Design of an adaptive interface between video compression and transmission protocols for mobile communications

Arjen van der Schaaf, Koen Langendoen, and R. (Inald) L. Lagendijk

Delft University of Technology, The Netherlands
{arjen,koen}@ubicom.tudelft.nl

Abstract -- *In this paper we develop an adaptive interface between video compression and transmission protocols to handle QoS fluctuations that are common to mobile communication systems. We consider various generic design alternatives for QoS adaptation and identify 'QoS negotiation' as the most promising. This method gives the best possibilities to obtain system-wide efficiency. To handle the induced system complexity we apply a design philosophy (called ARC) that separates implementation dependencies by introducing abstract QoS interfaces between system modules. In the ARC philosophy the implementation details are hidden in the subsystems. To assure efficient adaptation, the QoS must be negotiated between modules. We select the abstract QoS parameters that are both necessary and sufficient for efficient negotiation between the video encoder and protocol modules. We describe the relation between the abstract QoS parameters at the interface and the internal parameters of common video coding methods and protocol elements. Furthermore, we describe a negotiation procedure that allows a system-wide optimum to emerge.*

Introduction

Usually mobile systems operate in highly variable context. The two main factors in context variability are the user and the mobile channel. The characteristics of mobile transmission strongly depend on the user's location and its environment. This is reflected in variable throughput, reliability, and required transmission power. The user context is variable because mobile systems are often designed to support multiple interactive services, which impose different workloads on the system in different situations. Handling the variable context is not the only requirement for mobile systems. They must also be efficient, since resources (especially battery power) are scarce in mobile systems. Handling variable workloads efficiently under variable conditions demands for an adaptive system.

The mobile system that we consider in this paper supports multimedia streaming over a wireless link. It consists of various adaptive modules, including a video encoder and protocols (see Figure 1). The video encoder is driven by a workload obtained from an application module, and the protocols induce a workload on a radio module. The application module directly experiences the user context variability, while the radio module is subjected to the fluctuations of the mobile channel.

In Figure 1 it is apparent that the video encoder and the protocol modules are not directly subjected to any context variability at either side of the system. However, if we optimize the system under global efficiency constraints, then the video coding and protocol module will indirectly experience context changes from neighboring modules, since these will have to adapt. Therefore, all modules will show context dependencies in their optimized behavior. Since all modules are context dependent, their operation

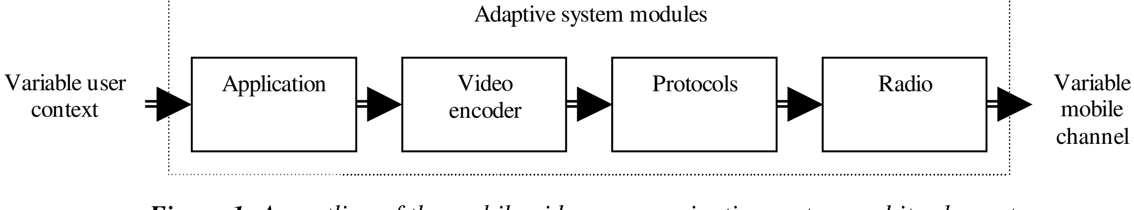

Figure 1: *An outline of the mobile video communication system and its elements.*

capabilities must be flexible and adapt to the occurring situations.

Each module in an adaptive system usually contains many parameters that influence its behavior. System-wide optimization of the adaptation parameters is difficult for two reasons. First, optimizing many parameters together is more difficult than optimizing small groups of parameters. Second, every single module is a complex system in itself, containing implementation details and specific knowledge. To keep the optimization procedure manageable, we must somehow separate the global optimization problem into several smaller problems. Ideally, we can perform independent optimization for each individual module. However, as we have argued, the optimization of one module depends on the context and implementation of other modules. So, the adaptation of individual modules cannot be optimized separately. Instead, context and implementation details must be exposed and shared, but in an appropriate format.

Related work

Optimizing dynamic context variability demands for a dynamic adaptive approach. The adaptive methods for compressed video over networks with variable characteristics that are found in literature can roughly be classified in four categories. In this section we present the general method of each category and compare their potential efficiency (see Fig. 2). Combinations of the presented methods are often found in literature, but we restrict this survey to methods with clearly distinct characteristics. Also, we do not consider joint source channel coding methods, which do not comply with the modular approach we aim for. In any of the approaches we see that a client imposes a workload on a server. The way in which the operation of both client and server is adapted differs.

First, we look at a forward scalable system approach [1] (see Fig. 2A). In this case the client does not adapt at all, but imposes a scalable workload that is adapted at the server side to the current state. The amount of adaptation specific information that must be shared between video coder and protocols can be very small. Examples of these systems appear as layered video encoders [2] with selective packet dropping at the protocol level. Even without priority information we can use multiple description video coding [3] with random packet dropping. More specific information is required when trans-coding is used. In forward scalable systems the generated workload must be suitable for every context, which requires usually more effort (power consumption, CPU load) than strictly required. This inefficiency makes the forward scalable method unsuitable for mobile devices with limited resources.

Figure 2B shows a backward adaptive method. In this case the protocol module adapts its operation to a

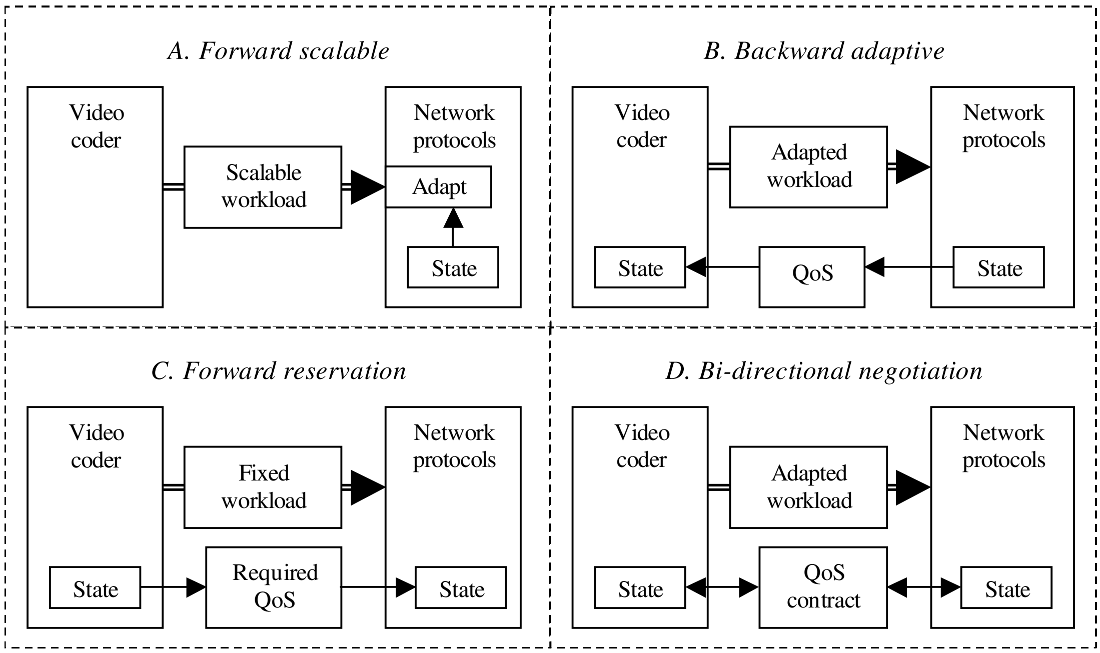

Figure 2: *Different approaches to adaptive interfaces.*

local context, and signals its performance as Quality of Service (QoS) information to the video encoder, which adapts the workload accordingly. Examples of this approach are found in general Constant Bit-Rate (CBR) video systems, in which case the feedback mechanism keeps the bit-rate constant. The QoS feedback method is also applied for rate control with variable channels (e.g. [4]). The attractive feature of backward adaptation compared to the scalable approach is that there is no overhead in generating the workload. A drawback is that the protocols module does not know anything about the context of the video encoder, so that it cannot adapt to context specific requirements like the required level of protection of the bit-stream.

The third method (Fig. 2C) is inspired by quality of service reservation systems, like ATM. The protocols reserve a fixed amount of resources for the video encoder. In this situation the transport protocols can adapt to the context specific needs of the video coding module, but the encoder cannot adapt to the context of the transport protocols. Also, the client must produce a fixed workload. The only way to accomplish mutual adaptation with forward reservation systems is to renew the reservation contract. In that case the client issues a contract with new requirements and the server can accept or refuse it. This is a very limited way for the two modules to come to a mutually optimized solution. The advantage of the contract approach is that both client and server can individually optimize their operation to the given context with their mutual dependence fixed in the contract. Individual optimization is simpler than mutual optimization, but usually less efficient.

Figure 2D is based on QoS architectures, as presented in [5]. It is a combination of backward adaptation (Fig. 2B) and forward reservation (Fig. 2C). In this case a contract is a result of bi-directional negotiation between the protocols and the video encoder, and therefore reflects both their contexts. If the negotiation is a continuous dynamic process, then the encoder can continuously adapt the workload to the changing contract (like Fig. 2B). On the other hand, if we temporarily fix the contract, then both sides can independently optimize their operation with respect to the fixed contract and their variable context (like Fig. 2C). This combined approach shares the efficiency advantages of scenarios 2B and 2C.

Efficient resource usage is the highest priority in mobile systems. Therefore, we adopt the bi-directional negotiation method to achieve an efficient mobile video communication system. Unfortunately, contract negotiation may induce serious overhead, since much of the individual context information must be shared among separate modules. To avoid the negotiation to become too complex, we must restrict the procedure to only the really necessary parameters. For the sake of manageability these parameters must be implementation independent, such that we can modify modules independently without changing the interface. In this paper we present a feasible design of a QoS negotiation interface. First we recapitulate a general framework for abstract QoS negotiation. Subsequently we focus our attention on adaptive QoS negotiation between video encoder and protocols.

QoS negotiation and ARC

The general QoS negotiation method that we adopt is called Adaptive Resource Contracts (ARC), and has been described in previous work [6]. The method is illustrated in Figure 3. Notice that each module acts both as a server to 'higher' layers and as a client to 'lower' layers. This is in conformance with the hierarchical setup shown in Figure 1. The hierarchal concatenation of modules is important, because subsequent modules determine each other's context. A QoS interface deals with the interaction of one module acting as server and one module acting as client. The contract that is to be negotiated in the interface holds a number of abstract QoS parameters. The process of QoS negotiation starts with the client issuing a request. The request is a partial specification of the expectations the client has about the performance of the server. The server (now knowing approximately what is expected) responds with an offer, stating possible performance options. This informs the client about the context dependent capabilities of the server. The client can respond to the offer with selecting an option and issuing it as a contract, or restarting the negotiation by formulating a new (modified) request. Once the contract is established, the client can put the appropriate workload on the server. The server, in turn, must inform the client on the status of the workload processing. This feedback of context dependent QoS information to the client is essential for fast local adaptation. If the QoS status of the server becomes unsatisfactory for the client (due to context changes), then the contract must be re-negotiated. This form of adaptation is slower, but involves more precise mutual tuning of QoS parameters, which improves efficiency.

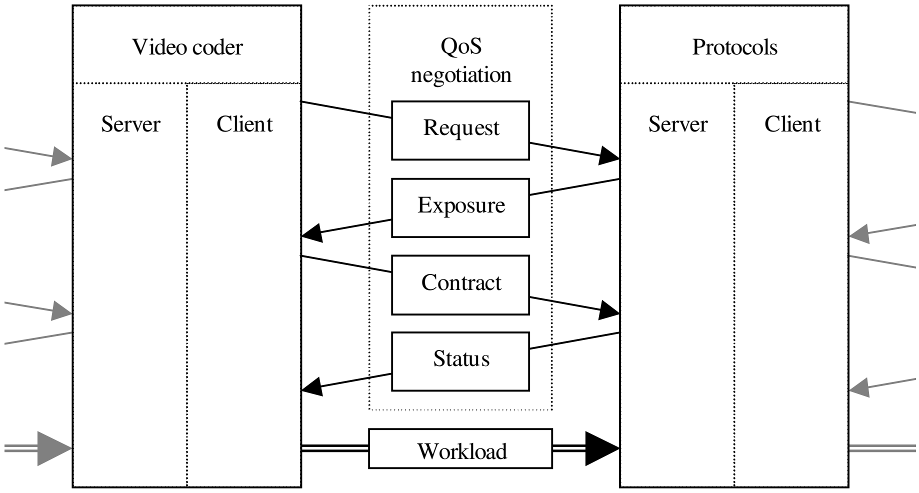

Figure 3: *ARC interface outline.*

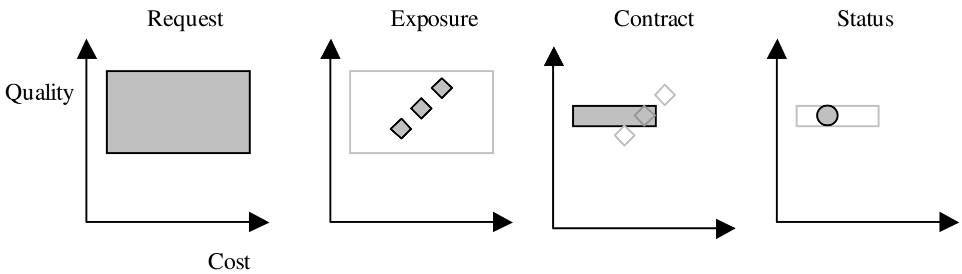

Figure 4: *ARC negotiation*

As an example of QoS negotiation consider Figure 4. The abstract QoS is here represented in a two-dimensional space. The two parameters are denoted by cost and quality. Contrary to how QoS is often used, an ARC interface reflects costs besides quality parameters. In our opinion QoS can only be used for system-wide optimization if we specify both costs and benefits. The request from the client in Figure 4 is a range selection from the QoS space. The server knows now what the client is interested in, and it responds with a number of detailed offers. All the offers are enclosed within the initial request range, but they give a tighter description of the QoS that the server can offer within the current context. Then the client can select a QoS range that includes at least one offer and set it as the contract. Choosing a range rather than a single point for a contract gives some room for adaptation within the contract boundaries. When the contract is established, the server tries with best effort to keep the actual QoS within the contract range. The actual QoS status is returned to the client as a single point in the QoS domain. Using an abstract QoS domain as common language between the client and the server effectively hides explicit implementation details from the negotiation procedure. The implementation details are reflected in the content of the contract, not in its form.

An adaptive video-protocol interface

In this section we concentrate on the design of an ARC-style QoS negotiation interface between a video encoding module and transport protocols. The video encoder module works as client and the protocols as server, as apparent from Figure 1. The workload consists of transporting a compressed video-stream. The video encoder, in turn, acts as server to the higher application layer, which provides the raw video data and context dependent quality requirements. The protocol module acts as a client towards the lower radio layer, from which it needs context dependent transport services. The resources in the system are limited. First, battery power is scarce and can be exhausted. Second, we envision that the system runs on a limited number of CPU's, such that computational power is also limited. Third, the transport capacity of the radio is limited. The interface between the protocols and video encoder must allow enough flexibility to optimize the global system operation with respect to the given context variability under the mentioned resource constraints.

Selecting the QoS-domain

The first step in specifying an interface is selecting the language. In our case that means selecting which parameters must be specified in the QoS contract. We determine the parameters with the following criteria in mind:

- The parameters must be necessary. Each parameter must describe a unique mutual dependency between the modules. Applying more parameters than necessary increases the complexity needlessly.
- The parameters must be sufficient. Enough of the context variability must be captured by the parameters, such that each module can adapt towards the global optimum. Applying fewer parameters than needed reduces the potential efficiency of the system.
- The parameters must be abstract. Each parameter may not depend explicitly on the implementation details of either module. Applying implementation dependent parameters reduces the manageability of the system.

In a previous theoretical study [7] we derived a number of abstract parameters. The theoretical setup involved optimization of source and channel coding for progressively encoded video under transport constraints. We considered variable source characteristics (strong and weak progressive character) and variable channel characteristics (throughput and reliability). For the protocols we considered various packetization and error protection options.

We found that constrained latency is the main factor that prevents source and channel coding to be optimized separately. Therefore, latency requirements must be communicated between modules in the QoS interface. Latency constraints usually originate from the application level, and depend much on the application context. Latency information must be propagated through all other layers, such that each successive module knows how much time it has to process the workload.

A QoS contract generally specifies at least the workload processing capabilities of the server. The workload that the video encoder puts on the protocols is a compressed video stream, which must be transported. The protocol module uses the lower layer radio module for the actual transport, but it has several internal options to adapt the quality delivered by the radio to the requirements from the video-coding module. Of course, increasing the quality will have negative consequences for other QoS parameters. In our theoretical assessment, we showed that the protocol capabilities are best expressed by the properties of the transported data-stream: net-throughput and net-bit-error probability. Our assessment showed that communicating more precise information through the QoS interface couldn't increase the efficiency of the system.

Our theoretical study involved only external restrictions on transport resources. In a practical setup we envision that the modules must be flexible enough to optimize internal resource usage as well. Here we are interested in computational and power resources. Therefore, we include two more parameters in our interface. The first is power dissipation. This parameter expresses the total power consumption of the transmission, including protocols and radio contributions. Communicating total power consumption, including all sub-systems, allows finding a global optimum rather than optimizing for local low-power. The second additional parameter is CPU share. This parameter specifies the fraction of CPU time that the protocols may consume for computation. With this parameter an explicit tradeoff between source coding complexity and channel coding complexity can be made. This parameter is only necessary if both the protocols and video coding modules run on a single CPU with limited computational capacity.

A final parameter that we include is radio share. This parameter specifies how much of the total radio capacity is used by the video stream. The purpose of this parameter is to allow the application module to distribute the radio capacity over multiple media streams.

Because we assume the context of our system to vary rapidly, and we must avoid renegotiating contracts too often, we must not fix the contract too tightly. Therefore, we specify contracts over a QoS range, rather than specifying a single point in the QoS-domain. Our experience showed that specifying either a maximum or minimum value for each QoS parameter suffices for this purpose (see Table 1).

To improve tradeoffs between QoS parameters within the contract range, we specify an order of significance of the parameters. This ordering reflects the priorities of the video-coding module. The protocol module uses the priority information to make efficient internal tradeoffs. The list of QoS parameters is shown in order of priority with a motivation and a description in Table 1.

Table 1: *The selected QoS interface parameters (in order of descending priority).*

Parameter	Description	Priority motivation
Maximum latency	The time required to send a single bit, from the moment that it is offered to the protocols until it is available at the receiver side.	Data that arrives after a deadline is useless.
Maximum bit-error rate	The net bit-error probability after FEC and ARQ (see next section).	Corrupted data is partly useless.
Maximum CPU usage	The allowed share of CPU time for protocols and transmission processes on the CPU where the video encoder is running.	Conflicts of processes on the CPU induce timing (latency) problems.
Maximum radio usage	The allowed share of radio capacity to be used by the data-stream.	Radio conflicts induce throughput problems.
Minimum throughput	The minimum net throughput without protocol and channel coding overhead.	The video encoder can gracefully trade bit-rate with quality.
Maximum power	The maximum allowed power consumption for the transmission, including protocols, FEC, radio, and other subsystems.	Power-consumption is a long time-scale issue.

Internal parameter adaptation

The QoS interface is an abstract interface. Its content must be translated into implementation-specific internal parameters of the client and server. In this section we show how our adaptive interface (see Table 1) relates to internal parameters of standard video coding methods and protocol elements. We also show how the adaptive video coding and protocol modules relate to the rest of the adaptive system.

Video coding side

Figure 5 shows the video encoding module with internal parameters and the defined QoS interface with the protocols. The internal parameters are implementation dependent. Also shown in Figure 5 is the interface with the application module. This is an abstract negotiated interface as well, but with different parameters. Its function is to communicate necessary context information between the application and the video encoder. This interface sets part of the context of the video encoder. Therefore, we discuss it briefly.

As shown in Figure 5, the application specifies a latency requirement for the video communication. This requirement depends on the application context: interactive video communication requires lower latency than viewing a prerecorded video sequence. Furthermore, the application context determines the required quality of a transmitted video, possibly including spatial, temporal and SNR measures. Power is included in the interface, such that the application can restrict the total power that is dissipated for transmitting the video stream. The allowed power dissipation may depend on the relative importance of video material, the remaining battery charge, and the length of the video sequence. Finally, the application may involve transmission of more than a single media stream. In this case the relative importance of the various media streams determines the fair share of computational and transmission resources for each stream.

When the values of the abstract parameters in both QoS interfaces are set, the internal parameters can be derived from implementation-specific optimization. In this paper we assume a very general hybrid DCT block encoding method, like the MPEG-2 or H.263 standards. The three main internal adaptation parameters for these kinds of coders are shown in Figure 5. These are: skipping blocks or complete frames,

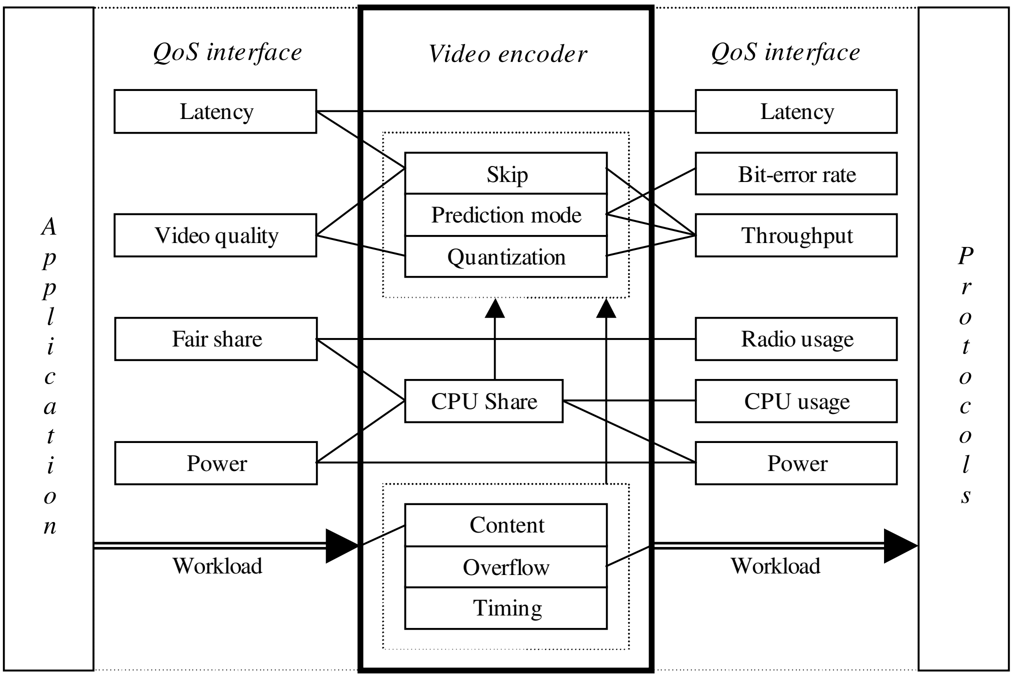

Figure 5: *video-coding parameters relation*

prediction mode, and quantization step-size. Depending on these three parameters, the encoding process will consume a specific amount of CPU capacity. The allowed CPU share for the video coding process is determined from both QoS interfaces and is restricted by power usage and CPU availability. Finally shown in Figure 5 is the internal context specification of the video-coding module, which includes a content description of the video, an overflow signal of the output stream, and timing information. Figure 5 gives an overview of the relations between the parameters; we discuss the underlying details below.

Skipping blocks or frames reduces the temporal resolution of the video, effectively reducing the bit-rate (throughput) of the compressed stream. Furthermore, skipping parts of the encoding process reduces the induced computational load. The quality reduction by skipping frames depends much on the motion content of the video. Skipping blocks or frames is an efficient method to solve problems with timing, or to reduce the throughput temporarily when the output buffer overflows. Hybrid video coders can issue either intra frames, without temporal prediction, or inter frames with various types of prediction. The prediction mode may include many parameters, like: number of motion vectors, motion vector search parameters, and references to multiple frames. In general, the more accurate a prediction is, the more it reduces the bit-rate (throughput), but also, the more CPU capacity it requires to calculate. Furthermore, whether predictions are effective in reducing the bit-rate largely depends on the motion content of the video sequence. A side effect of using prediction is that errors in the compressed bit-stream usually propagate through multiple frames of the reconstructed sequence. Therefore, a sequence of inter frames is less robust than a sequence of intra frames and requires to be transmitted with a lower bit-error rate [8]. The quantization step-size of DCT-based coders trades both spatial and SNR quality of the video sequence with bit-rate (throughput). The final bit-rate also depends on the content of the sequence. Fine-tuning the coding of a sequence with varying content characteristics, such that the produced bit-rate matches the throughput in the QoS contract, is usually done by continuously adapting the quantization step-size. The interaction of the interface parameters with the internal parameters of a video coder is complex and depends on implementation details of the encoder. The exact relation that determines the optimized adaptation behavior of the module can partially be derived theoretically, using algorithmic models, computation models, source models and channel models. However, experimental optimization and empirical feedback are needed to cope with the complex interactions and with implementation details that are out of the scope of the models.

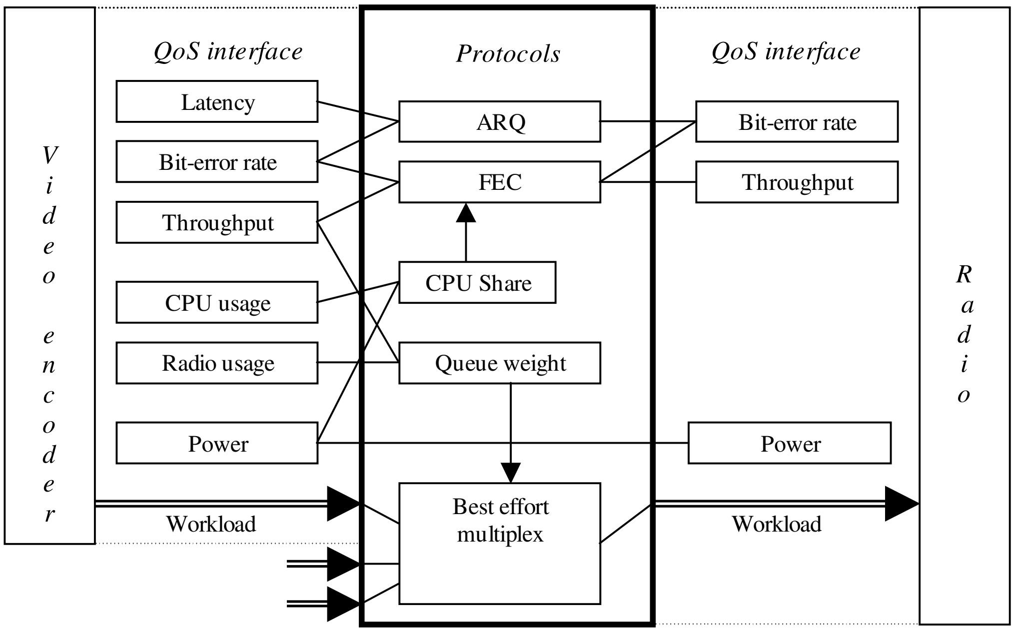

Figure 6: *protocol parameters relation*

Protocol side

The transmission protocols implement the other side of the abstract QoS interface. Figure 6 shows how the QoS parameters relate to the internal protocol parameters, which in turn relate to the QoS interface with the radio module. The transmission protocols use two mechanisms to enhance the reliability (i.e. bit-error rate) of the radio: forward error correction (FEC) and retransmissions (ARQ). The usage of FEC improves the bit-error-rate at the expense of throughput, because of the redundancy bits transmitted with the raw data. For critical data (e.g., frame-headers) ARQ can be used to reduce the bit-error-rate to zero at the expense of increasing latency.

The protocols share a CPU with the video and application. To ensure proper operation, the protocols must stay within the CPU usage specified at the QoS interface. Power is the other constraint that limits the possibilities of the protocols. Both constraints are combined into the internal CPU share parameter. CPU capacity is mainly required to calculate the FEC.

In general the transmission protocols service multiple streams, which are multiplexed over a single radio link. When the combined workload exceeds the capabilities of the radio, the protocols revert to best-effort mode. To ensure fairness amongst the multiple streams, the protocols use weight factors that are proportional to the relative importance of the streams. These weight factors are based on the requested throughputs and limited by the maximal radio usage per stream.

We have shown that the internal parameters of a common block based video coding system and the common protocol elements depend on the QoS interface parameters. Furthermore, we showed that all parameters of the QoS interface have a distinct function. This suggests that the proposed interface parameters are indeed both necessary and sufficient, as we required.

Negotiation procedure

Internal adaptation is required to meet the QoS contract settings in a variable context. This, however, is only one part of the ARC-design. Establishing the contracts through negotiation is the other part, which is discussed in this section.

The negotiation is initiated by issuing a request that specifies the expectations of the video coder towards the protocols. The initial expectations usually originate from a request that the video encoder received from the application to transmit a video sequence with a certain quality. The video coder translates these requirements from the application into a request for service to the protocols by considering its own

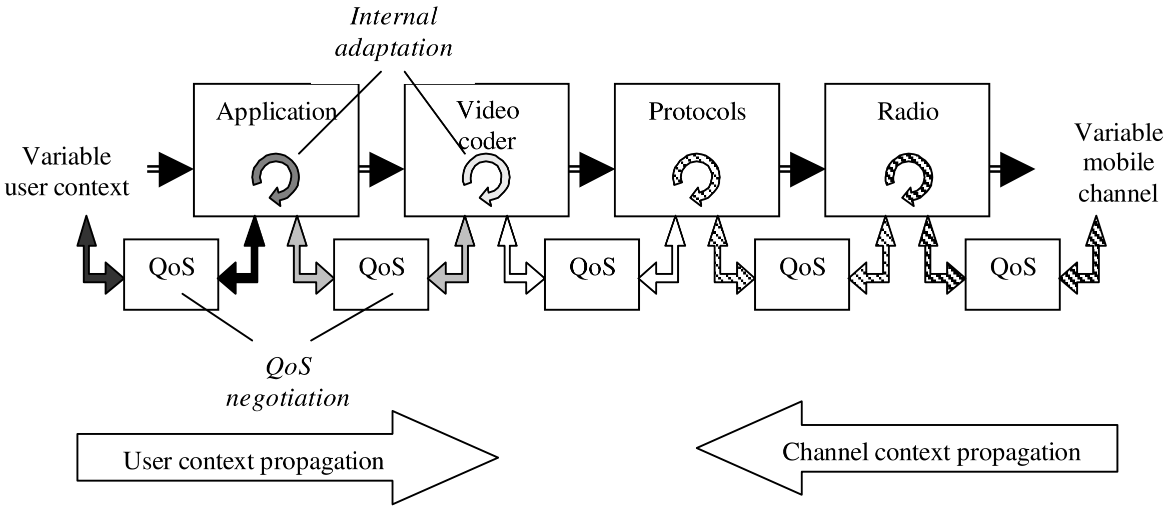

Figure 7: *Emerging behavior to context changes*

capabilities. The initial request does not need to be precise. The necessary precision depends on the priority of the QoS parameters and on the flexibility of the video-coding module: if the encoder is capable to adapt, then its requirements can be weak. The request must specify only minimum requirements. The protocols will offer better solutions if available, but not solutions that are worse than the request.

The protocols respond to the request with offering a number of options. For this, the protocols must first know how much transmission capacity can be acquired from the radio. Therefore, the protocols, issue a request to the radio layer. When the radio layer offers resources, then the protocols can offer various services to the video module, depending on the offered options of the radio and the capabilities of the protocol module. The protocols offer only options that are within the requested range. If no option satisfies the request, then nothing is offered, and the video coder must issue a new (weaker) request. If the number of options in the request range is excessively large, then the protocols must reduce their number, as to reduce the total complexity of the negotiation. The video encoder specifies the maximum number of offers it is willing to consider. The protocols filter out all options for which there exists an other option that is better with respect to all parameters. Also, options that differ too little from other options may be removed, effectively subsampling the distribution of offers in the QoS-space.

After receiving the offers from the protocols, the video-coding module in turn offers options to the application layer. Here the video coding matches its own capabilities with the offers it got from the protocols. The application layer selects an option and issues a contract to the video coding module, which in turn can then issue a contract to the protocols, et cetera.

When the contracts are set, the server modules feed back status information to their clients, which allows them to adapt quickly to context changes. It is important that the protocols feed back possibilities of the current status rather than past achievements. The video encoder needs to know how much throughput it can use in the future rather than how much it used in the past. This allows both upscaling and downscaling to QoS changes.

If the context changes so much that the QoS status goes out of the contract range, then a module can decide to renegotiate. In this case the whole cycle starts all over, except that now more context information is present in each module. Therefore, the negotiation can be more accurate, and does not always need to involve all modules in the hierarchical chain.

An overview of the negotiation is shown in Figure 7. In this figure two types of adaptation are shown; internal adaptation in each module and QoS negotiation between the modules. The latter method accomplishes QoS propagation through the hierarchical chain, which allows a system-wide optimum to emerge. A change in the user context propagates through the system from left to right and a change in the channel context propagates in the opposite direction. In each module internal adaptation solves part of the context change, such that the reach of context changes is limited. This guarantees stability of the QoS propagation.

Conclusions and future work

The capabilities of mobile systems are rapidly increasing to the level where (live) video connections can be supported. In contrast to wire-line video connections, advanced algorithms are required to handle the highly variable context (user demands and channel conditions) in which mobile systems operate. The system modules (e.g., video encoder and protocols) must adapt their mode of operation to use the limited resources most effectively. In particular, battery energy is so scarce that individual modules cannot be optimized separately; system-wide power efficiency can only be achieved by exchanging information across module (layer) boundaries. We argue that the approach of bi-directional QoS negotiations is most suited despite the extra complexity and costs over traditional adaptive approaches.

The adaptive interface between video compression and transmission protocols presented in this paper follows the generic ARC approach to structuring hierarchical adaptive systems. We identified the (abstract) QoS parameters needed in the negotiations between the two modules based on earlier theoretical work on joint source and channel coding. We also reasoned about the relation between the QoS parameters (e.g., latency and bit-error rate) and typical (internal) video-encoding parameters such as prediction mode and quantization step-size. The optimal setting of the video encoder depends on the context provided by the application on the one side, and communication performance on the other side. Whenever the context changes the video encoder has two options: 1) adapt locally, which is fast, and 2) engage a full negotiation with application and protocols, which is efficient system-wide.

In the future we would like to study the trade-offs between local adaptation and global negotiations. We are developing a mobile platform consisting of several modules connected with negotiated QoS interfaces. This platform enables experimental optimization of internal adaptation for situations in which theory does not suffice. Furthermore, we can determine experimentally for each module how much flexibility is required in practical implementations. Finally, it allows observing the emerging behavior of system-wide QoS negotiation.

Acknowledgements -- This work was conducted within the Ubicom program (www.ubicom.tudelft.nl) and funded by the TU Delft, DIOC research program. We thank Hylke van Dijk and Maarten Ditzel for useful comments on the manuscript.

References

[1] A. Lallet, C. Dolbear, J. Hughes, & P. Hobson "Review of scalable video strategies for distributed video applications", *Distributed Imaging IEE European Workshop*, pp. 2/1-2/7, 1999

[2] D. Saparilla & K.W. Ross, "Optimal streaming of layered video", *Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2000)* pp. 737 -746 vol.2, 2000

[3] A.E. Mohr, R.E. Ladner, & E.A. Riskin "Generalized Multiple Description Coding through Unequal Forward Error Correction", International Conference on Image Processing (ICIP), 1999.

[4] C.Y. Hsu, A. Ortega, & M. Khansari "Rate-control for robust video transmission over burst-error wireless channels" *IEEE Journal on selected areas in communications 17*, 1999

[5] C. Aurrecoechea, A.T. Campbell, & L. Hauw "A survey of QoS architectures" *7th International Workshop on QoS '99*, 1999

[6] H. van Dijk, K. Langendoen, & H. Sips "ARC: a bottom up approach to negotiated QoS" *3rd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2000),* December 2000

[7] A. van der Schaaf and R.L. Lagendijk "Independence of source and channel coding for progressive image and video data in mobile communications" *Visual Communications an Image Processing (VCIP 2000),* June 2000

[8] N. Färber, K. W. Stuhlmüller, & B. Girod, "Analysis of Error Propagation in Hybrid Video Coding with Application to Error Resilience", *International Conference on Image Processing (ICIP'99)*, pp. 550-554, October, 1999.