" between QoSState and NetQoSState, while PCV=1 indicates "full compatibility". The final line of the PCVF ensures that PCV is in the range [0, 1]. This normalised value is a uniform, consistent and scaleable way of representing PCVs. The algorithm for the PCVF is also computationally simple. We see that as the variability of the QoSParam (i.e. the QoS fluctuation) increases, so L_q increases. Unless the fluctuations are completely contained within the region defined by *_lo* and *_hi* tuple thresholds (Figure 3(a)), as L_q increases, we have decreasing compatibility between the QoSState and NetQoSState with respect to that particular QoSParam (Figure 3(b) and (c)).

We choose a very simple measure for variability, v_p, to allow us to evaluate p_q_lo and p_q_hi: we chose v_p to be the difference between the current and previous QoSParam values:

$$\begin{aligned} v\_p &= \mathrm{ABS}(p_t - p_{t-1}) \\ p_q\_lo &= p_t - (v\_p / 2) \\ p_q\_hi &= p_t + (v\_p / 2) \end{aligned}$$

If we were to use p_t and p_{t-1} to produce an *instantaneous* estimate of the standard deviation, σ, of P, we would have:

$$\mu = \frac{p_t + p_{t-1}}{2}$$

$$\sigma^2 = \frac{(p_t - \mu)^2 + (p_{t-1} - \mu)^2}{2}$$

Some manipulation gives:

$$\sigma = \left| \frac{p_t - p_{t-1}}{2} \right|$$

So, our measure of variability, v_p, can be seen as a range of [-σ, +σ] (given by [p_q_lo, p_q_hi]) about our current value, p_t.

The **state compatibility value function (SCVF)**, must take the PCVs for all the QoSParams in the QoSState and transform them into a SCV. Any of the usual arithmetic summarisation functions that combine the values (such as a mean), will provide an incorrect PCV due to relative scaling. For example, consider five QoSParams, that give rise to the set of PCVs, $S_A = \{1.0, 1.0, 1.0, 1.0, 0.0\}$. The application may decide that it is likely to take a SCV of 0.8 to consider that flow-state is useable. This may seem reasonable, but we can see that the mean of S_A is 0.8, yet clearly one of the parameter conditions cannot be supported (hence the PCV of 0.0). The application would make an incorrect decision and this could lead to application and/or network instability.

Looking at it another way, we have seen, in (4.4) that the PCVF evaluates the function WITHIN. For N QoSParams and a QoSState with tuples T_n and NetQoSState with tuples T_{qn} ($\forall\ n = 1 \ldots N$), we base a SCVF algorithm on the following statement from:

$$\begin{array}{lllll} \text{if} & T_{q1} & \text{WITHIN} & T_1 & \textbf{and} \\ & T_{q2} & \text{WITHIN} & T_2 & \textbf{and} \ldots \\ & T_{qN} & \text{WITHIN} & T_N & \\ \text{then} & SCV & \text{is} & \text{HIGH} & \end{array}$$

This reasoning makes linguistic sense. We see that if the QoSParam values (the NetQoSState) *all* fall WITHIN the thresholds defined in the tuples for the corresponding QoSStates, then the degree to which the T_{qn} tuples are WITHIN their corresponding T_n tuples is the degree to which the SCV is HIGH. The key to this assessment is how to evaluate **and** and so generate a value for HIGH.

Fuzzy logic provides a suitable interpretation of **and** as the MIN function. So, we can modify our statement to say:

$$\begin{array}{ll} \text{SCV} = & \text{PCVF}(T_{q1}, T_1) \quad \textbf{and}_\mathbf{F} \\ & \text{PCVF}(T_{q2}, T_2) \quad \textbf{and}_\mathbf{F} \ldots \\ & \text{PCVF}(T_{qN}, T_N) \end{array}$$

where $\textbf{and}_\mathbf{F}$ is the fuzzy AND operator (MIN). We see from our example for the set of SCVs, S_A, (above) that we would now have the correct behaviour. The use of $\textbf{and}_\mathbf{F}$ (MIN) means that the SCV is in the range [0, 1], and is also computationally simple to evaluate.

Where a QoSState does not have a tuple defined for a particular QoSParam, then for the purposes of the SCVF, this is ignored. (Equivalently, the PCVF can evaluate a PCV = 1.0 for that QoSParam. This is also correct because it means that the QoSState is not dependent on that particular QoSParam, so it always has maximum compatibility with respect to that QoSParam.)

The effect of the using the PCV instead of just simple thresholding (to generate Boolean true/false indications of whether p_p is within p_lo and/or p_hi boundaries) is shown in Figure 4. We define three simple QoSStates, each with a single tuple:

$$qs3 = \{\langle P, 10, -, -, - \rangle\}$$
$$qs2 = \{\langle P, 6, -, -, - \rangle\}$$
$$qs1 = \{\langle P, 4, -, -, - \rangle\}$$

marked as horizontal lines at $P = 10$, $P = 6$ and $P = 4$, respectively, in Figure 4 (a). The area of interest is between times $t = 3$ and $t = 5$. Here we see a sudden drop in P, from a point where it can support all three QoSStates to a point where it can only support *qs1*. In the Boolean thresholds shown in Figure 4 (b), we see that we have high confidence for *qs2* at $t = 4$ even though we know that the variability of P is high at that point. In contrast, at $t = 4$, the PCV for *qs2* is not high, as shown in Figure 4(c). At $t = 5$, both the Boolean threshold and the PCV converge, but we see that the use of the PCV may help the application to avoid state-flapping (unnecessarily going into a transient state).

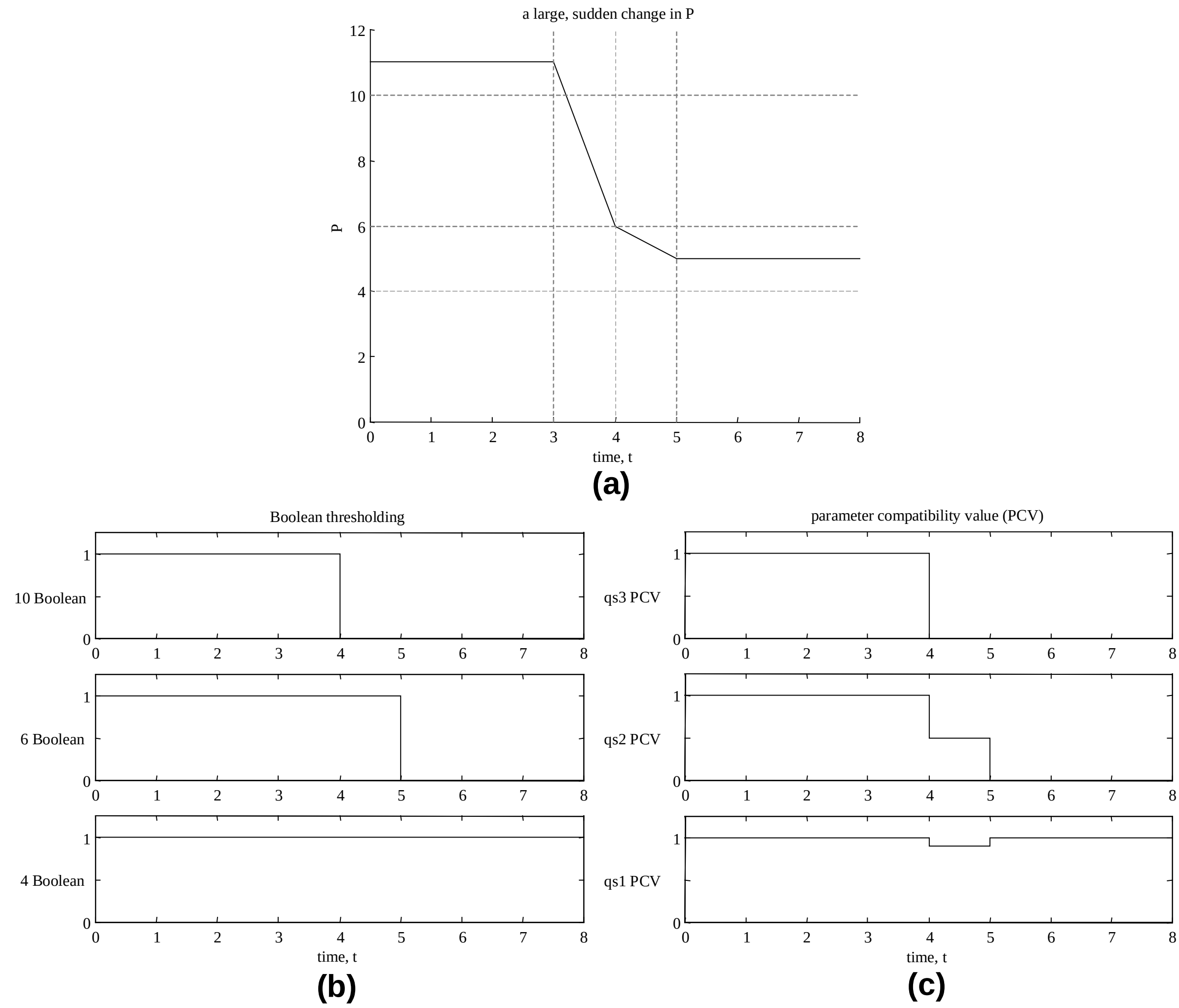

Figure 4: Comparison of Boolean thresholding and PCVs

4.4 QoSiStates

Although our model can, detect large, sudden changes close to QoSState boundaries, it can not spot slow gradual changes until they cross a boundary (this behaviour arises from our chosen definition of *v_p*). The application may like to have an indication when a QoSParam value in the NetQoSState is nearing a corresponding QoSState tuple *_lo* or *_hi* threshold. This would mean that the application is operating near a QoSState boundary (and so may soon need to change state). We can detect this through the use of **QoS intermediate states** or **QoSiStates**. These are quasi-states that do not represent a flow state as a QoSState does, but are an indication of the proximity of the QoSParam value to the *_lo* or *_hi* threshold of a QoSState. The QoSiState only exists within a QoSState, and is an optional part of the QoSState definition. A QoSiState is also specified by use of a boundary, *_qlo* or *_qhi*. The relationship of the QoSiState to the QoSState with respect to a single QoSParam, *P*, is depicted in Figure 5. It is clear that a QoSiState can only exist if it has a corresponding *_lo* or *_hi* defined in the QoSState.

There are parameter compatibility values, PCV_hi and PCV_lo, associated with the *_hi* and *_lo*, QoSiStates, respectively, for each QoSState tuple. These are evaluated in exactly same way as a PCV for the QoSParam but using the QoSiState tuple, $T_q = \langle P, p\_qhi, p\_hi \rangle$ or $T_q = \langle P, p\_lo, p\_qlo \rangle$ in place of the QoSState tuple, $T_p = \langle P, p\_lo, p\_hi \rangle$ as required. The *_qhi* and *_qlo* boundaries are specified by the application, and can be left undefined.

The QoSiStates also have a state compatibility value, SCV_I, which is evaluated in a different manner to that of the SCV for the QoSState. Consider a QoSState defined using *N* QoSParams. If *any* of the *N* QoSParams tuples, T_{qn}, in the NetQoSState suggest that the corresponding QoSParam value might be WITHIN a QoSiState tuple, T_{qqn}, then we know that the whole QoSState is operating close to one of its boundaries. The application may be cautious and conclude form this that the state may soon not be supported, but this is an application-level decision. So, our reasoning for evaluating the SCV_I is:

$$\begin{array}{lll} \text{if} & T_{q1} \ \text{ WITHIN } \ T_{qq1} & \textbf{or} \\ & T_{q2} \ \text{ WITHIN } \ T_{qq2} & \textbf{or} \ldots \\ & T_{qN} \ \text{ WITHIN } \ T_{qqN} & \\ \text{then} & SCV\_I \ \text{ is } \ \text{HIGH} & \end{array}$$

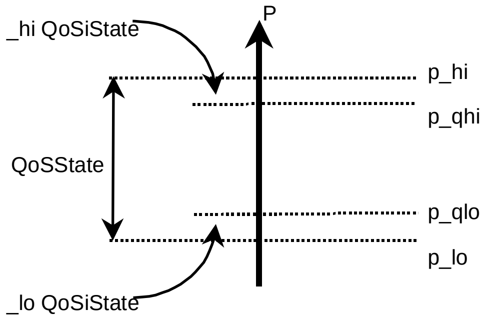

Figure 5: QoSiStates

In similarity with the evaluation of the SCV, we see that this makes (linguistic) sense. We need an interpretation for **or** that lets us assign a value to SCV_I. Again, fuzzy logic offers us an interpretation of **or** as the MAX function. So, we have:

$$\begin{array}{ll} \text{SCV\_I} = & \text{PCVF}(T_{q1}, T_{qq1}) \ \ \textbf{or}_\textbf{F} \\ & \text{PCVF}(T_{q2}, T_{qq2}) \ \ \textbf{or}_\textbf{F} \ldots \\ & \text{PCVF}(T_{qN}, T_{qqN}) \end{array}$$

where $\textbf{or}_\textbf{F}$ is the fuzzy OR operator. The $\textbf{or}_\textbf{F}$ operator (MAX) makes sense: it only needs one QoSParam to enter a QoSiState to indicate that the QoSState is operating near a boundary.

In Figure 6 we demonstrate the use of QoSiStates. We use a *_lo* QoSiState for *qs3* with *p_qlo* = 10.5, marked as a horizontal line at $P = 10.5$ in Figure 6(a). We can see in the lower graph of Figure 6(b) how the value of PCV_lo shows that *qs3* is in the region of the QoSiState, “forewarning” of a possible state change.

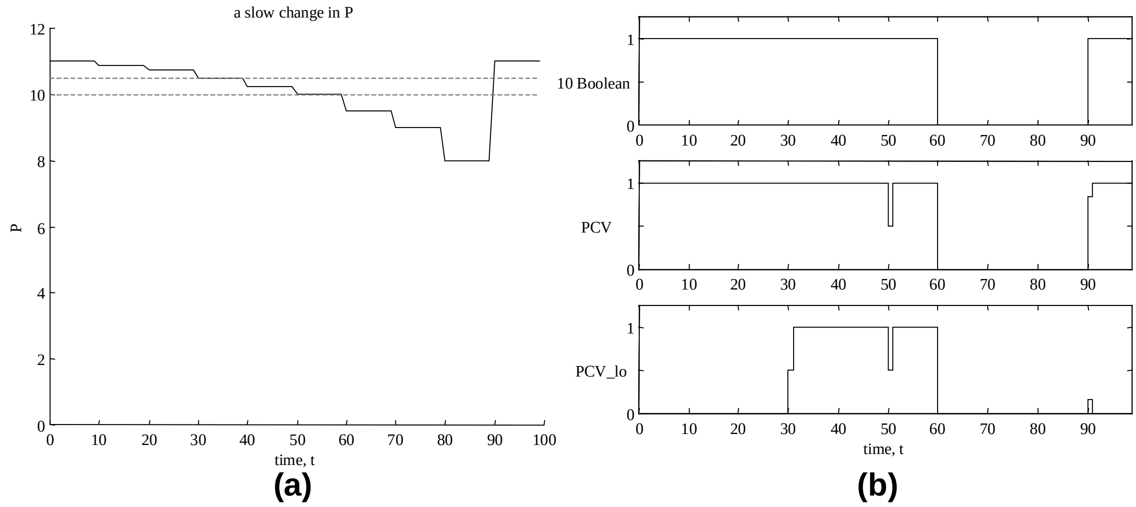

Figure 6: Using QoSiStates to detect operation close to a QoSState boundary

Notice the spike at $t = 50$ in the PCV and PCV_lo graph of Figure 6(b). This may seem to be “incorrect” behaviour as we can see that value of P at that time is within the operating region for *qs1*. However, we only know it is “correct” because we can see what happens at $t = 51$. At $t = 50$, all that we can see is that there is a downwards change in the value of P very close to its boundary. We do not know at $t = 50$ what will happen at $t = 51$ so our model assumes that the variability in the value of P will remain high. There is no reason why we can not assume that the value at $t = 51$ will be the *same* as at $t = 50$, but we choose to side with inertia(!) and offer the counter example in Section 4.3 and Figure 4. We will see how the spike at t = 50 can be removed by the application (if required) in Section 5.

5. Dynamic adaptation

In this section, we describe how our model is used to enable dynamically adaptable applications. To demonstrate the use of the QoSSpace, we describe a fictitious audio tool that can adapt its audio flow data rate in response to information about data rate availability for that flow. It does this by changing the audio encoding it uses. We will refer to our example audio tool as ***daat*** **(dynamically adaptable audio tool)**. *daat* is modelled on information presented in [BV96] for an audio tool developed at UCL [HSK98]. [HSHW95] shows that

encoding name	data rate of flow [Kb/s]	relative CPU cost	power cost
PCM	64.0	1	0.00001
ADM6	48.0	13	0.00014
ADM4	32.0	11	0.00011
ADM2	16.0	9	0.00009
GSM	13.0	1200	0.01250
LPC	4.8	110	0.00114

Table 1: Flow-state information for daat

mixing audio encodings in an audio flow provides usable quality audio streams for Internet wide use. *daat* is capable of the voice encoding schemes shown in Table 1, where the first three columns are taken from [BV96]. The fourth column is generated from the third column by dividing by 96000, and presents (artificial) power consumption per time unit, for each encoding scheme.

We will assume that an instance of *daat* is running on a mobile host and consider this *daat* instance in isolation. We define two QoSParams, R, the data rate available to the flow in bits/s, and B, the battery power available on the host. B takes values in the range [0, 1], 1 indicating that the battery on the mobile hosts is fully charged or that the mobile host is operating on mains power, and 0 indicating that there is no power. We also use the third column of the table to generate some *_lo* thresholds for *B* by dividing by 2400. We generate *_qlo* thresholds for R by adding 10% to the corresponding *_lo* threshold value. We choose to produce the following QoSStates for *daat*[3]:

$\langle pcm, \langle R, 64.0, –, –, – \rangle\rangle$

$\langle adm6, \langle R, 48.0, –, 52.8, – \rangle, \langle B, 0.0054, –, –, – \rangle\rangle$

$\langle adm4, \langle R, 32.0, –, 35.2, – \rangle, \langle B, 0.0046, –, –, – \rangle\rangle$

$\langle adm2, \langle R, 16.0, –, 17.6, – \rangle, \langle B, 0.0038, –, –, – \rangle\rangle$

$\langle gsm, \langle R, 13.0, –, 14.3, – \rangle, \langle B, 0.5, –, –, – \rangle\rangle$

$\langle lpc, \langle R, 4.8, –, –, – \rangle\rangle$

We will consider three scenarios for *daat*:

1. audio rate adaptation
2. audio rate adaptation and power conservation
3. helping to prevent state-flapping

[BV1996] presents a combined error and rate control mechanism, and we chose only to demonstrate a rate control feature. If we use an additional QoSParam based on packet loss figures we could produce a suitable demonstration of combined error and rate control. We chose to demonstrate only the rate control to simplify our presentation so that the dynamics of our model can be observed.

We first introduce the application adaptation function (AAF) for *daat*. The AAF is based around a threshold value to apply to SCVs, called ***q_compatibility***, and a stability time that governs the rate of state changes to a

[3] We choose to show that the QoSSpace can cope with heterogeneous definitions of QoSStates.

"better quality" state, called ***q_time***. We set *q_compatibility* = 0.8 and *q_time* = 60s. These measures can be interpreted as saying "do not change to a better state unless you have 80% compatibility over a 2 minute period for a better state". Here, "better" is defined by numbering the QoSStates with the values 1 - 6, with 6 representing the QoSState with the highest data rate requirement (*pcm*) and 1 representing the QoSState with the lowest data rate requirement (*lpc*). We then use the definitions and the AAF shown in pseudo-code in Figure 7.

5.1 Audio rate adaptation for daat

Here we use measurements captured from the Internet using ICMP ECHO [ICMPv4] probes[4] that have been processed by the QoSEngine to produce values for R. For this scenario, we ignore the QoS tuples defined in terms of B as we assume that the mobile host is operating on mains power. (For these data sets, the values of R are such that the Q_ISCORE has no significance on the flow-state change, so we ignore this, too.)

We use two data sets, each with probes at 1 second intervals, between the following hosts:

- darhu – theakston: darhu is a Sun SPARCClassic running SunOS4.1.3 at UCL and theakston is a WinNT4.0 workstation connected to UCL via a single BR-ISDN (Basic Rate ISDN) channel at 64Kb/s.
- waffle – tmnserver: waffle is a Sun4 running SunOS4.1.3 at UCL and tmnserver.misa.ch is an IBM server running AIX at IBM Laboratories, Zurich.

In Figure 8 and Figure 9:

(a) shows the SCV values for the QoSStates
(b) shows the QoSState selected by the AAF
(c) shows the value of R and also the data rate of the audio flow using the AAF

We see how the audio flow rate is fairly well matched to the available data rate (i.e. the white-space under the line in Figure 8(a) and Figure 9(a)), even though R fluctuates considerably. Increasing the value of *q_time* acts to smooth the data rate fluctuations as the Q_SCORE is effectively a mean of the thresholded SCV values, as shown in Figure 10. This also helps to smooth residual error from the QoS parameter measuremnets that manages to pass through the PCVF. However, increasing *q_time* may make the application sluggish in response to QoS fluctuations. As the value of *q_compatibility* approaches 1.0, this makes a good Q_SCORE harder to achieve. This will make it harder for the application to move into (and stay in) better quality states, as shown in Figure 11. We have found that *q_compatibility* = 0.8 appears to provide a reasonable threshold, but the exact value chosen may depend on the flow, e.g. video may require a higher *q_compatibility* value than audio. The value of *q_time* could be application specific, user defined, or its value could also be dynamically adaptable (an issue for further study).

5.2 Audio rate adaptation and power conservation for daat

We now use artificial values of R and B to demonstrate a (naïve) power conservation policy on the mobile host using *daat*. Here we consider that the mobile host moves from the user's home to the user's place of work. A battery powers the mobile host until reaching work, when mains operation is possible. During battery operation, we wish to conserve power. We use a naïve scheme in which the *_lo* thresholds for each QoSState in the tuples of B mark the lowest battery power charge that is allowed before that audio encoding can be used[5]. The simulation of this scenario is shown in Figure 12. (For clarity, in the values of R and B, we have not simulated any noise.) At home our user connects via ISDN (64Kb/s, time < 20), then by GSM (13Kb/s, 20 ≤ time < 120) on the way to work, and finally by Ethernet at work. The user starts with a fully charged battery (time = 0) and is not connected to the mains power until time > 200. We see in Figure 12(a) and Figure 12(c) how the *daat* switches from the GSM encoding to the LPC encoding (even though the GSM rate is still achievable) when the battery power, shown in Figure 12(b), goes down to 0.5 (time = 80). We have used *q_time* = 1 and *q_compatibility* = 0.8.

---

[4] This produces particularly noisy measurements to demonstrate the robustness of our QoSEngine-QoSSpace system. The values of R that we use were created by processing the raw measurements with our QoSEngine.

[5] A better scheme might be for the application to perform a calibration when it is first started, by using each encoding scheme to encode a buffer of data. This would measure the rate at which each encoding drains the battery power and then set these values as *_lo* thresholds for a QoSParam that is the *rate of change* in B rather than the absolute value of B itself. We chose our naïve approach in order to demonstrate better the dynamics of the AAF.

5.3 Helping to prevent state-flapping

We now use new artificial values of R in order to show how the QoSiStates are used to counter a possible state-flapping scenario. First we generate a scenario where there is state-flapping, as shown Figure 13 (*q_time* = 1, *q_compatibility* = 0.8). Here, QoSiStates are not used. We see in Figure 13(b) and Figure 13(c) how the value of R wavers around the *adm6 _lo* boundary causing the *daat* to see-saw between *adm4* and *adm6*. (Again, for clarity we have chosen not to simulate noise for R.). In Figure 14, we see how the use of QoSiStates removes state-flapping) Figure 13(a) shows the values of the SCVs, which are identical in both cases while, while Figure 14(a) shows the additional *_lo* QoSiState SCV_I values used only in the scenario of Figure 14.

6. Summary and conclusions

We have shown how it is possible to allow applications to make adaptation decisions automatically and dynamically from measured QoS parameter values. We have considered:

- **the QoSSpace:** a model of the network that allows application flow-states and network QoS to interact. The QoSSpace can produce QoSReports that give an assessment of the relative compatibility of the network QoS and the application's flow-states. The QoSSpace is intended to for general-purpose use. The reports it produces contain **state compatibility values (SCVs)** which are scaleable, simple in nature and easy to understand and manipulate. **QoSStates** are descriptions of the application's flow requirements using **QoSParam** boundaries, providing a simple and adequate QoS mapping between application and network. The QoSSpace does not unduly constrain the design, construction or operation of the application. QoSParams can be used to model quantities other than network QoS parameters, such as batter power, host load, cost, etc.
- **that dynamic adaptation is possible:** we have used simulations to show how the SCVs form the QoSSpace can be used. A simple algorithm, the application adaptation function (AAF) for the *daat*, was used to incorporate user preferences and application requirements into an automatic adaptation policy that controls the operation of the *daat* application. The AAF uses a SCV threshold, *q_compatibility*, and a stability time, *q_time*, to produce an evaluation, the Q_SCORE/Q_ISCORE, of the suitability of a QoSState for use by the application.

The QoSSpace is designed to be easily integrated into applications. The QoSStates specify the requirements of a flow but the definition of a flow is not constrained and remains application specific.

The work on the QoSSpace is ongoing. We need to investigate the interactions at the application level further.

Acknowledgements

This work has been funded through the projects QEDANA (HP Laboratories Bristol, UK) and IBCoBN (ACTS).

Q_SCORE(s, n, M)	for QoSState s, where s = 1 … 6, is defined as: $$\frac{1}{M}\sum_{m=0}^{M-1}\text{BOOLEAN}(SCV^{s}{}_{n-m} \geq q\_compatibility)$$ BOOLEAN(X) returns 0 for X true and 1 for X false, and SCV^{s}_{n-m} is the *SCV* for state s at time (n-m)
Q_ISCORE(s, n, M)	same as Q_SCORE but with *SCV_I* in place of *SCV*
n	the n^{th} time unit
$scv(s, n)$	SCV for state s at time n
$sdi(n)$	the SDI (state decision information) at time n is a value of s identifying the state that the application is in at that time
q_n_time	the value INTEGER(*q_time*/t_i), where t_i is the interval at which measurements are taken [seconds]
q_epoch	the previous value of n at which there was a state change

```
if n < q_n_time                                 // just started ..

   q_score_hi = 0,     q_s = 0
   for s = 1 to S                               // higher states override lower ones
       q_s = Q_SCORE(s,n,n)
       if (q_s >= q_score_hi) & (q_s >= q_compatibility)    // use state with highest score
              sdi(n)     = s
              q_score_hi = q_s
              q_epoch    = n
       endif
   endfor

else                         // has been going for some time ..

   if sdi(n-1) > 0           // and was previously in a state
       q_scv = Q_SCORE(sdi(n-1),n,q_n_time)
       q_scv_i = Q_ISCORE(sdi(n-1),n,q_n_time)
   else
       q_scv   = 0
       q_scv_i = 0
   endif

   q_n = n - q_epoch
   if (q_n <= q_n_time) &                                       // if within epoch and ..
       (q _scv >= q_compatibility) & (q_scv_i < q_compatibility)    // state is still usable ..
       sdi(n) = sdi(n-1)                                        // stay in present state

   else                      // else check for state change
       for s = 1 to S        // higher states override lower ones
              q_scv = Q_SCORE(s,n,q_n_time)
              q_scv_i = Q_ISCORE(s,n,q_n_time)
              if (q_scv >= q_compatibility) &       // Q_SCORE OK for this state and ..
              (q_scv_i < q_compatibility)           // Q_ISCORE OK for this state so ..
                     sdi(n) = s                     // use this state
              endif
       endfor

       if sdi(n) <> sdi(n-1)                        // state changed so mark new epoch
              q_epoch = n
       endif

   endif
endif
```

Figure 7: AAF for *daat*: definitions and pseudo-code

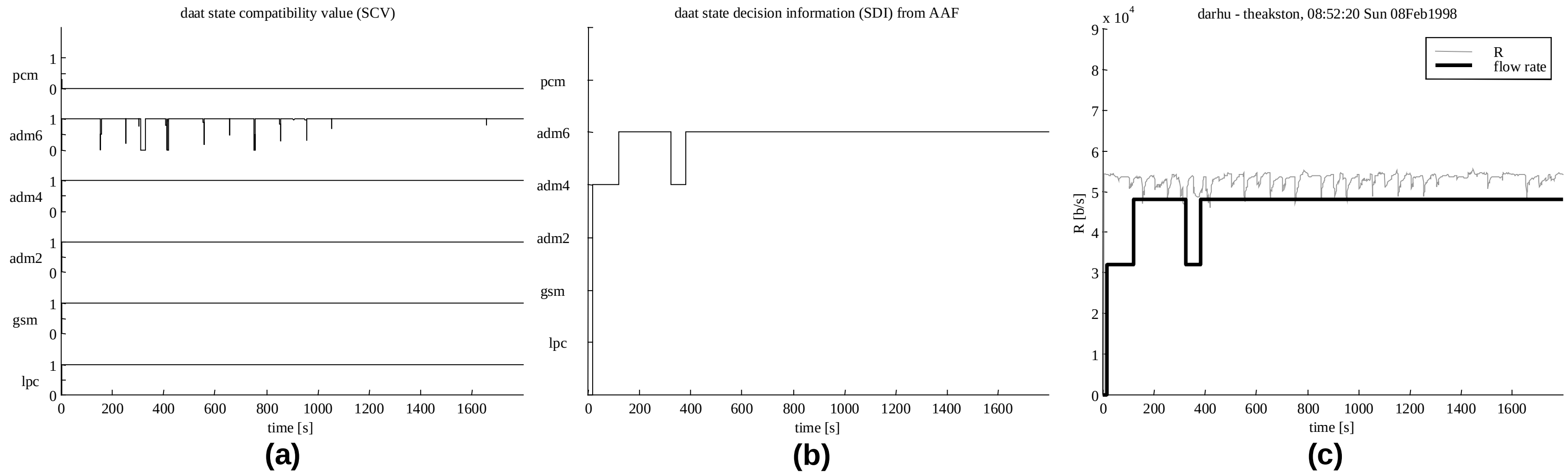

Figure 8: *daat* with the darhu - theakston data (*q_compatibility* = 0.8, *q_time* = 60s)

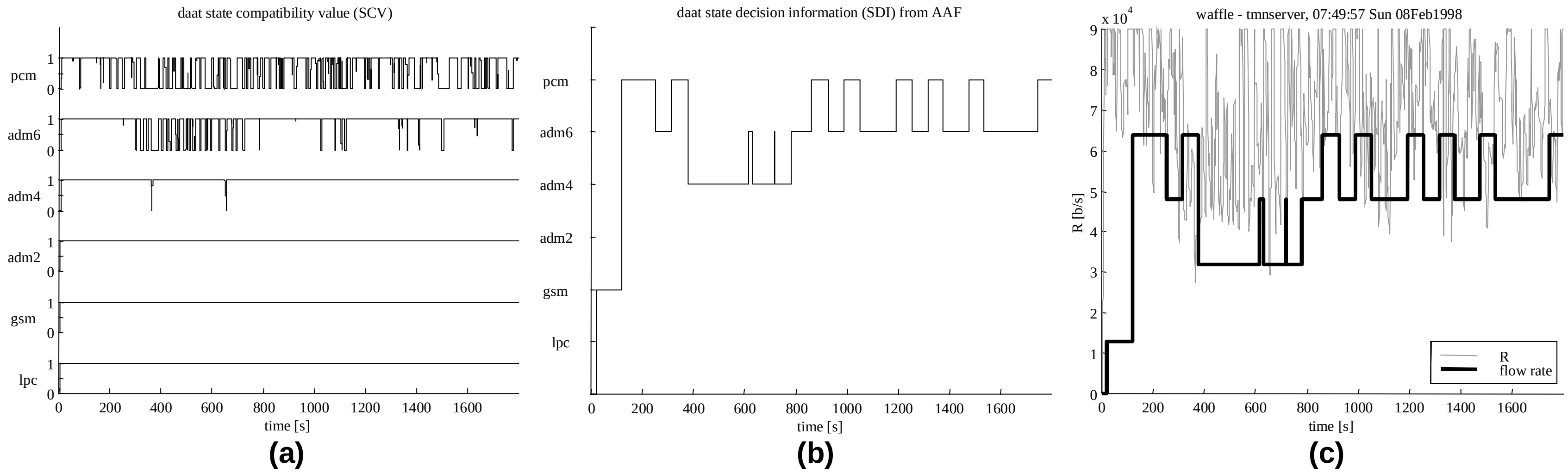

Figure 9: *daat* with the waffle - tmnserver data (*q_compatibility* = 0.8, *q_time* = 60s)

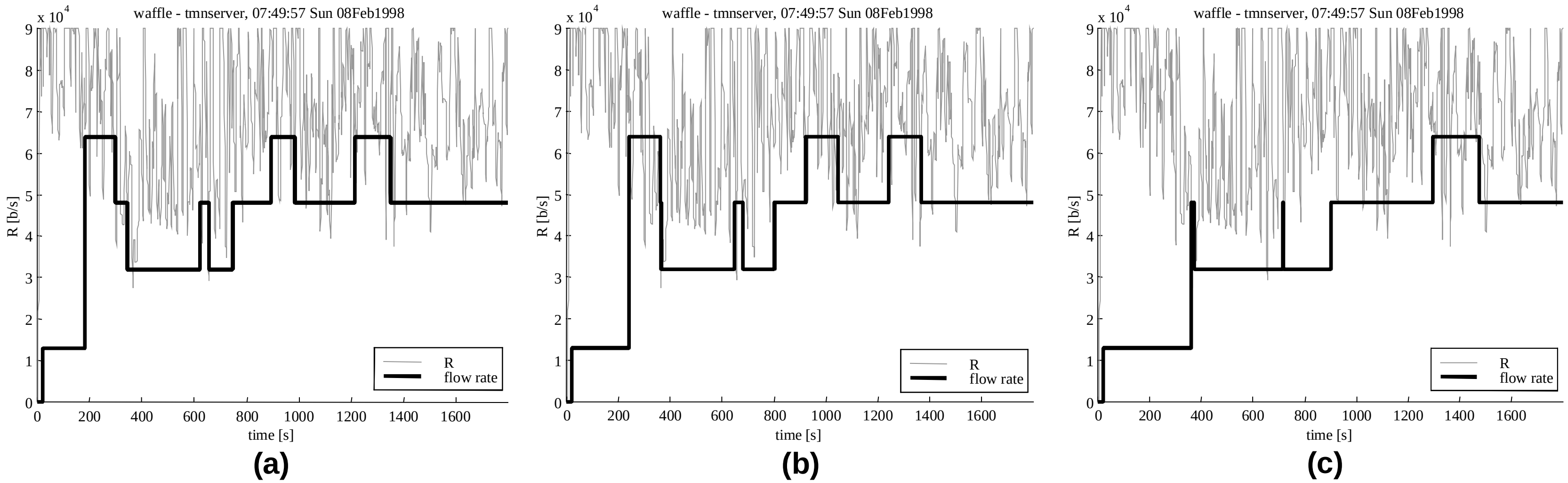

Figure 10: *daat* with waffe - tmnserver data; *q_compatibility* = 0.8 and (a) *q_time* = 90s (b) *q_time* = 120s (c) *q_time* = 180s

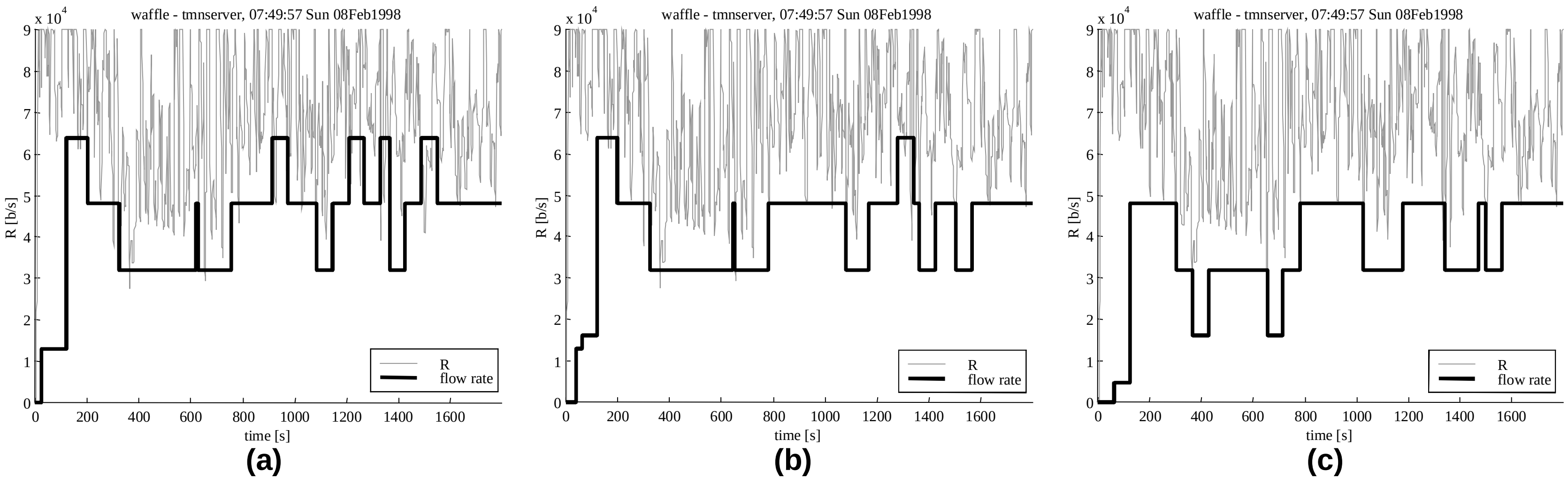

Figure 11: *daat* with waffe - tmnserver data; *q_time* =60 and (a) *q_compatibility* = 0.85 (b) *q_compatibility* = 0.90 (c) *q_compatibility* = 0.95

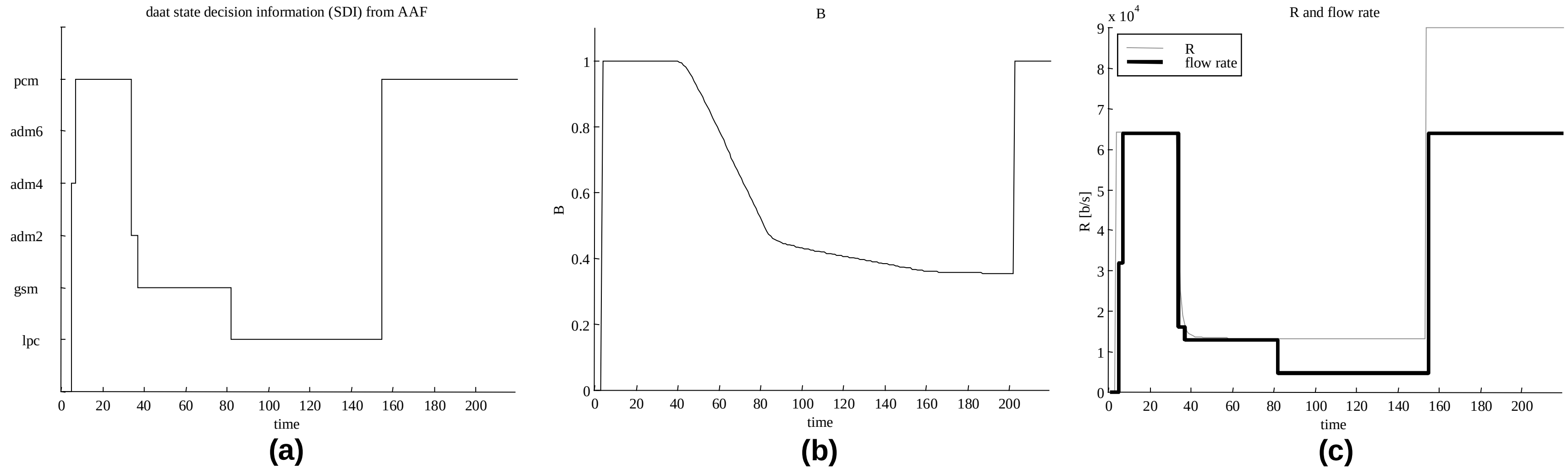

Figure 12: Artificial rate adaptation and power conservation scenario for daat

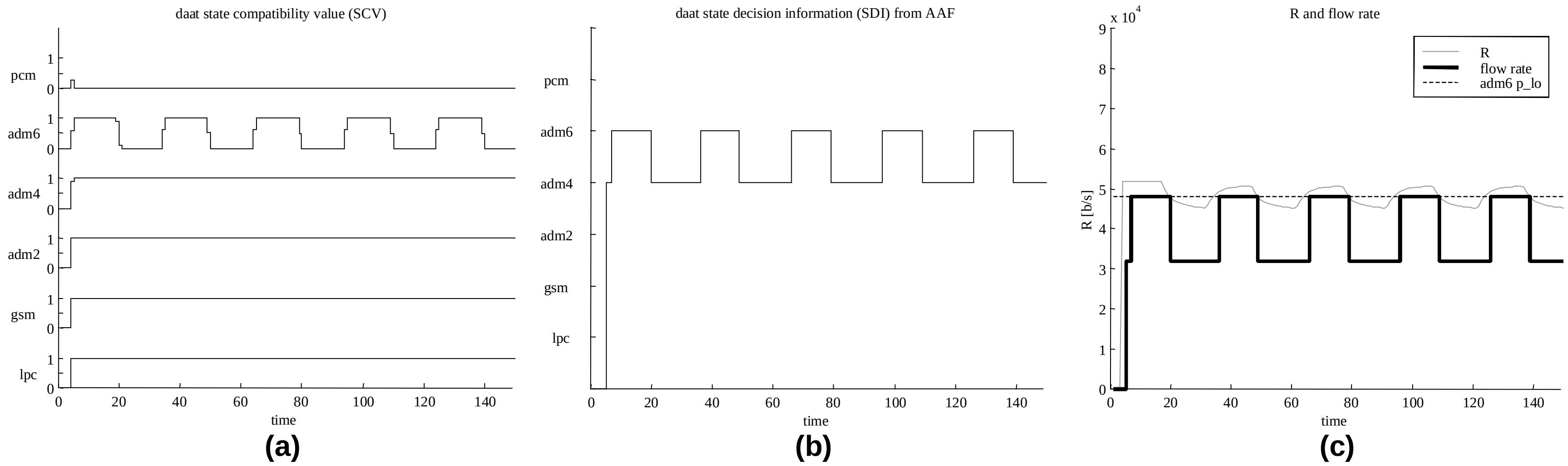

Figure 13: *daat* with state-flapping

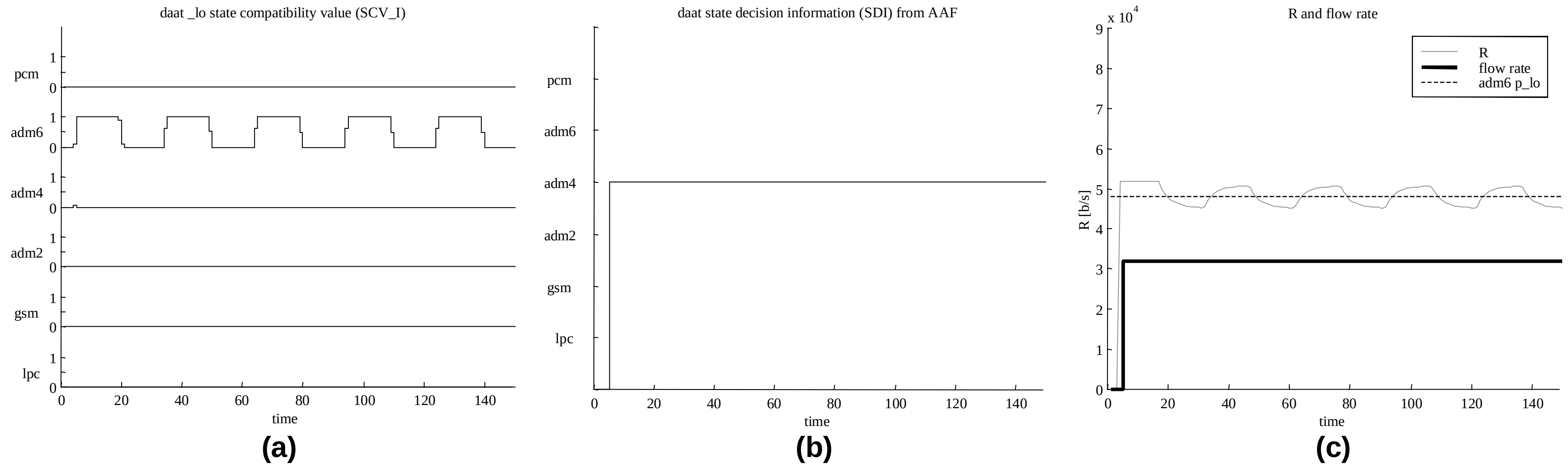

Figure 14: Using QoSiStates to combat state-flapping

7. References

[AMV96] E. Amir, S. McCanne, M. Vetterli, "A Layered DCT Coder for Internet Video", Proc. ICIP'96, Lausanne, Switzerland, Sep 1996.
http://http.cs.berkeley.edu/~elan/pubs/papers/ldct.ps

[BV96] J.-C. Bolot, A. Vega-Garcia, "Control Mechanisms for Packet Video in the Internet", Proc. IEEE INFOCOM'96, pp232-239, San Francisco, California, USA, Mar. 1996
http://www.cs.columbia.edu/~hgs/papers/Bolo9603_Control.ps.gz

[CAH96] A. Campbell, C. Auurecoechea, L. Hauw, "A Review of QoS Architectures", Proc. of 4th IFIP International Workshop on Quality of Service (IWQS'96), Paris, France, March, 1996.
ftp://ftp.ctr.columbia.edu/CTR-Research/comet/public/papers/96/CAM96a.ps.gz

[Cam97] A. T. Campbell, "Mobiware: QoS-Aware Middleware for Mobile Multimedia Networking", Proc. IFIP 7th International Conference on High Performance Networking, White Plains, New York, Apr 1997.
http://comet.ctr.columbia.edu/~mobiware/wireless/PUB/hpn97.ps.Z

[CCH94] A. Campbell, G. Coulson, D. Hutchison, "A Quality of Service Architecture", ACM Computer Communications Review, vol. 24, no. 2, 1994, pp6-27.
ftp://ftp.comp.lancs.ac.uk/pub/mpg/MPG-94-08.ps.Z

[CCH96] A. Campbell, G. Coluson, D. Hutchison, "Supporting Adaptive Flows in Quality of Service Architecture", ACM Multimedia Systems Journal, May 1996.
ftp://ftp.ctr.columbia.edu/CTR-Research/comet/public/papers/96/CAM96c.ps.gz

[D+93] L. Delgrossi, C. Halstrick, D. Hehmann, R. G. Herrtwich, O. Krone, J. Sandvoss, C. Vogt, "Media scaling for Audiovisual Communication with the Heidelberg Transport System", Proc. ACM Multimedia, pp99-104, Jun 1993.

[DIFFSERV] K. Nichols, S. Blake (Eds), "Differentiated Services Operational Model and Definitions", IETF DIFFSERV WG, work-in-progress, Feb 1998.

[FPM95] K. Fall, J. Pasquale, S. McCanne, "Workstation Video Playback Performance with Competitive Process Load", Proc. 5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video, Durham, New Hampshire, pp179-182, Apr 18-21, 1995.
http://hulk.bu.edu/nossdav95/papers/KevinFall.ps

[HSHW95] V. Hardman, M. A. Sasse, M. Handley & A. Watson, "Reliable Audio for Use over the Internet", Proc. INET'95, pp171-178, Hawaii, USA, 27-30 Jun 1995.
http://www.cs.ucl.ac.uk/staff/A.Sasse/inet95audio.ps

[HSK98] V. Hardman, A. Sasse, I. Kouvelas, "Successful Multi-party Audio Communication over the Internet", University College London, to appear in Communications of the ACM, May 1998.

[Kat94] R. H. Katz, "Adaptation and Mobility in Wireless Information Systems", IEEE Personal Communications, vol. 1 no. 1, 1994

[LLB97] S. Lu, K.-W. Lee, V. Bharghavan, "Adaptive Service in Mobile Computing Environments", in Building QoS into Distributed Systems, (A. Campbell, K. Nahrstedt, Eds), pp25-36, [Chapman & Hall] 1997

[LSD98] B. Landfeldt, A. Seneviratne, C. Diot, "User Services Assistant: An End-to-End Reactive QoS Architecture", Proc. 6th International Workshop on Quality of Service (IWQoS'98), Napa, California, USA, 18-20 May 1998.

[Pax97a] V. Paxson, "End-to-End Internet Packet Dynamics", Proc. ACM SIGCOMM'97, pp139-152, Sep 1997.
ftp://ftp.ee.lbl.gov/papers/vp-pkt-dyn-sigcomm97.ps.Z

[Pax97b] V. Paxson, "End-to-End Routing Behavior in the Internet" IEEE/ACM Transactions on Networking, vol. 5 no. 5, pp601-615, Oct. 1997
ftp://ftp.ee.lbl.gov/papers/vp-routing-TON.ps.Z

[RFC1890] H. Schulzrinne, "RTP Profile for Audio and Video Conferences with Minimal Control", RFC1890, Jan 1996.

[RSVP] R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, S. Jamin, "Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification", RFC2205, Sep 1997.

[SCEH96] S. Shenkar, D. Clark, D. Estrin, S. Herzog, "Pricing in Computer Network: Reshaping the Research Agenda", ACM Computer Communications Review, vol. 26. No. 2, pp19-43, Apr 1996
ftp://ftp.parc.xerox.com/pub/net-research/picn.ps

[SNMPv2] J. Case, K. McCloghrie, M. Rose, S. Waldbusser, "Introduction to Community-based SNMPv2", RFC1901, Jan 1996.

[WGS97] L. Wolf, C. Gridwodz, R. Steinmetz, "Multimedia Communication", Proceedings of the IEEE, vol. 85 no. 12, pp-1915-1933, Dec 1997.

[YGHS96] N. Yeadon, F. Garcia, D. Hutchison, D. Shepherd, "Filters: QoS Support Mechanisms for Multipeer Communications", IEEE Journal of Selected Areas in Communication, vol. 14, no. 7, pp1245-1262, Sep 1996.

[YL95] R. Yavatkar, K. Lakshman, "A CPU Scheduling Algorithm for Continuous Media Applications", Proc. 5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video, Durham, New Hampshire, pp223-226, Apr 18-21, 1995.
http://hulk.bu.edu/nossdav95/papers/yavatkar.ps