ADAPTIVE END-TO-END OPTIMIZATION OF MOBILE VIDEO STREAMING USING QOS NEGOTIATION

Jacco R. Taal, Koen Langendoen, Arjen van der Schaaf, Hylke W. van Dijk, and R. (Inald) L. Lagendijk

Faculty of Information Systems and Technology
Delft University of Technology, The Netherlands
{jacco,koen,hylke}@ubicom.tudelft.nl

ABSTRACT

Video streaming over wireless links is a non-trivial problem due to the large and frequent changes in the quality of the underlying radio channel combined with latency constraints. We believe that every layer in a mobile system must be prepared to adapt its behavior to its environment. Thus layers must be capable of operating in multiple modes; each mode will show a different quality and resource usage. Selecting the right mode of operation requires exchange of information between interacting layers. For example, selecting the best channel coding requires information about the quality of the channel (capacity, bit-error-rate) as well as the requirements (latency, reliability) of the compressed video stream generated by the source encoder. In this paper we study the application of our generic QoS negotiation scheme to a specific configuration for mobile video transmission. We describe the results of experiments studying the overall effectiveness, stability, and dynamics of adaptation of our distributed optimization approach.

1. INTRODUCTION

Mobile systems often operate in a highly variable context. The two dominant factors causing context variability are the user and the mobile channel. The characteristics of mobile transmission strongly depend on the user's location and 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 energy) are scarce in mobile systems. Handling variable workloads efficiently under variable conditions necessitates the use of collaborative adaptive modules.

The mobile system that we consider in this paper supports video 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 fluctuating conditions of the mobile channel.

In Figure 1 it is apparent that the video encoder and the protocol modules are not directly influenced by any context variability at either side of the system. However, if we optimize the system under global efficiency constraints, then the video encoding and protocol module will indirectly experience context fluctuations from neighboring modules when they adapt. Therefore, all modules will have to be context dependent, requiring their internal operation to be flexible and adaptive to local context fluctuations.

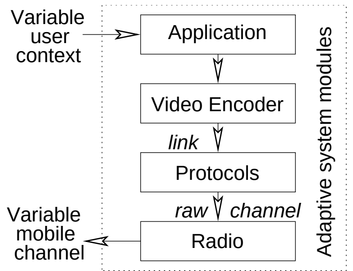

Fig. 1. Mobile video communication system.

Each module in an adaptive system usually contains many parameters that influence its behavior. System-wide optimization of the decision variables is difficult for two reasons. First, jointly optimizing many parameters yields computationally complex solution strategies. Second, every single module is a complex system in itself, whose application requires specific domain knowledge. To keep the optimization procedure manageable, we have to decompose 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 inner workings of other modules. Therefore, the adaptation of individual modules cannot be optimized separately [1, 2]. Instead, context and implementation details of the components must be exposed and shared, but in an appropriate format.

The problem we address in this paper is therefore essentially one of coordination. Figure 1 gives a computational view on the system, emphasizing functionality but hiding implementation aspects. Ideally the component-specific parameters must be tuned such that system behavior complies to its objectives while being constrained by its conditions. Taking an engineering view on the system, however, introduces additional conditions. Suppose we opt – for the sake of this paper – for a straightforward implementation. We then effectively map the functionality of Figure 1 to the limited processing resources of a mobile terminal. In a simple battery-powered mobile terminal, the components that compose Figure 1 *share* CPU, memory, and battery capacity. Consequently,

This work was conducted within the Ubicom program and funded by the TU Delft, DIOC research program.

the distribution of *shared resources* over the components must be added to the coordination process. Controlling QoS with more than a single parameter is a complex problem. Solutions do exist in literature, but they usually result in ad-hoc control structures [3, 4].

In previous work, we have introduced a generic QoS negotiation method, called Adaptive Resource Contracts (ARC) [5]. This method is able to handle complex systems, such as the one studied in this paper, and facilitates evolution. In this paper we apply the distributed and non-iterative ARC framework to the problem of mobile video streaming.

2. QOS NEGOTIATION: ARC

The ARC QoS negotiation method is illustrated in Figure 2. In conformance with the hierarchical setup shown in Figure 1, each module acts both as a server to 'higher' layers and as a client to 'lower' layers. The hierarchical concatenation of modules is important, because subsequent modules determine each others context. A QoS interface deals with the interaction of one module acting as server and one module acting as client. A contract is negotiated at the interface, holding 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 underlying server. The server (now approximately aware of 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, either by selecting an option and issuing it as a contract, or restarting the negotiation by formulating a 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 changes in context), 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.

As an example of QoS negotiation consider Figure 3. The abstract QoS is here represented in a two-dimensional space, the two parameters denoting capacity and quality. Contrary to how QoS is often used , an ARC interface reflects capacity (i.e. costs) in addition to quality parameters. The request from the client in Figure 3 is a range selection from the QoS space. Now the server knows what the client is interested in, and responds with a number of detailed offers. Each offer is in the initial request range, but gives a tighter description of the QoS that the server can offer within the current context. The client then marks a QoS range that includes at least one offer and sets it as the contract. Specifying a range rather than a single point for a contract leaves 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 negotiations.

The QoS interface is the result of a collaborative design by the server and the client; they share a consistent interpretation of the QoS parameters. For reasons of efficiency, run-time implementations need not be that explicit. Minimization of power dissipation is a system-wide *implicit* agreement. Another example of an implicit agreement is ranking of parameters which improves the integrity of the system. In case of a contract violation a server can continue operation in a predetermined way. In Figure 3, for instance, the server will degrade quality, utilize more capacity, or do both.

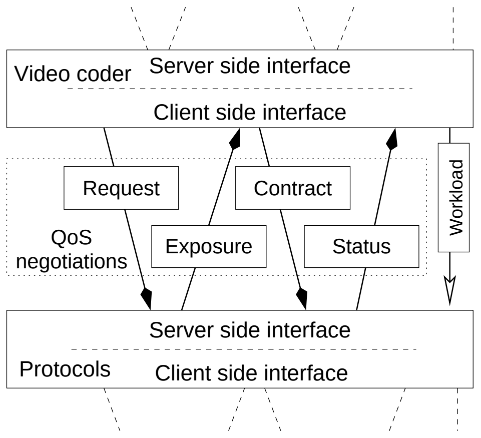

Fig. 2. ARC protocol outline.

3. MOBILE VIDEO STREAMING IMPLEMENTATION

The implementation of the mobile video communication system involves a video encoder, a protocol component, and a radio transmission component. In our experiments we concentrate on the video encoder and protocol components, and assume without loss of generality that the applied radio component is non-adaptive. The applied channel model is time-varying though. Both the video encoder and protocol component support the ARC framework for doing QoS negotiations. Their fundamentals are described in this section.

The QoS parameters at the interface between the video encoder and protocols components are given in Table 1. For an detailed description of the design of this interface we refer to [6].

Table 1. QoS parameters (descending priority).

Parameter	**Description**
latency (max)	The time required to transmit a single bit.
bit-error rate (max)	The net bit-error probability after FEC and ARQ.
CPU usage (max)	The allowed share of CPU capacity for protocols and transmission processes.
throughput (min)	The minimum net throughput.

3.1. Video Encoder

The video encoder implements a flexible H.263 encoder/decoder pair. Internally there are numerous video encoding parameters that can be tuned. In the context of ARC, we apply an abstract behavior description of the encoder, much like the work in [1, 2]. In this paper we consider as dominant decision variables the frame-skip N_{fs}, the maximal motion vector length R_{mv}, and the rate control's target bit-rate r. The variable user context is observed through the video

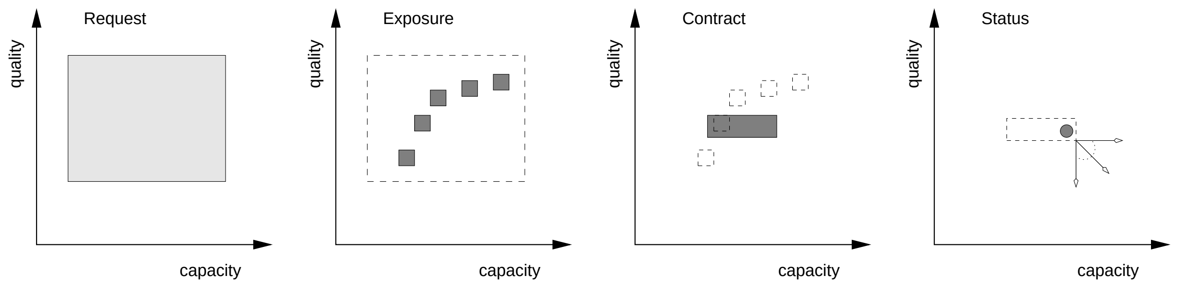

Fig. 3. ARC operation spaces.

sequence, namely through a model of the temporal predictability $\hat{\tau}$ and the (average) amount of variance σ_0^2.

The control model focuses on *a priori* estimation of the rate-distortion behavior as a function of R_{mv}, N_{fs}, $\hat{\tau}$, and the choice whether or not to use motion compensation. The model expresses the prediction gain G as follows:

$$G = \left(G_{\text{mv}} - (G_{\text{mv}} - G_0) e^{\frac{-R_{\text{mv}}}{L_{\text{mv}}(N_{\text{fs}}+1)}} \right) e^{\frac{-N_{\text{fs}}}{L_{\text{mc}}}} + 1 \qquad (1)$$

where G_0, G_{mv}, L_{mc}, and L_{mv} are the model parameters of $\hat{\tau}$. G_0 and G_{mv} are the prediction gains without and with motion compensation, respectively. L_{mc} is the motion coherence, describing the decay of prediction gain when the frame skip increases. L_{mv} is the average motion vector length, which characterizes the amount of motion in the sequence.

Given σ_0^2 we derive an estimate of the amount of variance to be encoded: $\sigma_d^2 = \sigma_0^2 / G$. The distortion is estimated by a parameterized rate-distortion curve that describes the performance of the quantizer and the arithmetic coder. The estimated quantization noise is

$$\sigma_q^2 = \sigma_d^2 \, 2^{2(b(e^{\frac{-r}{a}}-1)-r)} \qquad (2)$$

The distortion after transmission and decoding is a PSNR value estimated from the quantization noise including effects of skipped frames and bit errors. Skipped frames yield a PSNR based on the last received frame and $\hat{\tau}$. Bit errors degrade the PSNR assuming that one bit error destroys half of a Group of Blocks.

With our distortion model we evaluate different modes of operation. There is a trade-off between distortion and CPU utilization. The encoder offers the application a set of non-inferior points.

3.2. Protocols

The communication protocols run on top of a very simple radio that offers a TDMA scheme with fixed length transmission/receive slots to access the physical channel. Due to interference, fading, and other factors the data transmitted over the radio channel is subject to errors. The protocols employ two methods to counter the high bit-error rate (BER) of the physical channel: forward error correction (FEC) and automatic retransmit requests (ARQ).

Note that, contrary to many implementations, FEC is implemented in software. We use a Reed-Solomon protection scheme with four different code rates: 0%, 12.5%, 25%, and 50%. The code rate determines the maximum effective throughput that can be offered. We have run a number of off-line tests to determine the computational complexity and effective BER of the four code rates using white Gaussian noise. These results have been collected in a lookup table that is consulted during on-line execution.

For data that must be delivered reliably (i.e., without any error) packets are extended with a 32-bit checksum. When the receiver observes a checksum failure, it sends a retransmit request back to the sender, who will in turn re-send the data. ARQ increases latency (L) and reduces the throughput (T) that can be obtained. We use the following model to quantify the efficiency loss due to retransmits:

$$T_{\text{ARQ}} = \frac{T_{\text{FEC}}}{1 + P_{\text{err}}} \qquad (3)$$

$$L_{\text{ARQ}} = (1 + 2P_{\text{err}}) L_{\text{FEC}} \qquad (4)$$

where P_{err} is the probability that a packet is corrupted.

By combining the lookup tables for FEC and the ARQ model, the protocol layer can quickly evaluate what its best setting (code rate + enable/disable ARQ) is, given the current channel conditions and contract with the video encoder. When asked for an offer it prunes the inferior points out of the eight alternatives.

4. EXPERIMENTAL EVALUATION

The experiment set up is as follows. A pre-recorded video stream (`carphone`) is encoded at a mobile terminal, transmitted over a (simulated) wireless link and decoded at a base station. Latency requirements are such that *live* viewing is possible (< 0.5s). The radio has fixed settings: constant transmission power and constant modulation schemes during the length of the experiment. However interference at the physical channel will occur. Interference causes an increased bit error rate at the radio channel.

We present here the following three experiments. For all experiments the user requests the best quality possible within 100% CPU budget.

1. *Steady run*. There is constant interference on the mobile channel. Therefore neither of the components adapts during this experiment. Note the video encoder also does not adapt to changing characteristics of the incoming video source. We have run this experiment for a) a bad channel, BER $= 2\ 10^{-2}$ b) a medium channel, BER $= 10^{-2}$ and c) a good channel, BER $= 10^{-4}$. The raw channel bitrate is kept at $1.8\ 10^4$bit/s.

2. *Frozen run*. After 20 seconds from the start of the experiment the initial *medium* channel changes to a *bad* channel. The throughput is maintained at $1.8\ 10^4$bit/s. At $t = 47.5$s the channel changes to a *good* state. The protocols layer

adapts instantaneously to the changed raw channel conditions. Initially it keeps the contracted BER at the link to the video encoder but it has to sacrifice throughput at the link. The video encoder establishes an initial contract assuming a (worst-case) BER of $2\ 10^{-2}$ right after the start of the experiments. For the remainder of the experiment, all internal settings (and contracts) are frozen. To maintain the agreed CPU budget and real-time objectives, some frames will be skipped, which decreases the delivered quality.

3. *Adaptive run*. The physical channel and protocols layer behave as in the frozen run above. This time, however, the video encoder initiates QoS negotiations and adapts to the changed conditions. Like in the frozen run the net effect is that the video encoder maintains the agreed real-time and CPU-budget constraints.

Figure 4 shows the results of the experiments. The top diagram shows the BER of the raw channel for the steady cases as well as the changing channel case. The diagram in the middle shows the effects on the throughput delivered by the protocols to the video encoder. The bottom diagram has four curves, three for the "steady" runs and one for the adaptive run. The frozen run closely follows the "bad" steady run, and is left out for clarity. The curves plot the quality (PSNR) per received frame. As can be expected the "good channel" steady run has the highest quality. Quality variations over time are due to variations of the input source characteristics. Observe that the curve for the adaptive run switches between the three steady curves (medium→bad→good) when the channel conditions change. This shows that ARC-based negotiations succeed in selecting appropriate settings of the video coder that outperform a coder that assumes worst-case conditions.

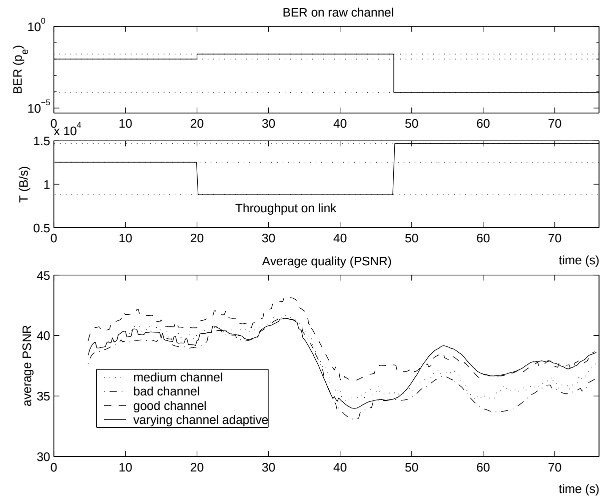

Fig. 4. Experiment results.

Compared to the *good* channel steady run the average quality of the frozen run is 1.97 dB less. The average quality of adaptive run is only 0.91 dB less than the steady run. In the adaptive run, 864 operation points were evaluated in three negotiations. The total time needed for these quality of service negotiations is 80ms. It is instructive to present the internal settings of the respective components for each of the experiments. Table 2 shows the results. The values shown for the frozen and adaptive run are averages, because the parameters are changing over time.

Table 2. Parameter settings.

	steady			frozen	adaptive
	medium	bad	good		
R_{mv}	12.00	9.00	12.00	9.00	14.30
r (bpp)	0.67	0.47	0.79	0.47	0.51
N_{fs}	1.15	1.03	1.07	1.01	1.04
Video CPU-budget	50%	45%	56%	52%	86%
FEC	25%	50%	0%	43%	38%
ARQ	0	0	0	0	0
Proto CPU-budget	6%	14%	2%	8%	7%

Until now we did not adapt to changes in source characteristics, but they vary drastically. An experiment in which the parameters were optimised taking into account the changing characteristics, improved the average quality with 1.1dB. This result was obtained using the *bad* channel and a CPU-budget of 30%. The user can trade quality for resources.

5. DISCUSSION

The experiments show that our ARC framework is able to improve the overall performance in cases where the channel conditions or video source characteristics fluctuate. Moreover the ARC framework makes it possible to keep the resource usage within bounds, even when the channel status is changing. It is able to make more efficient use of the available resources. The ARC-framework allows for a flexible implementation of modules. Therefore an ARC setup can operate in different environments: mobile or internet.

Stability problems may arise when the context changes occur too often, the optimiser might lag behind, and persists in making the wrong decisions. One way to avoid this problem, is to relax the contract margins. This allows the component to perform internal adaptations at the expense of being suboptimal.

References

[1] B. Girod and N. Farber, *Compressed Video Over Networks*, chapter Wireless Video, Marcel Dekker, 1999.

[2] T.-H. Lan and A. Tewfik, "Power optimized mode selection for H.263 video coding and wireless communications," in *IEEE Conference on Image Processing, (ICIP 98)*, 1998.

[3] G. Le Bodic, J. Irvine, and J. Dunlop, "Resource cost and QoS achievement in a contract-based resource manager for mobile communications systems," in *Proceedings of Eurocomm*, May 2000.

[4] M. Bechler, H. Ritter, and J. Schiller, "Quality of service in mobile and wireless networks: The need for proactive and adaptive applications," in *Hawaii Int. Conf. on System Sciences (HICSS-33)*, Jan. 2000.

[5] H. van Dijk, K. Langendoen, and H. Sips, "ARC: a bottom-up approach to negotiated QoS," in *3rd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2000)*, Monterey, CA, Dec. 2000, pp. 128–137.

[6] A. van der Schaaf, K. Langendoen, and R. Lagendijk, "Design of an adaptive interface between video compression and transmission protocols for mobile communications," in *11th Packet Video Workshop (PV-2001)*, Kyongju, Korea, Apr. 2001.