System-Level Power/Performance Analysis of Portable Multimedia Systems Communicating over Wireless Channels†

Radu Marculescu Amit Nandi
Department of ECE
Carnegie Mellon University
Pittsburgh, PA 15213
{radum,anandi}@ece.cmu.edu

Luciano Lavagno
DIEGM
Universita di' Udine
33100 Udine, Italy
lavagno@diegm.uniud.it

Alberto Sangiovanni-Vincentelli
Department of EECS
University of California
Berkeley, CA 94720
alberto@eecs.berkeley.edu

Abstract

This paper presents a new methodology for system-level power and performance analysis of wireless multimedia systems. More precisely, we introduce an analytical approach based on concurrent processes modeled as Stochastic Automata Networks (SANs) that can be effectively used to integrate power and performance metrics in system-level design. We show that 1) under various input traces and wireless channel conditions, the average-case behavior of a multimedia system consisting of a video encoder/decoder pair is characterized by very different probability distributions and power consumption values and 2) in order to identify the best trade-off between power and performance figures, one must take into consideration the entire environment (i.e., encoder, decoder, and channel) for which the system is being designed. Compared to using simulation, our analytical technique reduces the time needed to find the steady-state behavior by orders of magnitude, with some limited loss in accuracy compared to the exact solution. We illustrate the potential of our methodology using the MPEG-2 video as the driver application.

1. Introduction and objectives

Portable embedded multimedia systems have a few distinctive features that make them special within the general class of embedded systems. First, they are characterized by 'soft' real-time constraints and then may tolerate missed deadlines. In other words, their behavior is not necessarily characterized by a single number (the hard real-time constraint), as is the case for reactive embedded systems used in safety critical applications, but by a *probability* (or distribution of probabilities) which captures some sort of variability in the performance metrics. Another important characteristic is the notion of Quality of Service (QoS) which embraces all the non-functional properties of a system (e.g. power consumption, latency, jitter, cost, etc.). In multimedia systems, QoS requirements vary considerably from one media type to another. For example, video connections require consistently high throughput, but can tolerate reasonable levels of jitter and bit or packet errors. In contrast, audio applications do not require such high bandwidth, but place tighter restrictions on jitter and error rates. The ability to explore several design alternatives while trying to satisfy QoS requirements is of crucial importance, especially early in the design cycle to avoid costly redesign steps [1,2].

Based on the distinctive features discussed above, the objective of this paper is to propose a new modeling and analysis methodology for multimedia systems communicating over wireless channels. More precisely, we provide an *analytical* technique for *average-case* power/performance analysis that can be used early in the design cycle to identify the best match (in terms of performance and power consumption) among several possible application-architecture combinations. Indeed, the computational requirements of such systems show such large statistical variations that designing them based on a single fixed number that represents the worst-case behavior (typically, one or two orders of magnitude larger than the actual execution time [3]) would result in completely inefficient systems.

Despite the great potential for embedded system design, the area of average-case analysis has received little attention [3,17,20,21]. The target of our research is to investigate this very issue and, using abstract representations, provide quantitative measures of power/performance estimates that can be used in early stages of the design process. Our effort complements the existing results for worst-case time analysis [5,18,19] and is quite distinct from other approaches for performance analysis based on rate analysis [6], and adaptation process [3]. We also point out that, while simulation has been predominantly used to evaluate the average-case behavior, our analytical technique dramatically reduces the time needed to find the steady-state regime, with some limited loss in accuracy compared to the exact solution.

While our formalism is completely general, to illustrate its technicalities, we consider the MPEG-2 video application throughout the paper [13]. In its most abstract form, the application consists of an encoder-decoder pair talking to each other over an ideal channel (Fig.1(a)). As we move the level of abstraction down, the communication mechanism gets refined as shown in Fig.1(b) and the effects of the lossy communication channel start to become important. The prac-

†This research was supported in part by NSF under grant CCR-00-93104 and MARCO/DARPA Gigascale Silicon Research Center (GSRC).

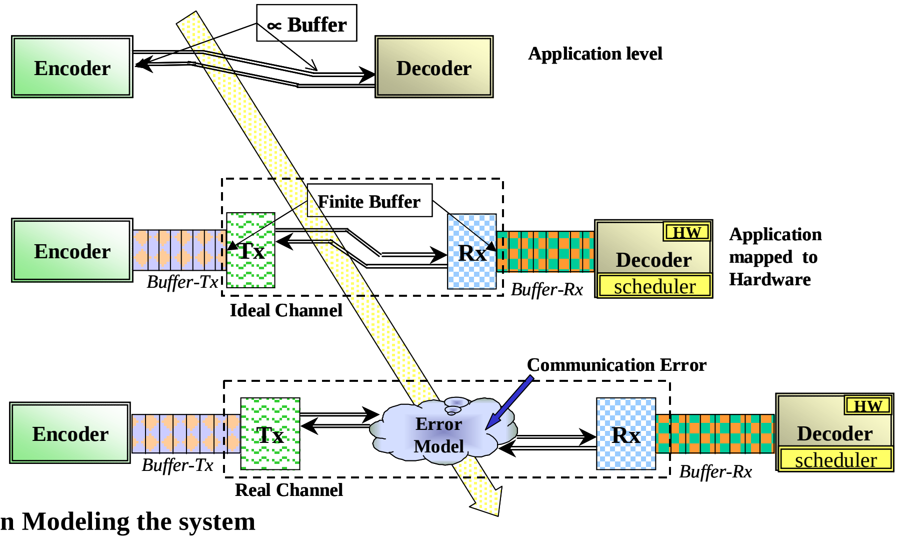
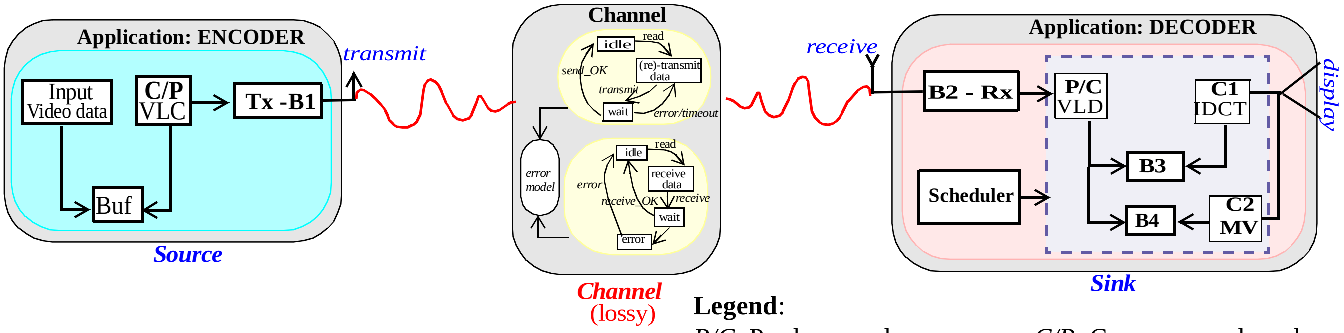


Fig 1(a). Stages in Modeling the system



Legend:
P/C: Producer and consumer *C/P*: Consumer and producer
B1, *B2*, *B3*, *B4*: buffers *Tx*, *Rx*: transmitter, receiver

Fig 1(b). Model of the complete system

tical problem that we are trying to solve is to find efficient mappings of the standard MPEG-2 onto a platform designed to support portable encoding/decoding devices that interact and communicate over a wireless lossy channel. Although the focus of our analysis is on the *decoder*, we show that, in order to identify the best trade-off between power and performance, one must take into consideration the *entire* environment (i.e., encoder, decoder, *and* communication channel) for which the system is being designed. By doing so, the designer can decide (at the highest level of abstraction) how the speed of the encoder should look like, how much re-transmission can be afforded, etc.

To carry out the analysis, we propose to use *concurrent Markov chains* as an effective formalism for system-level analysis. The challenge is to model - at process level - the system consisting of encoder, decoder, and lossy communication channel, and measure the degradation in performance (i.e., QoS) seen at the decoder-level when the channel becomes lossy. This is possible since we model explicitly processes that have both computation and communication states. Also, inside processes, these states have associated delay and power costs which are taken into consideration during the analysis step. We believe that our analysis brings a new perspective by providing an unified model for the *application*, *architecture*, and *communication medium*.

To evaluate our methodology, we analyze the Markov model under different conditions of the wireless channel. This may be a significant contribution since we apply *fast* and *precise* techniques for analytical calculations of performance metrics (power, delay) to communicating and interacting processes that represent multimedia applications. Using a fully analytical solution helps in avoiding lengthy profiling simulations for predicting power and performance figures. This is important for multimedia systems since thousands of runs are typically required to gather relevant statistics for average-case behavior. Considering that 5 min. of compressed MPEG-2 video needs roughly 1.2 Gbits of input vectors to simulate, the impact of having such a tool to evaluate power/performance estimates becomes evident.

Our analysis on power and delay variation (as a function of the channel error rate) shows that some encoding rates simply make the communication too expensive; that is, the design constraints that should be satisfied become too severe (either in terms of power consumption and/or end-to-end latency). In these cases, encoding data at a certain rate becomes a *necessity*, not an option based on designers taste.

Also, starting with some anticipated behavior of the channel, we show what is achievable (in terms of power and/or end-to-end latency) by doing trade-offs between computation and communication needs. From this perspective, our paper makes a first step towards providing *design support* for platform-based implementation of multimedia applications. This is important since most of the published papers to date (both in design automation and mobile computing communities) are focussed only on providing *run-time support* [10,11] for PC-based platforms when the *Advanced Configuration and Power Interface* (ACPI) is available [16].

The paper is organized as follows: Section 2 presents the overall modeling strategy. In Section 3, we present a detailed analysis of the MPEG-2 encoder/decoder application. The power/performance results are described in Section 4, together with some implications in the design process. Finally, we conclude by summarizing our main contribution.

2. Modeling

To model the application of interest, we use *process graphs* where each node corresponds to a process in the application. Communication between processes is achieved using *event* and *wait* synchronization signals. Data is exchanged asynchronously between processes through *buffers* that behave like fixed-length *FIFOs*. Furthermore, the process graph is characterized by *execution rates* which, under the hypothesis of exponentially distributed activity durations, can be used to generate the underlying Markov chain [12]. This way, the whole process graph translates into a network of automata called Stochastic Automata Network (SAN) [22].

2.1 The multimedia stream abstraction

At the highest level of abstraction, the MPEG-2 application consists of three top-level components: the *Source* (encoder), the *Sink* (decoder), and a lossy *Channel* (communication medium) (Fig.1(a)). The *Source* generates a continuous sequence of packets which are relayed by the *Channel* to the *Sink*, which then displays them. The *Channel* is assumed to support asynchronous communication between the *Source* and the *Sink*. In addition, the channel is unreliable and may lose messages.

We model the process graph obtained from the MPEG-2 application following the *Producer-Consumer* paradigm (Fig.1(b)). The multimedia input stream is first analyzed at macroblock level in intra/motion-compensated format (namely, I, P and B frames) [13]. To do this we use the *mpeg-stat* [15] tool which, for a given clip, provides the exact way in which the macroblocks would be decoded by the MPEG decoder. We note that profiling with *mpeg-stat* is *not* the same thing with simulation since it consists of just parsing the input video stream and collecting statistics on the *number* and *type* of macroblocks. Later, we feed these parameters back to the analytic model and use them for our calculations.

The three basic *actions* which support the flow of data are *transmit*, *receive*, and *display*, which respectively signal the transfer of packets from *Source* to the *Channel*, from *Channel* to the *Sink*, and finally their display at the *Sink*. Specific rates (λ_{trans}, λ_{rec}, and λ_{disp}) are associated with the actions of *transmit*, *receive*, and *display*, respectively.

2.2 *Source* modeling

The *Source* simply transmits frames over the communication medium at a rate of λ_{trans}. Since the objective of our model is to understand the behavior of the *decoder* under different error-levels in the *Channel*, the encoder is simply modelled as a *Producer* (the *VLC* block in Fig.1b) which can encode frames at *varying* encoding speeds; that is, λ_{trans} may vary to adapt to the channel behavior. The encoded frames are put into the buffer *B1* which is associated to the transmitter *Tx*.

2.3 *Channel* modeling

The *Channel* models the communication medium; it accepts frames from the *Source* (via the action *transmit*), and then either passes them on to the *Sink* (via the action *receive* at rate λ_{rec}), or loses them (via the action *loss* at rate λ_{loss}). An ideal channel is described by setting λ_{loss} to zero.

We model the *Channel* in Fig.1 as an automaton which simply transmits packets from buffer *B1* to buffer B2. The packets may be sent over the channel with error, or may be lost (at rate λ_{loss}) during transmission. If the packets are transmitted successfully, then they are stored into the buffer *B2* (i.e., the *Rx-buffer*). The '*error model*' block in Fig.1(b) models the error over the channel which in our case is assumed to be Gaussian.

The action *transmit* in *Channel* is passive since the *Channel* accepts frames from the *Source* at any rate. However, when a message is lost due to some channel error, a *re-transmission* action is initiated and the *Source* re-transmits the message until it gets to the *Sink*. We assume a *non-blocking* communication scheme, where the *Source* may send packets to the *Sink* even if the *Rx-buffer* (i.e. *B*2) is full (in which case the packet is lost). This is because, in a real scenario, the transmitter has no way of knowing whether the *Rx-buffer* is full or not, and hence it will transmit the packets irrespective of the buffer condition. The receiver can, however, request a re-transmission of these lost packets when the buffer is free.

We point out that, in the most general case, the *Channel* can actually model the transmission protocol in all details. For instance, we may assume a full duplex communication and model the channel with two automata on the transmitter side (one of which deals with sending data, and the other one with handling acknowledgements), and another two automata on the receiver side (one for receiving data, and another one for sending acknowledgements). These automata make communication possible via synchronization signals and shared buffers.

To complete our channel modeling, we need to model the error which determines whether the synchronization signals are to be sent or not. To keep our analysis general, we model the channel as an *error process* which not only models the error over the channel, but also *synchronizes* the *Rx* and *Tx* buffers at a certain rate determined by the communication speed. This process knows when an error has occurred and assumes certain durations (passed as parameters) during which an error can be detected and/or the data can be transmitted. If the error process determines that the packet is affected by error, it waits in the error state for a duration in which the error can be detected, and then retransmits the packet. If there is no error, then this process waits in the transmit state for the duration necessary to transmit the packet, and then synchronizes the *Rx* and *Tx* buffers to update their status. If at this point the *Rx-buffer* is full, it assumes that the packet is lost and retransmits it.

2.4 *Sink* modeling

The *Sink* (or decoder) receives frames from the communication channel via buffer *B2* and displays them at rate λ_{disp}. The *Sink* consists of the *VLD (Variable Length Decoder),* the *IDCT/IQ* unit and the *MV (Motion Vector)* unit. We describe the *VLD* process as the *Producer* and the other two units as two *Consumer* processes (*C1* and *C2* in Fig.1(b)). The *VLD* block decodes the packets which arrive at buffer *B2*, generates macroblocks that are put into the buffer *B3* and motion vectors that are put into the buffer *B4*, with a rate λ_{VLD}. These packets are picked up by the *Consumer* processes to compute *IDCT's* (at rate λ_{IDCT}) and decode motion vectors (at rate λ_{MV}) which are used to reconstruct the frames. These individual processing rates are obtained either from application specification or off-line data profiling.

As we have described earlier, the exchange of data between the *Producer (VLD)* and the *Consumer* processes *(IDCT/IQ* and *MV*) occurs through buffers *B3* and *B4*. These buffers have been modelled with five different states, which represents the number of elements stored in them, starting from zero (empty) to four (full). The buffer accesses in the decoder constitutes accesses to the memory which is a shared resource. Hence, only a single process is allowed to access it at any time. The average length of these buffers reflects their *utilization* over time. Since the variable length decoding constitutes a simple table-lookup, it is assumed to run on some dedicated hardware, while the *IDCT/IQ* and the *MV* units run on a shared processor. The scheduling is based on a first-come first-serve (FCFS) basis, without preemption. This is not a limitation since this scheduler itself can be easily modeled within our framework.

3. Analysis

The objective of the SAN analysis is the computation of the stationary probability distribution π for an *N*-dimensional system consisting of *N* stochastic automata which operate and interact concurrently. This involves two major steps: 1) SAN model construction and 2) SAN model evaluation. The following two sub-sections briefly describe these two steps. For more details, the reader is referred to [7,22].

3.1. The SAN model construction

The SAN model can be described using continuous-time Markov processes based on *infinitesimal generators*:

$$Q = \begin{bmatrix} -\sigma_{0,0} & \sigma_{0,1} & \sigma_{0,2} & \cdots \\ \sigma_{1,0} & -\sigma_{1,1} & \sigma_{1,2} & \cdots \\ \sigma_{2,0} & \sigma_{2,1} & -\sigma_{2,2} & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix} \qquad (1)$$

with $\sigma_{i,i} = \lim_{t \to 0} \frac{1 - p_{i \to i}}{t} = -p'_{i \to i} \quad i = 1, 2, \ldots, n$,

$\sigma_{i,j} = \lim_{t \to 0} \frac{p_{i \to j}}{t} = p'_{i \to j} \quad i, j = 1, 2, \ldots, n \ (i \neq j)$, and $\sum \sigma_{i,j} = \sigma_{i,i} \quad i, j = 1, 2, \ldots, n (i \neq j)$, where $p_{i \to j}$ is the transition probability from state *i* to state *j* during time 0 to *t*, and $p'_{i \to j}$ is its derivative. Each entry σ_{ij} in the infinitesimal generator represents the *execution rate* of the process in that particular state [12].

The automata interact via *synchronization signals* (transitions). More precisely, a transition in one automaton may force a transition to occur in one or many other automata. Given *N* stochastic automata (with associated matrices $Q^{(1)}, Q^{(2)}, \ldots, Q^{(N)}$) which interact via *E* synchronizing events (*j* = 1, 2,..., *E*), the infinitesimal generator of the system can be written as:

$$Q = \sum_{j=1}^{2E+N} \bigotimes_{i=1}^{N} Q_j^{(i)} \qquad (3)$$

This quantity is called the *global descriptor* of the SAN and it can be written as a sum of tensorial[1] products as shown in [7,12].

3.2. Performance model evaluation

Once we have the SAN model, our goal is to find out its *steady-state* behavior. This is simply expressed by the solution of the equation

$$\pi \cdot Q = 0 \qquad (4)$$

with the normalization condition $\pi \cdot e = 1$, where π is steady-state probability distribution and *e* is a column vector s.t. e^T= (1,1,...,1).

[1] $X = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}, Y = \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ y_{31} & y_{32} & y_{33} \end{bmatrix}, X \otimes Y = \begin{bmatrix} x_{11}Y & x_{12}Y \\ x_{21}Y & x_{22}Y \end{bmatrix}$

In order to avoid the state explosion problem, we solve eqn. (4) using numerical methods that *do not* require the explicit construction of the matrix Q but can work with the descriptor in its compact form. To solve for π, we use the *power method* which can be applied to the discretized version of Q. More precisely, from Q one can easily construct another matrix P s.t. $P = I + (\Delta t)Q$ (where $\Delta t \leq (1/(max_i|\sigma_{i,i}|))$) which can be also written as a tensorial product [12]. Thus, the iterative process of calculating π becomes

$$\pi^{l+1} = \pi^{l} + (\Delta t \times \pi^{l}) \times \left(\sum_{j=1}^{2E+N} \bigotimes_{i=1}^{N} Q_j^{(i)} \right) \qquad (5)$$

and the underlying operation which we need to compute very efficiently is $\pi \bigotimes_{i=1}^{N} Q^{(i)}$. Fortunately, exploiting the properties of tensorial product (which are unique to the SAN model!) this can be done using only $\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} n_i$ multiplications, where n_i is the numbers of states in the *i-th* automaton [7,12]. We note that this is far better than the brute-force approach which would require $\left(\prod_{i=1}^{N} n_i \right)^2$ multiplications.

Once the steady-state probability distribution is determined, different performance measures such as *throughput*, *utilization*, *average response time* can be easily derived. To this end, we use the *true rates* of the activities, which are based on the probability of each activity being enabled. For instance, the rate at which the *Sink* displays packets is directly influenced by the loss-rate over the *Channel*. The true (or equilibrium) rate of an activity is thus given by the specified activity rate multiplied by the *probability* that the activity is enabled.

3.3. System-level latency estimation

The end-to-end latency of our system is controlled by three different segments, namely the *Source*, the *Channel*, and the *Sink* [14]. Hence, the total latency of the system is:

$$latency_{total} = latency_{Source} + latency_{Channel} + latency_{Sink}$$

where the source latency is given by $latency_{Source} = 1/\lambda_{trans}$, the channel latency is given by $latency_{Channel} = (avg\_frames\_\mathrm{rec})/\lambda_{trans}$, and finally, the sink latency is given by $latency_{Sink} = (avg\_frames\_\mathrm{sink})/\lambda_{disp}$. The average number of frames received (*avg_frames_rec*), can be estimated from the average buffer length of *B2*, while the buffers *B3* and *B4* provides estimates on the *avg_frames_sink*.

3.4. System-level power estimation

For a subsystem k, the average power consumed is given by:

$$P^{(k)} = \sum_{all\ \ i} \pi_i \cdot P_i + \sum_{all\ \ i,j} \lambda_{ij} \cdot P_{ij} \qquad (6)$$

where P_i and P_{ij} represent the power consumption per state and per transition, respectively, π_i is the steady-state probability and λ_{ij} is the transition rate associated with the transition between states i and j. Having already found the steady-state regime, a π_i value (for a particular i) can be determined by summing up the appropriate components of the global probability vector π. The P_i and P_{ij} costs are determined during an off-line step using other proposed techniques [8].

To obtain the power values, we use the Wattch [9] simulator that estimates the CPU power consumption based on a suite of parametrized power modes. For instance, by specifying a low-power *Strong-Arm* like architecture, we obtain an average power value of 0.356W for the *VLD* module, 0.4W for the *IDCT* and 0.51W for the *MV* unit, at a clock frequency of 100MHz. For transmitter and receiver units (*Tx* and *Rx*), we estimate the power consumption as in [11], where the transmitter power varies with the square of the distance between source and destination. For a piconetwork environment, assuming conservatively a distance of 100m and a transmission rate of 1Mb/sec., we estimate a transmitting and receiving power of 0.5W and 0.1W, respectively.

Using these power figures, we can determine the average power characterization of the entire system under varying workloads. We present these results in the next section.

4. Results and discussion

To specify the system that we want to analyze, we chose the Stateflow component of Matlab which uses the semantics of Statecharts, formally proposed by Harel [4]. To create the Stateflow model of the MPEG-2 video application, the sequential C code of the encoder and decoder was split into eleven processes and the communication among processes made explicit by using eighteen synchronization signals. Of the eleven processes, five processes are associated to the MPEG application itself (encoder and decoder), four to the buffers, and two to the schedulers (CPU and memory). We note that the overall system has more than three million states which would be prohibitive for any kind of direct analysis based on building the global descriptor of the system. The SAN approach can perform this analysis since, as explained in Section 3, it exploits the tensorial property and then avoids building the global matrix Q.

Once the SAN model is constructed, we analyze its steady-state behavior and derive the metrics of interest (like state occupancy, buffer length, etc.) using the iterative approach in eq. (5). All the steady-state results that we obtained using the analytical approach, have been also *validated* by extensive simulations in Matlab.

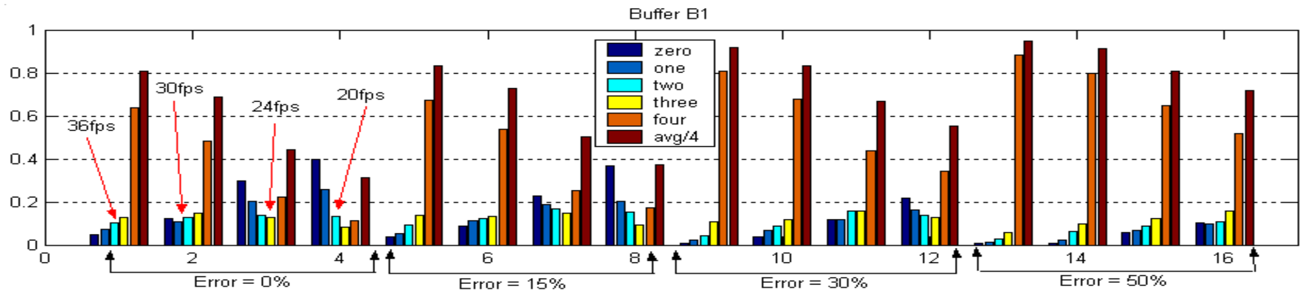


Fig.2. Steady-state probabilities for buffer B1 and average buffer length estimates

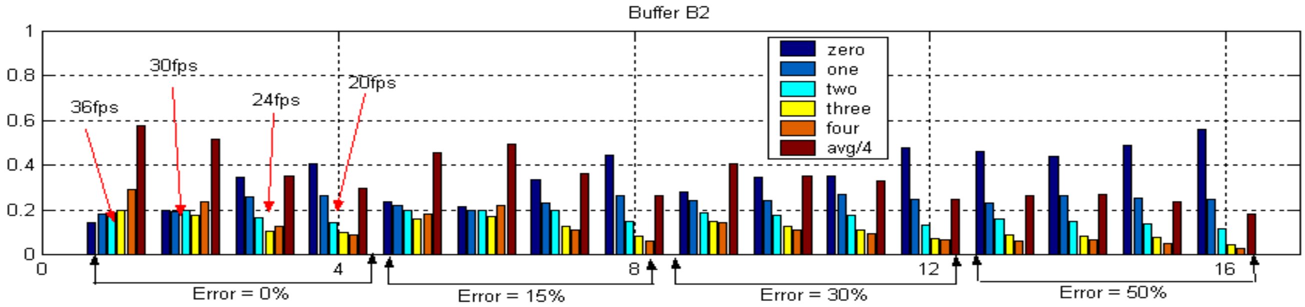


Fig.3. Steady-state probabilities for buffer B2 and average buffer length estimates

In the experiments reported in this section, we use the SAN analytic technique to determine the system behavior for various encoding speeds (frame-rates) and under varying error-levels in the channel. To this end, we observe the system under 0%, 15%, 30% and 50% error-level in the channel, at four different encoding speeds, namely 20, 24, 30 and 36 frames per second (*fps)*. The error in the channel is assumed to be Gaussian, though results can be easily derived for other types of distributions. The capacity/bandwidth of the channel is assumed to be sufficient for transmitting data at the highest frame-rate when channel error-level is zero.

We first present the buffer-length distributions corresponding to buffer *B1* (Fig.2). The buffer *B1* has five slots: ‘one’, ‘two’, ‘three’, ‘four’ (full), plus ‘zero’ (empty) which corresponds to storing one, two,... or zero items (that is, macroblocks). Each group of six bars in Fig. 2 (that is, zero, one, two,..., four, and the average buffer-length normalized by 4[1]) represents a *run* that corresponds to a certain *frame-rate* of the encoder and a certain *error-level* in the channel. For instance, the 2nd run in Fig.2 corresponds to having 30*fps* sent over an ideal channel, and a probability of 0.48 of getting *B2* full. Also, for the same ideal conditions, the average buffer length is 0.7×4 = 2.8 for 30*fps* but it becomes 0.32×4 = 1.28 if the encoding speed decreases to 20*fps*.

By analyzing all 16 runs in Fig.2, we observe that for a given level of error, decreasing the encoder speed makes the buffer less congested (in the 1st run, the average buffer length is 3.24, while in the fourth run it is only 1.28), while increasing the error-level in the channel increases the congestion (in the 1st run, the average buffer length is 3.24, while in the 13th run it becomes 3.79). These results suggest that if the error in the channel is high, then the system (per overall) needs to slow-down and adapt to channel behavior.

Next, in Fig.3, we consider the buffer-length distribution of buffer *B2*. The bars have been organized in the same manner as in the previous case; that is, the first four runs correspond to having an ideal channel, the next four to 15% error, so on so forth. We observe that increasing the error-level, decreases the average buffer-length. For instance, in the 1st run, the average buffer length is 2.32 while in the 13th run it is only 1.06. We also observe that the average buffer length (across al runs) is about 1.5, which is more than twice shorter compared to the length 4 which would be predicted by the worst-case analysis.

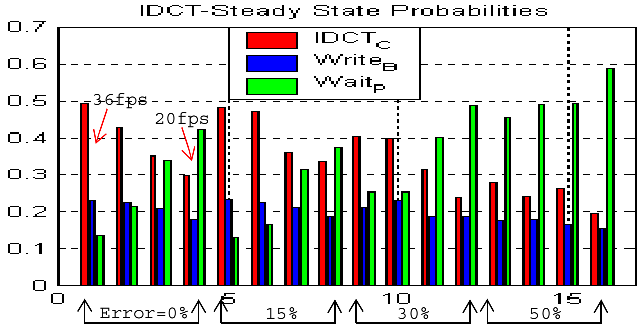


Fig.4. Steady-state probabilities for IDCT unit

The next set of plots (Fig.4) show the steady-state behavior of the *IDCT/IQ* process in the *Decoder*, again as a function of frame-rates and error-level in the channel. The three bars in each run represent the probability of the *IDCT* process spending its time in computing (*IDCT*$_C$*)*, writing

[1] This normalization has been done to save some space in the plots, and has no other implications.

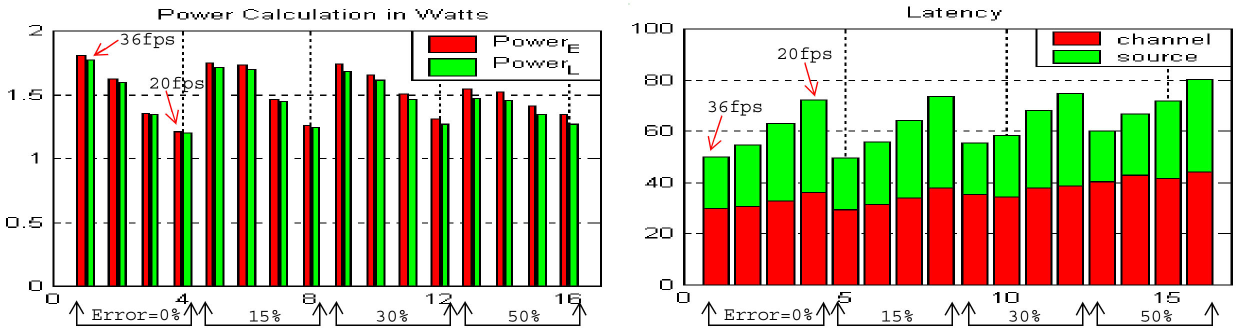


Fig.5. Power and Latency plots for different frame-rates and error levels

into the buffer (*Write*$_B$*)*, and waiting for data-packets to arrive (*Wait*$_P$*)*, respectively. From this plot, it is clear that the probability of the processes computing and writing into the buffer decreases, as the error-level increases and the frame-rate decreases. We also note that considering the *entire* environment into steady-state calculations (for buffer lengths and decoder states) is very important. For instance, from Fig. 3, we can see that by ignoring the contribution of the noisy channel, we may overestimate the average length of buffer *B2* by more than 30% (it may get to more than 100% for 50% noise in the channel!). In terms of probabilities of *IDCT* states (Fig. 4), we can see that the overestimation can be more than 40%. These overestimations will further propagate and severely affect the power/energy calculations for the decoder.

Based on the steady-state results above, we next present the estimates of the *power consumption* and *latency* values for various error-levels and frame-rates (Fig.5). This is important since they are directly related to the QoS of the system. Regarding the power results (Fig.5, left plot), we must note that an unsuccessful packet transmission includes two cases. In the first case, the packets may reach the *Sink* correctly, but the data in the packets may be corrupted (*Power*$_E$ bars in Fig.5), while in the second case the packets may be completely lost (*Power*$_L$ bars). This distinction arises because, in the former case, the decoder comes to know that the packet is corrupt only *after* it is completely received, while in the latter case, a packet lost in transmission is *not* received at the receiver (thus, no power is wasted in the reception process). The first bar of each run shows the power consumed in the first case, while the second column shows the power consumed by the receiver in the second case, which is obviously less. We observe that the receiving power decreases with the decrease in the frame-rate. However, this rate of decrease also decreases with the error-level (the change at 0% error-level is more that 3.5 times the change at 50% error-level).

Finally, in Fig. 5 (right plot), we present the latency of the system under varying channel conditions. We see that the latency not only increases with the frame-rate decrease, but also varies as a function of the error-level. Since the number of retransmissions due to error in the packets predominates, we would expect that the spread of the channel latency values should constantly decrease as the error-level increases. However, when we move between 0% to 15% error, we observe that the spread of latency values actually increases. This is because, within this error range, the number of retransmissions due to the full buffer condition decreases.

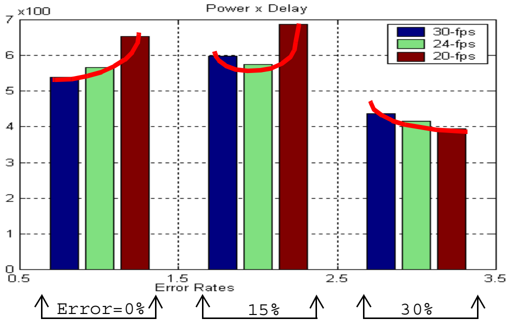


Fig.6. Power×Delay plots for different frame rates and error levels

Before concluding this section, we show the *power*×*delay*[1] figures (Fig.6) which is an important metric in low-power design. Since the system processes packets in a pipelined manner, the delay of the system is directly proportional to the maximum delay seen in any of the stages. Also, since the *volume* of data that is transmitted per time unit varies as a function of the encoding speed and the error in the channel, we need to consider the rate of packets that have been successfully processed (that is, rate of *committed packets*). For instance, the video data for 10 sec could be encoded at either 36*fps* or 20*fps*, thus resulting in very different volumes of data that are sent over the channel. Further, if the error is high, there will be a large number of packets that are actually not processed because they are going to be lost during the transmission. Hence the *power*×*delay* product is computed as: $power \times \left[\underset{all\ buffers}{MAX} (\overline{avg\_buffer\_length}) \times \frac{1}{\lambda_{eff}} \right]$, where λ_{eff} is the rate of committed packets.

[1] We note that the inverse of (power×delay) is actually the throughput per Watt or number of committed frames per Joule which may be another metric of interest.

From Fig. 6, we observe that, in the case of 0% error, the value of the *power*×*delay* product is minimum at 30*fps*. For 15% and 30% error rates, 24*fps* and 20*fps* show the minimum values, respectively. These results obtained via analysis were also verified (for consistency) against simulation and the agreement was perfect. We also point out that a similar analysis can be easily carried out using the *energy*×*delay* metric. This may be of interest since the *power*×*delay* figure is good only if one optimizes for energy.

Finally, the CPU time needed in the iterative algorithm that computes the probability distribution π as in eq. (5) is about 20-50 secs/iteration (we generally need a few tens of iterations to converge to the steady-state solution), while the error is less than 10% compared to the simulation results. This is orders of magnitude faster than the active simulation time (which typically takes about 20~25 hours, for a 2 minute video clip) required to obtain the same results. Hence, the analytical approach can significantly cut down the design cycle and, at the same time, enhance the opportunities for exploring the design space.

5. Conclusion

We presented a new modeling and analysis technique based on concurrent processes (specified with concurrent Markov chains) which can be effectively used to integrate power and performance metrics for multimedia systems communicating over wireless channels. We showed that, under various input traces and wireless channel conditions, the steady-state behavior of the multimedia system (encoder, decoder, and channel) is characterized by very different probability distributions and power consumption values. Also, our analysis on power×delay product variation showed possible trade-offs between computation and communication needs when the encoding rate of the encoder adapts to the wireless channel behavior.

The relevance of our approach is clear when we examine the results presented in the previous section. To obtain the best results in terms of power and performance, one must *not* just consider the design in isolation, but also take into account the *entire environment* (i.e., encoder, decoder, *and* channel) for which the system is being designed. Only by doing so it is possible to get a deep understanding of the possible power/performance trade-offs. Our analysis in Section 4, is the first step towards achieving such a goal.

Based on the results obtained, we believe that analytic approaches for system-level design should be the object of intense research because of their advantages over simulation-based techniques. Indeed, starting at a higher level of abstraction than commonly done offers the opportunity of using more formal approaches that are likely to increase the quality of the design and reduce time-to-market.

6. References

1. F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara, ‘Hardware-Software Co-Design of Embedded Systems: The POLIS Approach,’ Kluwer Academic Publishers, 1997.
2. A. Ferrari, A. Sangiovanni-Vincentelli, ‘System Design: Traditional Concepts and New Paradigms,’ *Proc. ICCD*, Austin, TX, Oct. 1999.
3. A. Kalavade, P. Moghe, ‘A tool for performance estimation of networked Embedded End-Systems,’ *Proc. DAC*, San Francisco, CA, June 1998.
4. D. Harel, ‘Statecharts: A visual formalism for complex systems,’ *Sci. Comp. Prog*, Vol. 8, 1987.
5. S. Malik, M. Martonosi, Y.-T. Li, ‘Static Timing Analysis of Embedded Software,’ *Proc. DAC*, Anaheim, CA, 1997.
6. A. Mathur, A. Dasdan, R. Gupta, ‘Rate Analysis for Embedded Systems,’ *ACM TODAES*, Vol. 3, no. 3, July 1998.
7. P. Fernandes, B. Plateau, ‘Efficient Descriptor-Vector Multiplications in Stochastic Automata Networks,’ *Journal ACM*, Vol. 45, May 1998.
8. T. Simunic, L. Benini, G. De Micheli, ‘Cycle-Accurate Simulation of Energy Consumption in Embedded Systems,’ *Proc. DAC*, New Orleans, June 1999.
9. D. Brooks, V. Tiwari and M. Martonosi, ‘Wattch: a framework for architectural-level power analysis and optimizations,’ *Proc. ISCA*, June 2000.
10. T. Simunic, L. Benini, P. Glynn, G. De Micheli, ‘Dynamic Power Management for Portable Systems,’ in *Proc. Mobicom*, Boston, MA, Aug. 2000.
11. W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, ‘Energy-Efficient Routing Protocols for Wireless Microsensor Networks,’ in *Proc. HICSS*, Jan. 2000.
12. W. J. Stewart, ‘An introduction to the Numerical Solution of Markov Chains,’ Princeton Univ. Press, New Jersey, 1994.
13. J. L. Mitchell, W. B. Pennebaker, C. E. Fogg, and D. J. LeGall, ‘MPEG Video Compression Standard,’ Chapman & Hall, I. T. P., 1996.
14. H. Bowman, J. Bryans, and J. Derrick, ‘Analysis of a multimedia stream using stochastic process algebra,’ *Proc. Intl. Workshop on Process Algebras and Performance Modelling*, Nice, Sept. 1998.
15. http://bmrc.berkeley.edu/ftp/pub/multimedia/mpeg/stat/
16. Intel, Microsoft and Toshiba, ‘Advanced Configuration and Power Interface Specification,’ *http://www.intel.com/ial/powermgm/specs.html*, 1996.
17. P. Lieverse, P. Van der Wolf, E. Deprettere, and K.Vissers, ‘A Methodology for Architecture Exploration of Heterogeneous Signal Processing Systems,’ *Proc. SiPS*, 1999.
18. T.-Y.Yen, W. Wolf, ‘Performance Estimation for Real-Time Distributed Embedded Systems’, *Proc. ICCD*, Oct. 1995.
19. K. Suzuki, A. Sangiovanni-Vincentelli, ‘Efficient Software Performance Estimation Methods for Hardware-Software Codesign’, *Proc. DAC*, June 1996.
20. A. Xie, P.A. Beerel, ‘Accelerating Markovian Analysis of Asynchronous Systems using State Compression’, in *IEEE Trans. on CAD*, July 1999.
21. T. Zhou, X. Hu, and E.Sha, ‘A Probabilistic Performance Metric for Real-Time System Design’, in *Proc. CODES*, May 1999.
22. B. Plateau, K. Atif, ‘Stochastic Automata Network for Modelling Parallel Systems,’ *IEEE Trans. on Software Engineering*, Vol. 17, no. 17, pp. 1093-1108, 1991.