xTune: A Formal Methodology for Cross-Layer Tuning of Mobile Embedded Systems

MINYOUNG KIM, MARK-OLIVER STEHR, and CAROLYN TALCOTT, SRI International
NIKIL DUTT and NALINI VENKATASUBRAMANIAN, University of California, Irvine

Resource-limited mobile embedded systems can benefit greatly from dynamic adaptation of system parameters. We propose a novel approach that employs iterative tuning using lightweight formal verification at runtime with feedback for dynamic adaptation. One objective of this approach is to enable trade-off analysis across multiple layers (e.g., application, middleware, OS) and predict the possible property violations as the system evolves dynamically over time. Specifically, an executable formal specification is developed for each layer of the mobile system under consideration. The formal specification is then analyzed using statistical property checking and statistical quantitative analysis, to determine the impact of various resource management policies for achieving desired timing/QoS properties. Integration of formal analysis with dynamic behavior from system execution results in a feedback loop that enables model refinement and further optimization of policies and parameters. We demonstrate the applicability of this approach to the adaptive provisioning of resource-limited distributed real-time systems using a mobile multimedia case study.

Categories and Subject Descriptors: C.3 [**Special-Purpose and Application-Based Systems**]—*Real-time and embedded systems*

General Terms: Design

Additional Key Words and Phrases: Cross-layer optimization, iterative tuning, formal methods

ACM Reference Format:
Kim, M., Stehr, M.-O., Talcott, C., Dutt, N., and Venkatasubramanian, N. 2012. xTune: A formal methodology for cross-layer tuning of mobile embedded systems. ACM Trans. Embedd. Comput. Syst. 11, 4, Article 73 (December 2012), 23 pages.
DOI = 10.1145/2362336.2362340 http://doi.acm.org/10.1145/2362336.2362340

1. INTRODUCTION

The next generation mobile embedded applications are highly networked and involve end-to-end interactions among multiple abstraction layers (application, middleware, OS, hardware architecture) in a distributed real-time environment. An overarching characteristic of these applications is that they are often data-intensive and rich in multimedia content with images, video, and audio data that is fused together from disparate distributed information sources. The content-rich data is expected to be obtained from, delivered to, and processed on resource-constrained devices (sensors, PDAs, cellular handsets) carried by users in the distributed network. Clearly, in such a scenario, the dual goals of ensuring adequate application QoS (quality of service)

Preliminary versions of this article appeared in Kim et al. [2007a, 2007c].
This work was partially supported by NSF grants 0615438, 0615436, and 0932397.
Authors' addresses: M. Kim, M.-O. Stehr, and C. Talcott, Computer Science Laboratory, SRI International, Menlo Park, CA 94025; email: {mkim, stehr, clt}@csl.sri.com; N. Dutt and N. Venkatasubramanian, School of Information and Computer Sciences, University of California, Irvine, Irvine, CA 92697; email: {dutt, nalini}@ics.uci.edu.

DOI 10.1145/2362336.2362340 http://doi.acm.org/10.1145/2362336.2362340

and optimizing resource utilization at all abstraction layers of the system presents significant challenges. A unified framework is needed to derive, analyze, and validate cross-layer polices/parameters while proving various properties pertaining to energy usage, delays, and so on for any given configuration of policies/parameters as the system evolves dynamically over time.

A holistic approach to understand cross-layer interaction in such systems is essential since policies made at one layer can (sometimes adversely) affect behavior at other layers. Our prior experience (FORGE [Mohapatra et al. 2005], DYNAMO [Mohapatra et al. 2007]) in developing algorithms for managing QoS/energy trade-offs in distributed mobile multimedia applications has given us valuable insights into the issues to be addressed. GRACE [Yuan et al. 2006] also aims to trade off multimedia quality against energy by introducing a hierarchy of global (i.e., coordinating all layers) and internal (i.e., within the individual layers) adaptation. PACE [Lorch 2001] investigated the OS role in processor energy management by inferring task information from user interface events to consider application workloads. While existing work has shown the effectiveness of cross-layer adaptation, many of these efforts try to address the average case behavior without verifiable guarantees on their solutions. Our hypothesis is that a comprehensive design methodology based on a formal reasoning framework can provide an effective basis for tuning mobile embedded systems under a multitude of constraints.

More specifically, policy selection and parameter tuning during system operation requires a procedure to determine if it has a significant impact on timing/QoS/resource provisioning. The analysis of certain properties of the multilayer system enables the development of optimized strategies for composite objectives (e.g., to optimize energy, conserve timing, improve quality). Intricate trade-offs across layers need to be addressed to achieve timing/QoS/resource provisioning. This problem becomes even more complex if we consider that the system and environment may keep evolving, requiring dynamic adaptation. Given a current configuration and a set of changes (e.g., new application/task; parameters for existing tasks such as frame rate/resolution/synchronization; device residual power level; network delay/jitter), we need to support cross-layer adaptation in a quantifiable manner (e.g., bound/sensitivity analysis on the impact of the selected policy).

It is important to have a rigorous semantic model of the system: the resources, the application behaviors, the interactions among layers. Using such a formal model, we can specify and perform reasoning about cross-layer adaptation within a unified framework for integrated treatment combining formal methods, monitoring of system dynamics, and adaptation strategies. The problem we address here is the following.

> How does one decide what policies and parameters to assign to each layer at runtime to minimize the overall energy consumption while providing a sufficient level of QoS with verifiable/quantifiable solution quality? This must be achieved for an energy-constrained mobile embedded device delivering delay-sensitive multimedia data over a lossy network.

In this context, the ability to compensate on-the-fly for property violations at different layers of abstraction is of paramount importance since there are several sources of unpredictability (e.g., delay, packet drop) in a distributed network that introduce nondeterminism. Furthermore, system-level optimizations for effective utilization of distributed resources can interfere with the properties of executing applications. Many applications have flexible QoS needs that dictate how tolerant they are to delays and errors; the lack of stringent timing needs can be adaptively exploited for better end-to-end resource utilization. In Kim et al. [2006], we demonstrated the need for integration of formal methods with experimentally, based cross-layer optimizations to ensure that corner-cases are covered, and we provided a means to determine

bounds for critical performance parameters. Recently, we proposed statistical formal methods for the analysis of given cross-layered optimization policies with quantifiable confidence [Kim et al. 2007c].

To leverage these prior efforts, we propose an iterative tuning approach for mobile multimedia systems that couples two important facets:

(1) a lightweight formal verification that can be used at runtime to evaluate the impact of various policies for achieving cross-layer timing/QoS properties; and
(2) a system realization that enables feedback of additional information on system execution behaviors to enhance our lightweight formal modeling and analysis.

The integration of formal analysis with observed system execution behavior permits adaptation with some notion of guarantees on cross-layer timing/QoS properties for mobile embedded systems that employ resource-constrained devices. In our current approach, we assume that all nondeterminism can be probabilistically quantified, meaning that it can be modeled by probabilistic choice with regard to a suitable distribution (not necessarily uniform), which we will see is possible in our particular case study. Some ideas on how this approach can be generalized to systems that exhibit both nondeterministic and probabilistic choices will be discussed in Section 3.1.

Hence, we propose a unified framework for iterative system tuning to support adaptations. Initially, our framework performs property checking and quantitative analysis of candidate policy/parameter settings via formal executable specifications followed by statistical techniques. Iterative tuning allows model refinement from up-to-date and continuous observations of system execution behavior. Furthermore, this can be used to improve adaptation by verifying given system properties or by relaxing constraints.

This article provides the following contributions.

—We propose a methodology to address iterative system tuning of mobile embedded systems by integrating two synergistic approaches: lightweight formal verification and model refinement based on feedback from a system realization. Lightweight formal verification provides degrees of confidence in the feasible solutions satisfying multidimensional constraints. The system realization enables dynamic adaptation by refining the model of the system and the environment.
—Within the framework, we provide a notion of quantifiable guarantee. We attempt to ensure the quality of selected cross-layer policies/parameters by applying statistical formal methods (e.g., statistical property checking and quantitative analysis) on executable formal models specifying a space of possible behaviors.
—We propose model refinement as a way of reflecting dynamics when the system evolves over time. We use a feedback loop between formal specifications and observed system execution behavior to obtain information on system dynamics under selected policies/parameters in order to improve the formal model.
—Our work is validated and tested via a specific case study with a simulation in the context of mobile multimedia applications, which demonstrates interesting opportunities for tradeoff analysis in highly dynamic changing environments.

2. XTUNE UNIFIED FRAMEWORK FOR ITERATIVE SYSTEM TUNING

Figure 1 illustrates how we position our framework, named *xTune*, within the bigger picture. Here the observation system monitors the current status of target devices and environment. The tuning module decides the strategies that will be deployed for each device. The tuning module may consult the verification engine to ensure its solution quality. The xTune framework supports a methodology for a tuning module that attempts cross-layer adaptation and a verification engine that performs formal analysis for quantification.

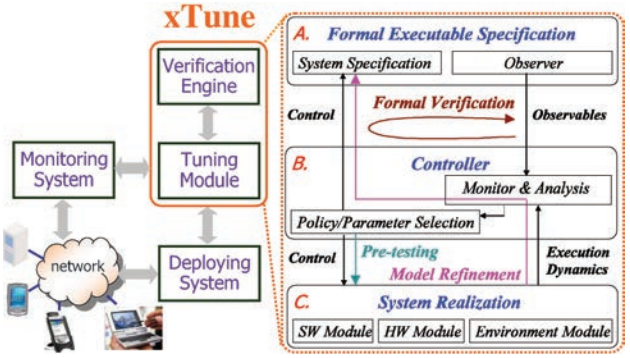


Fig. 1. xTune cross-layer system tuning framework.

2.1. Overview of xTune Framework

The right of Figure 1 presents the overall flow of our approach. Box A represents the formal modeling. The core of our formal modeling approach is to develop formal executable models of system components at each layer of interest. These models express functionality, timing, and other resource considerations at the appropriate level of detail and using appropriate interaction mechanisms (clock ticks, synchronous or asynchronous messages). Models of different layers are analyzed in isolation and composed to form cross-layer specifications. We use the Maude system for developing and analyzing formal specifications (see Section 3.1 for details). One advantage of formal executable models is that they can be subjected to a wide range of formal analysis, including single execution scenarios, search for executions leading to states of interest, and model checking to understand properties of execution paths.

The controller (Box B in Figure 1) performs the evaluation phase of given specifications to generate statistics for properties and values of interest. Specifically, we have developed new analysis techniques (statistical property checking and statistical quantitative analysis) that combine statistical and formal methods, and applied them to a multimedia case study. The main ideas are presented in Section 3.2.

Using such models and analysis, tools can be developed to achieve adaptive refinement of a system specification into appropriate policy/parameter settings. We propose an *iterative* tuning strategy that combines formal methods (verification) with dynamic system execution behavior (obtained by either simulation or implementation). The execution behavior from system realization (Box C in Figure 1) is fed back into the formal modeling to refine the executable specification (*model refinement*). In addition, we can assure the quality of a new policy/parameter constructed by the controller. In Figure 1, *Pretesting* on a system realization can lead to improvements because typically the formal model cannot cover all possible implementation details of a real system. We explain iterative tuning in Section 3.3.

2.2. Applying xTune to a Mobile Multimedia Case Study

Since the design space of mobile real-time embedded systems is extremely large, we pick certain facets of our target system, that is, we narrow down specific applications, layers, policies, and parameters of interest to effectively demonstrate our proof of concept. Multimedia applications operated on battery-powered mobile devices are viewed as one of the key application drivers for next generation distributed systems. Such mobile multimedia applications provide a rich set of QoS/power issues at multiple abstraction levels. Thus, although we intend our approach to be widely applicable, we begin by developing and evaluating formal specification models in the context of mobile

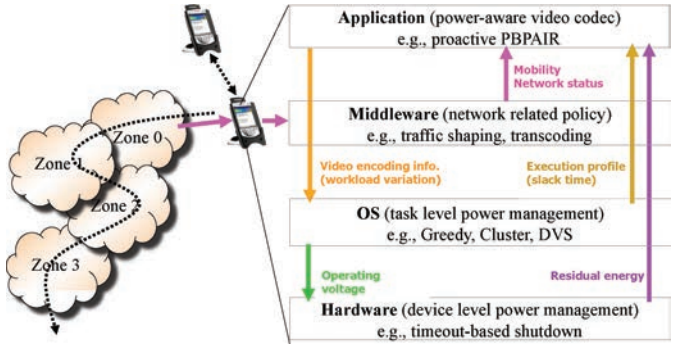


Fig. 2. Layer interaction.

multimedia applications. This type of application requires frequent policy and parameter tuning based on user input and/or node/network conditions (e.g., residual power level, packet drop rate, noise level). As an example, a high-end videophone would be able to better meet its timing constraints at maximum CPU performance while receiving packets via a reliable channel. However, if the residual power level dropped or packet loss rate increased significantly, then we might need to save energy by reducing QoS or suspending some tasks.

2.2.1. Cross-Layer Modeling for Our Target Application. Figure 2 illustrates how policies determine decisions made at different layers: a specific video encoding/decoding algorithm at the application layer, network monitoring at the middleware layer, and DPM (dynamic power management) and/or DVS (dynamic voltage scaling) at the OS layer [Kim and Ha 2001][1]. Network traffic shaping and/or transcoding at the middleware layer can be also utilized. Each policy has parameters that can be used to fine-tune the behavior. In addition, hardware parameters can be set.

For instance, we consider *PBPAIR* (Probability-Based Power-Aware Intra Refresh) [Kim et al. 2006] as an application layer policy. The PBPAIR scheme inserts intracoding (i.e., coding without reference to any other frame) to enhance the robustness of the encoded bitstream at the cost of compression efficiency. Intracoding improves error resilience, but it also contributes to reducing encoding energy consumption since it does not require motion estimation[2] (which is the most power-consuming operation in a predictive video compression algorithm). An additional proactive feature means that we can use a priori information on the user's mobility (e.g., current zone, speed, and trajectory) and network situation (e.g., packet loss rate, delay) that later will be used for selection among policies and related parameter tuning before the user enters a new zone. If PBPAIR is selected as an application layer policy, then algorithm-specific parameters such as the *intra-threshold* value must be chosen for the appropriate execution (see Figure 7(b) for the impact of different *intra-threshold* selection). Note that the parameter selection at one layer affects other layers. For example, PBPAIR increases intracoding by lowering the *intra-threshold* parameter when there is high network

[1]DPM puts a device into a low-power/performance state to save energy when the device is not serving any request during a suitably long time period determined by the shutdown and wake-up overhead of the device. DVS aims at saving energy by scaling down the supply voltage and frequency when the system is not fully loaded.

[2]In predictive coding, motion estimation eliminates the temporal redundancy due to high correlation between consecutive frames by examining the movement of objects in an image sequence to try to obtain vectors representing the estimated motion.

packet loss (monitored at the middleware layer), which impacts the DVS decision at the OS layer since the execution profile of the application is changed.

2.2.2. Specific Policies and Parameters in Our Target Application. We explain which features of our case study will be formally modeled at each layer.

—*Application Layer—Video Codec.* At the application layer, PBPAIR takes the user's QoS expectation, the network packet loss rate, and raw video sequences as inputs to generate a bitstream robustly encoded against network transmission errors. Therefore, our formal specification needs to generate the execution profile (e.g., when does encoding start/end?). Particularly, we specify an encoding workload profile as a distribution function. For example, we model actual execution time by a uniform distribution between best-case execution time (BCET) and worst-case execution time (WCET). We also consider a Gaussian distribution with the average and a boundary value [Im et al. 2004; Lorch 2001].

—*Middleware Layer—Network Monitoring.* This layer deals with the distributed aspects of the system by modeling network status to enable the proactive[3] control. In our model, other nodes are abstracted into the zone information. Zone information includes network delay and packet drop rate within the particular zone. Network delay is modeled as exponential inter-arrival time (Poisson process) with mean. Packet drop due to lossy network is modeled as a uniform distribution with given packet loss rate. Then, we define mobility as a triple (current zone, speed, trajectory) to identify the network situation in the current zone and to anticipate the next zone based on user's speed and trajectory.

—*OS Layer—Power Management.* Various DPM and DVS power management schemes assuming a worst-case scenario are modeled at the OS layer. The OS layer generates slack time information based on workload from the application layer. This slack time is used later to reduce energy consumption while guaranteeing QoS requirements for the next frame. Since we are targeting multitask environments, we also specify various (real-time) scheduling algorithms such as EDF (Earliest Deadline First) and RM (Rate Monotonic).

—*Hardware Layer—Enabling Technology.* To support a DPM and DVS strategy at the OS layer, we assume that the enabling technology (e.g., voltage-scalable processor, power-state controllable network card) is available at the hardware layer. In the case of a microprocessor, wakeup/sleep delay and power overhead for a state transition, DVS characteristics (i.e., power consumption for different operating mode, voltage, or frequency) are modeled. As a result of execution, the hardware layer reports residual energy to upper layers.

3. ITERATIVE TUNING IN XTUNE: COMBINING FORMAL VERIFICATION WITH SYSTEM REALIZATION

Our approach combines the following.

(1) *Modeling, specification and reasoning about cross-layer properties.* We propose a novel approach based on concurrent rewriting logic to formally specify and reason about timing/QoS issues across layers and study their interrelationships.
(2) *Design of policies and mechanisms for addressing trade-offs based on the cross-layer analysis.* Our work examines the impact of various resource management techniques on timing/QoS properties and enables informed selection of resource management policies along with rules for instantiation of parameters of the policies.

[3]Our iterative system-tuning approach is not restricted to proactive systems. The same technique can essentially be applied to reactive systems.

(3) *Model refinement and proactive control.* We enhance our lightweight formal modeling and analysis by integrating it with observations of system execution behavior to achieve adaptive reasoning and proactive control by providing more precise information on current execution and future state.

We begin by articulating the roles of formal methodology for adaptation strategies. First, a formal specification supports a powerful *model of abstraction* at different layers; a formal model is capable of describing QoS needs, exposing uncertainties, and enables systematic analysis of properties and subsequent optimization. Second, a formal framework based on sufficiently well-defined languages with well-defined semantics (functional logic and nonfunctional needs such as timing) enables *reasoning about properties* at different abstraction layers, including multidimensional QoS properties and the relationships among them. Third, statistical formal analysis provides a *quantifiable solution quality* in the cross-layer tuning. The formal specification is analyzed using statistical property checking and statistical quantitative analysis to determine the impact of various resource management policies for achieving desired properties. Last, an iterative tuning process based on *model refinement* enables adaptation and validates the correctness of the adaptation decisions. The formal specification itself can be refined to reflect the system dynamics, which is crucial to adapting cross-layer policies and to predicting the possible property violations as the system evolves dynamically over time.

In the following sections, we explain technical details of our approach: modeling effort toward formal executable specification (Section 3.1), statistical evaluation for verifiable/quantifiable solution (Section 3.2), iterative tuning by model refinement from dynamic execution behavior (Section 3.3).

3.1. Formal Modeling

As our first step, we formally specify the environmental changes as well as the policies/parameter settings that can be made at each of the layers in isolation and for the combined layers. Our formal modeling approach uses rewriting logic [Meseguer 1992], a simple logic well suited for distributed system specification. The state space of a distributed system is formally specified as an algebraic data type by a set of sorts (types), operations, and equations. The dynamics of such a distributed system can then be specified by rewrite rules of the form

$$t \rightarrow t' \ \textbf{if}\ c,$$

where t, t' are terms (patterns) that describe the local, concurrent transitions possible in the system, and c is a condition constraining the application of the rule. Specifically, when a part of the distributed state matches the pattern t and satisfies c, then this part can change to a new local state t'. Rewriting logic specifications are executable, as proofs in rewriting logic are carried out by applying rewrite rules that can also be viewed as steps of a computation.

We use the object-oriented specification style supported by Maude [Clavel et al. 2007]. The Maude system is based on an efficient rewriting engine, supporting the use of executable models as prototypes. It also provides the capability to search the state space reachable from some initial state by the application of rewrite rules. This can be used to find reachable states satisfying a user-defined property.

We develop abstract formal models of the aforementioned application, middleware, OS, and hardware layers as well as the corresponding policies and parameters. Using formal executable specifications in Maude, we model PBPAIR as an application layer policy as well as various power management schemes (Greedy, Cluster, DVS) as OS layer policies. In the *Greedy* scheme, the power manager shuts down whenever the

device is idle, while the *Cluster* scheme tries to aggregate idle periods to maximize energy efficiency. The *DVS* scheme lowers supply voltage as long as the deadline constraint is satisfied. The arrival of incoming processing requests from the network is modeled as a Poisson process with an average arrival rate. When the processor runs at full speed, the execution times of the tasks are modeled as a normal (Gaussian) distribution. It should be noted that our approach is not restricted to a specific distribution. If the characteristics of task execution times, hardware components, or environmental factors such as network status (e.g., transmission time, inter-arrival time) have been changed, we are able to refine the corresponding models by simply modifying the Maude operator for other types of distributions. However, finding the exact formula for such distributions is beyond the scope of this article. Video encoder and decoder tasks for videophone are modeled with the workload variation of a PBPAIR encoder [Kim et al. 2006] and an H.263 decoder [TMN 10 1998]. The network zone information is assumed to be given and the hardware implementation is for the PPC440GP processor[4].

In Maude syntax, the system state (configuration) is represented as a multiset of objects and messages. The objects have the general form

```
< ObjectName : ClassName |  Attribute_1 : Value_1, ... , Attribute_n : Value_n >
```

where *ObjectName* is an object identifier, *ClassName* is a class identifier, and each *Attribute : Value* pair specifies an attribute identifier and its value.

The Maude representation of the application layer using PBPAIR has the following form.

```
< PBPAIR : Application | WCET : worst_case_execution_times, BCET : best_case_execution_times,
                         Intra_Th : 0.73, Qsize : 4, deadlineMiss : 2, lostReq : 1, ... >
```

The preceding object PBPAIR has attributes like *WCET, BCET* for generating the worst-case and best-case workload profiles, respectively. The *Intra_Th* attribute is an algorithmic parameter for the PBPAIR policy. Attributes *deadlineMiss* and *lostReq* contain total number of deadline misses and lost requests due to buffer overflow when the encoding queue size is *Qsize*.

Models of the different layers are composed using multiset union, which is written as juxtaposition. The following represents the composition of the layers modeled.

```
< PBPAIR : Application | lostReq : 1, ... >
< ZoneInfo : Middleware | PacketLossRate : 10, ... >
< PowerManager : OS | ... >
< CPU : Hardware | residualEnergy : 100, ... >
```

The dynamic behavior of concurrent object systems is axiomatized by rewrite rules. For example, the rule

```
rl [wakeup]:
   < CPU : HW | PowerStatus : deepsleep, Timer : 0, Sched : idle, ... >
   =>
   < CPU : HW | PowerStatus : sleep, Timer : WakeupTime(sleep), Sched : transit, ... >.
```

defines a transition where the CPU object updates its attributes when it needs to be woken up from the deep sleep mode with the overhead of *WakeupTime(sleep)* defined as a function in Maude.

[4]We use the PPC440GP processor (http://www.ibm.com/chips/techlib/techlib.nsf/products/Powerpc_440_Embedded_Core) since there exists a corresponding full system simulation model. However, the PPC440GP is not capable of DVS. For the experiments, we use synthetic values of frequency-power pairs as 400MHz-1000mW, 200MHz-292mW, 100MHz-105mW with 20 micro-seconds voltage regulating overhead. These values are based on the characteristics of a similar processor PPC405LP [Zhu and Mueller 2005] that is capable of DVS. We believe the exact formula for the hardware component such as nonlinear scaling of voltage frequency is beyond the scope of this approach. More accurate data can be provided from outside to refine the model.

Passage of time is modeled by functions that update the configuration appropriately, for example, decrementing timers or decreasing remaining power. As proposed in Real-Time Maude [Ölveczky and Meseguer 2007], rules can be either instantaneous or tick rules of the form

$$C \rightarrow delta(C, T)\ in\ time\ T \quad \textbf{if}\ T \leq mte(C),$$

where C is a term representing the system configuration. This tick rule advances time by a time T less than or equal to *mte(C)*, the maximal time allowed to elapse in one step in configuration C, and alters the system state C using the function *delta*. For example, the *delta* function decreases timer value, and the *mte* function specifies the next moment when some instantaneous rule must be applied (e.g., timer expiration).

```
eq delta(< Oid : Cid | Timer : T, ... >, T') = < Oid : Cid | Timer : T - T', ... >.
eq mte(< Oid : Cid | Timer : T, ... >) = T.
```

For the system state, the function *delta* distributes over the objects and messages in a configuration, and the function *mte* computes the maximum time elapsed by taking the minimum over all parts of the configuration [Ölveczky and Meseguer 2007].

```
eq delta(Config_1 Config_2, T) = delta(Config_1, T) delta(Config_2, T).
eq mte(Config_1 Config_2) = min(mte(Config_1), mte(Config_2)).
```

To execute a certain scenario, we use the *rewrite* command in Maude with a random seed and given setting of policies/parameters. For example,

```
rew init(seed, dvs, th, battery, t, ...).
```

describes an initial configuration generated by the *init* function to examine the DVS policy and PBPAIR policy with an *Intra_Th* parameter. The residual energy is set to *battery*, and the *rewrite* command executes up to time t. A random seed is used to generate various distributions to replace all nondeterminism with probabilistic choices and time advances using a stochastic model (e.g., exponential distribution with rate) in the tick rule. Nondeterminism that is not probabilistic in nature would require the exhaustive exploration of all possibilities and is currently not supported in our approach. Hence, we use sufficient conditions similar to those of Agha et al. [2006] to guarantee the absence of this form of nondeterminism.

At the end of each execution, we examine the final configuration of a Maude specification that has several objects and messages. We refer to this execution as a sample trace generation in the following discussion. From those objects and messages, we need to extract meaningful data, observables. *Observables* can be properties or values. For example, to check whether or not the battery expires at the end of the execution, we need to check the *residualEnergy* attribute in CPU object at the hardware layer. If the value for the *residualEnergy* attribute is positive, then the battery is not empty. Otherwise, the *batteryEmpty* property returns *true*, meaning that the system used up the battery. We encode the check of properties into the model so that the result contains true or false, depending on whether or not a property holds. On the other hand, if we want to have the energy consumption rather than the boolean value of the property, we can utilize an observer such as the one shown in Figure 3. The observer replaces each object with suitable messages that have values for the observables. For example, *deadlineMiss* and *lostReq* in PBPAIR object are observables for this kind.

3.2. Statistical Evaluation

Once we extract observables from runs of a formal executable specification, the controller performs formal verification as illustrated in Figure 1. In particular, we perform Monte Carlo simulation by the rewrite command in Maude that generates one possible behavior of the system, starting from a given initial state using a user-specified seed

```
*** Property checker
  op batteryEmpty : Configuration $->$ Bool .
  eq batteryEmpty(< CPU : HW | residualEnergy : F, atts > C:Configuration)
  = (if (F <= 0.0) then true else false fi) .
*** Observer
  msg Obs : Bool -> Msg .
  msg EnergyConsumption : Float -> Msg .
  msg BatteryEmpty : Bool -> Msg .
  rl [cpuObs] :
      < CPU : Hardware | consumedEnergy : F, policy : P, atts >
      =>
      EnergyConsumption(F)
      BatteryEmpty(batteryEmpty(< CPU : HW | consumedEnergy : F, atts >)) .
```

Fig. 3. Maude specification: property checker and observer.

for sampling from distributions. The sample traces from Monte Carlo simulation are used for the subsequent statistical techniques that we explain in this subsection.

To evaluate feasible design points, we adapt and improve two statistical evaluation methods: statistical property checking and statistical quantitative analysis. For statistical property checking, probabilistic properties such as

> Probability that a system can survive with given residual energy for t time units is more than a threshold (θ %).

will be checked. In the case of statistical quantitative analysis, we estimate the expected value of certain observables such as

> Average energy consumption for t time units within a confidence interval (δ) and an error bound (α).

3.2.1. Statistical Theory Background and Implementation. The analysis of stochastic systems is typically carried out using either numerical or statistical techniques [Younes et al. 2006]. To solve probabilistic model checking problems, one can attempt to compute the probability measure of a set of paths using numerical techniques, but this is infeasible for systems with complex dynamics or large state spaces. Therefore, we use *statistical* techniques.

Statistical Property Checking. We use statistical property checking, a form of hypothesis testing based on Monte Carlo simulation results, to verify probabilistic properties. In hypothesis testing, we test whether the probability p of a property under examination is above or below the threshold θ. We can formulate this as the problem of testing the hypothesis $H : p \geq \theta$ against the alternative hypothesis $K : p < \theta$. We implemented two statistical property checking techniques in our framework: *sequential* testing [Wald 1945] and *black-box* testing [Sen et al. 2004]. Sequential testing generates sample execution paths until its answer can be guaranteed to be correct within the required error bounds. Black-box testing instead computes a quantitative measure of confidence for the given samples. Here, *black-box* means that the system cannot be controlled to generate execution traces, or trajectories, on demand starting from arbitrary states. The implementation of sequential testing and black-box testing can be found as part of the Ymer [Younes et al. 2006] and VeStA [Sen et al. 2004] tools, respectively.

Statistical Quantitative Analysis. Statistical evaluation can be performed with a large quantity of data that follows a normal distribution, and hence allows the estimation of the expected value and a confidence measure. To ensure the mathematical soundness of the approximation, we perform a Jarque-Bera (JB) normality test [Jarque and Bera 1987]. The normality is determined by testing the null hypothesis (that the sample comes from a normal distribution with unknown mean and variance) against the alternative (that it does not come from a normal distribution). The JB test computes the p-value (the smallest level of significance at which a null hypothesis may be

rejected) from the JB test statistic and the χ^2 (chi-square) distribution. One rejects the null hypothesis if the p-value is smaller than or equal to the significance level. With 5% significance level, a p-value $= 0.05$ indicates that the probability of getting a value of test statistics as extreme as or more extreme than that observed is at most 5% if the null hypothesis is actually true. Once normality of the data is ensured with high confidence for a large enough number of sample traces n, the approximate average falls inside a (1 - α)100% confidence interval $(\bar{x} - Z_{\frac{\alpha}{2}} \frac{s}{\sqrt{n}}, \bar{x} + Z_{\frac{\alpha}{2}} \frac{s}{\sqrt{n}})$, where $\bar{x}$ is the average of the sample variables, s is the samples' standard deviation, and $Z_{\frac{\alpha}{2}}$ is a *standard score* (also called *Z-score* or *normal score*) of the normal distribution [Hogg and Craig 1995]. To obtain the desired confidence, we want the size of this (1 - α)100% confidence interval to be less than or equal to δ, that is, $2Z_{\frac{\alpha}{2}} \frac{s}{\sqrt{n}} \leq \delta$, where α and δ are given.

3.2.2. Our Approach: Simplified Formulae and On-demand Sample Generation.

Statistical Property Checking. We note that both the Ymer and VeStA tools target complex properties of stochastic systems. For instance, those tools take properties specified in a temporal logic, namely, Continuous Stochastic Logic (CSL) [Aziz et al. 1996; Baier et al. 1999], for Continuous Time Markov Chain (CTMC) system specifications. The reason is that they want to support complex property checking (e.g., nested temporal/probabilistic operators, and also a form of hybrid model checking in between numerical and statistical methods). This can be overkill when it comes to analyzing practical optimization problems if we test only simple properties such as "Probability that a system can survive with a given residual energy for t time units is more than θ %". Those formulae are essentially a restricted version of CSL without nesting. Indeed, we found no need for nested formulae or an exact numerical solution for our application domain. Hence, we use hypothesis testing based on Monte Carlo simulation, which does not need to be tied to any particular specification model or temporal logic.

ALGORITHM 3.1: Statistical Quantitative Analysis

Inputs: error bound α, confidence interval δ,
observable under consideration
Output: Expected value E[observable]
Procedure:

```
do {
    trace generation until normality test succeeds;
    d = 2Z_{α/2} s/√n;
} while (d > δ);
return the average of observable;
```

Statistical Quantitative Analysis. Algorithm 3.1 shows pseudocode for the statistical quantitative analysis. As we mentioned earlier, to approximate the expected value by the mean of n samples such that the size of the (1 - α)100% confidence interval is bounded by δ, the sample data should follow a normal distribution [Hogg and Craig 1995]. For the normality test, we need to have a sufficiently large dataset. Since the trace generation takes most of the evaluation time, we generate sample traces only if more samples are required (i.e., JB test cannot accept the normality of data.). We also use the bootstrap distribution (e.g., resampling by the average of the randomly chosen original samples with replacement) [McCabe and Moore 2005] for the normality test and the subsequent analysis, which leads to further evaluation time reduction without loss of accuracy in a statistical sense. By generating traces on demand, we can

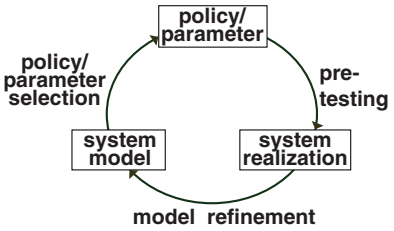


Fig. 4. Iterative tuning cycle.

significantly reduce the evaluation time since it is proportional to the trace generation time.

We implement these two statistical formal techniques, statistical property checking and statistical quantitative analysis, in our framework to formally verify the given policies/parameters and to enable informed selection of them. The experimental results in Section 5.1 from both theoretical analysis and Monte Carlo simulation followed by statistical methods demonstrate the applicability of our approach to the design of mobile embedded systems.

3.3. Iterative Tuning

Modeling with formal executable specifications, rather than implementing simulators of distributed systems under consideration, enables us to carry out formal analysis (e.g., statistical property checking and quantitative analysis). However, there exist opportunities at runtime to improve the formal model to better represent the system dynamics. For this purpose, we perform model refinement using observed system execution behavior by equipping the controller with a feedback loop to interact with the system realization. Note that a system realization can be implemented with a simulator, an emulator, or a real implementation. In our case, we used a realistic simulator as described in Section 4.1. As depicted in Figure 1, the controller interacts with system realization to obtain information on dynamic system behavior under selected policies/parameters in order to improve the formal model. We call this pretesting.

3.3.1. Model Refinement. Within our framework, feedback from the observation of system execution behavior can be used to improve the model (to make it more accurately match the real environment), and hence it can be used to indirectly improve the policy. As illustrated in Figure 4, our iterative tuning cycle consists of three phases: (i) policy/parameter selection, (ii) pre-testing, and (iii) model refinement. For instance, the formal specification initially models the execution times of the tasks as a normal (Gaussian) distribution with the average of $\frac{(BCET+WCET)}{2}$ and the boundary value of 3δ, where δ represents the standard deviation, based on profiled BCET and WCET from sample runs. This model is refined, in turn, by replacing BCET and WCET by observations from dynamic system execution in order to more realistically reflect the actual executions characterizing the system in practice.

Figure 5 illustrates model refinement based on the dynamic system execution behavior from a system realization. Using the formal specification, the controller performs verification/evaluation of the given policies based on the initial model ($phase_1$ in Figure 5). Statistical quantitative analysis is performed on Maude traces up to time t_1 to determine the best policies/parameters. These are used to configure the system realization ($event_2$ in Figure 5). Since obtaining execution behavior from the system realization usually takes a much longer time than formal analysis (e.g., in our case, it is of the order of hundreds of times slower than formal analysis), it is beneficial to find the best policy by initial formal analysis first. At time t_2, the system realization starts

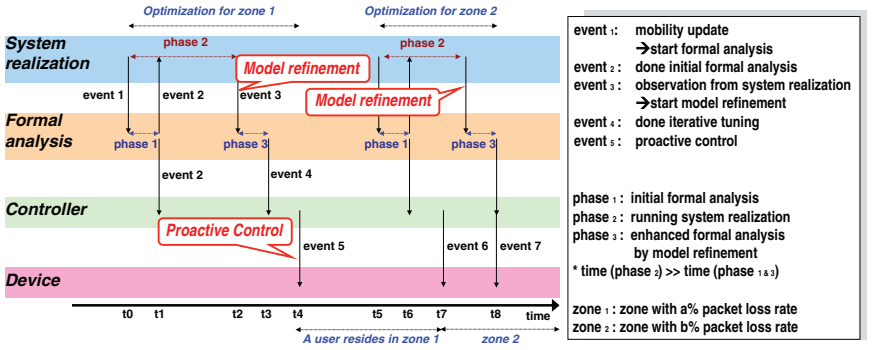


Fig. 5. Model refinement and proactive control.

generation of the BCET and WCET that reflect the actual executions as described by $event_3$ in Figure 5. Then, the formal model is refined by updating BCET and WCET to enhance the analysis results as shown between t_2 and t_3 ($phase_3$ in Figure 5).

3.3.2. Proactive Control. As mentioned in Section 2.2, we exploit given information on a user's mobility (e.g., current zone, speed, and trajectory) and network situation (e.g., packet loss rate, delay) to select among policies and related parameters before the user enters a new zone. The mobility information is used to identify the network situation in the current zone and to anticipate the next zone based on a user's speed and trajectory. Currently, we assume that the next zone information is forecast by the system realization ($event_1$). As mentioned in Section 4.1, prediction of future trends in network traffic based on user mobility information from a device is beyond the scope of this article.

Figure 5 also illustrates proactive control initiated by network status update. At time t_0, the middleware layer is informed about the next zone information that a user will reach, $zone_1$ with packet loss rate $a\%$ at time t_4. Our framework performs the iterative tuning process, formal execution followed by statistical analysis ($phase_1$) with subsequent model refinement ($phase_2$ and $phase_3$), for the next zone. As a result, our framework can generate proactive controls ($event_5$) to the device before the user enters the new zone (any time between t_3 and t_4). Similarly, at time t_5, the formal model is informed that a user will be in a zone with packet loss rate $b\%$ at time t_7. In the case of $zone_2$, the extended analysis results ($event_7$) are available at time t_8 and can be immediately forwarded to the device. Therefore, by time t_7 (when a user enters $zone_2$), our framework delivers $event_6$ from the initial formal analysis to the device as shown in Figure 5.

4. SYSTEM IMPLEMENTATION

As explained in Sections 2.1 and 3, the integration of formal analysis with a system realization (as illustrated in Figure 1) results in a feedback loop that includes the formal models, simulation, and monitoring of running systems for analysis of the system behavior and for optimizing the choice of policies and parameters. Figure 6(a) illustrates our evaluation platform that is composed of the (1) formal executable specification, (2) controller, and (3) system realization, which we describe in the following text.

4.1. System Realization

The system realization takes policies/parameters and returns the dynamic system execution behavior at each layer as seen in Figure 6(b). For instance, if the controller selects PBPAIR (with appropriate *Intra_Th* parameter) as the application layer policy and DVS as the OS layer policy, the system realization executes using the appropriate

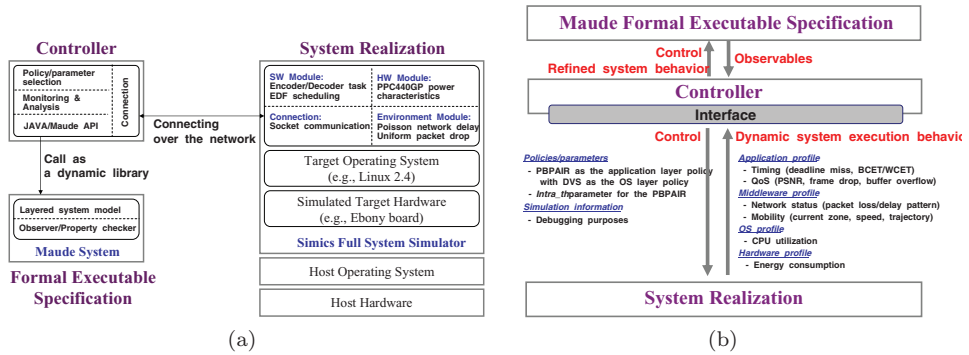


Fig. 6. (a) Evaluation platform, (b) Interactions between controller and system realization.

settings and reports profiled information like consumed energy and timing/QoS aspects, illustrated as pretesting and model refinement in Figure 1.

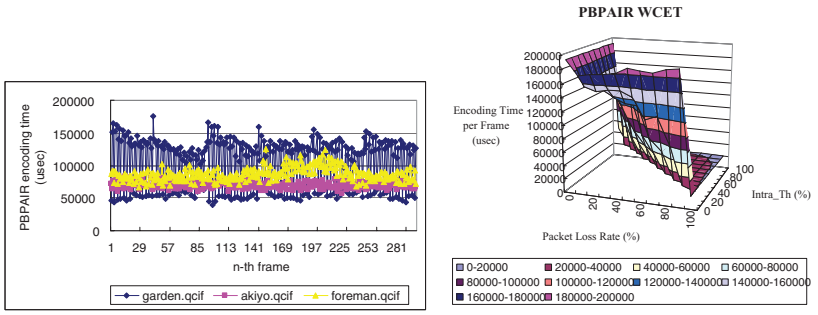
For this purpose, we need a full system simulation, meaning that our system realization should have an implementation for a target system at the level that an operating system acts with arbitrary workloads. Thus, we define a collection of library routines and their arguments that can be used to implement a system realization. The library routines allow users to create and manage applications on top of a mobile device while controlling network status, OS scheduling, and power management schemes.

As explained in Section 3.3, the execution profile from the system realization is fed back into the formal model to enhance the solution quality. We use the Simics full system simulation platform,[5] capable of simulating target systems that include a real network connection and run operating systems and workloads. Specifically, we use the Simics model of a PowerPC-based Ebony board[6] with a PPC440GP processor that boots Linux 2.4 as in Figure 6(a). On top of the Simics target OS, we extend the system realization to accommodate the SW, HW, and Environment modules as we describe in the following text. In our case study, we model two tasks (video encoder/decoder) within an energy-constrained device. The network status is modeled separately.

—*SW Module.* A task set is created at the application layer. As an example, a videophone has two tasks; video encoder/decoder with its own parameters (e.g., PBPAIR has the *Intra_Th* parameter). Also, the input/output data structure is task specific. For instance, an H.263 encoder with PBPAIR policy takes the *Intra_Th* parameter, the network packet loss rate, and raw video sequences as inputs and generates a bitstream robustly encoded against network transmission errors. In addition, there are encoder QoS-related parameters (e.g., quantization value, IP ratio, frame rate, buffer size). At the OS layer, the module includes OS-level power management schemes (e.g., DPM, DVS) and an EDF scheduling algorithm for energy-efficient real-time multitasking. As a result, application profile information such as QoS (peak signal to noise ratio (PSNR), frame drops) and timing (deadline misses, BCET, WCET) aspects are reported.

—*HW Module.* The HW module simulates the target hardware platform including its power states and characteristics of a voltage-scalable processor associated with each state. For example, the power consumption of mobile SoC (system-on-a-chip) processors for *idle* state is in the order of milliwatts, and transition overhead is less than 0.1 microsecond [Olsen and Narayanaswami 2003]. The transition overhead is

[5]http://www.simics.net.
[6]http://www.amcc.com/Embeded.



(a) (b)

Fig. 7. PBPAIR Timing Information from System Realization (a) Frame Encoding Time for Different Video Clips (Intra_Th = 0.73, Packet Loss = 10%), (b) Worst-Case Encoding Time Per Frame Varying Intra_Th under Different Packet Loss Rate (FOREMAN.QCIF 300 Frames).

represented by the minimum idle time of a device that compensates for state transition (or frequency change) overhead. This device-dependent parameter is typically referred to as the *break-even time* (T_{be}). As a result of execution, the energy profile is reported.

—*Environment Module.* As briefly mentioned, we define mobility as a triple (current zone, speed, trajectory) to identify the network situation in the current zone and to anticipate the next zone based on the user's speed and trajectory. Ideally, we need prediction techniques like time series analysis [Han and Venkatasubramanian 2001] to forecast the future trends in network traffic with some defined level of confidence. This is, however, beyond the scope of this article. Currently, our system realization randomly generates the next zone information such as network delay and packet drop rate within the particular zone.

The two most important observations from our system realization are timing (BCET, WCET) and network-related information. The framework uses timing information to refine the model and network-related information to generate proactive control. Therefore, in the experiments (Section 5.2), we demonstrate how the framework can achieve iterative system tuning for proactive control using those two pieces of feedback from system execution behavior.

5. EXPERIMENTAL RESULTS

To demonstrate the applicability of our framework to QoS/energy trade-off management, we explore several aspects of the system optimization. Our formal executable specification (Maude) and evaluation method can serve as a simulation study as well as a statistical guarantee for the design. The outcome of the formal analysis helps us determine the right blend of policies and parameter settings that enable better QoS and better energy efficiency. Our framework can guide more informed resource management in the context of both static instantiation and dynamic tuning of system policy/parameter. In the following sections, we present our experimental results on each aspect, beginning with static analysis.

5.1. Policy/Parameter Instantiation: Static Analysis

To evaluate the effect of cross-layer optimization, first we need to quantify the impact of the optimization at each layer and their composition. Currently, we model energy optimization policies (e.g., DVS, DPM) to reduce energy consumption while satisfying the QoS requirements even in the worst case scenario. However, as shown

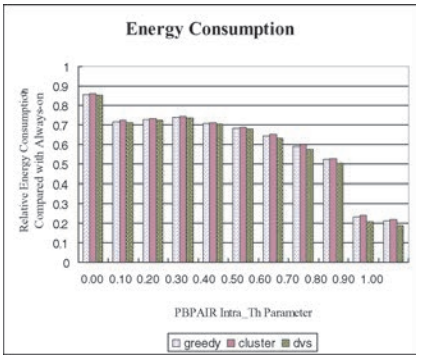


Fig. 8. Effect of cross-layer optimization: Energy perspective on varying OS layer policy and application policy/parameter (packet loss rate = 10%, buffer size = 2 frames).

in Figure 7(a) there is data dependency in the encoding time of each frame as well as significant variation from frame-to-frame. Typical multimedia applications finish execution much earlier than the worst-case execution time in most of the situations as shown in Figure 7(a), which leads to slack time for the next frame with proper buffering. Figure 7(b) shows that the worst-case execution time per-frame of PBPAIR encoding for the same video sequence depends on the network status as represented by the packet loss rate and on the parameter setting of the algorithm.

Interestingly, the PBPAIR parameter can be manipulated to cover large trade-off space as shown in Figure 7(b): from *Intra_Th = 1* (meaning a user wants to encode whole frames as intracoding for maximum error resilience) to *Intra_Th = 0* (indicating that a user wants to encode with maximum compression efficiency without considering any error resiliency). Without any packet loss, *Intra_Th* has no impact as shown in Figure 7(b). Through Figure 7, we are able to observe that PBPAIR can coordinate with the DPM or DVS policy by controlling its parameter based on the trade-off between the error resilience level and encoding energy consumption. In the following sections, we explain experimental results that illustrate the effect of this cross-layer optimization.

5.1.1. Monte Carlo Simulation. Figure 8 presents the energy profiles according to the different policies and their parameter selection with given packet loss rate (10%) and buffer size (two frames). Recall that *Intra_Th = 0* indicates DVS without PBPAIR policy. DVS with PBPAIR outperforms other policies from the perspective of relative energy consumption with respect to *Always-on* (i.e., without any policy). However, the effect of energy reduction incurred from the application layer parameter setting is much higher than from the OS layer policy selection, as shown by the decreasing trend in Figure 8. On another note, energy consumption needs to be carefully considered with QoS aspects. For example, Cluster consumes more energy than other policies in Figure 8, and yet it performs better in QoS aspects (see the case with buffer size equals 2 in Figures 9(a) and 9(c)).

QoS measures such as average deadline miss ratio and average frame drop ratio are also examined. Figure 9(a) shows that PBPAIR combined with any OS layer policy delivers more timely decoding than any OS layer policy without PBPAIR. As the buffering capacity grows, however, the decoder deadline miss ratio is not continuously decreasing as we expect. This can be interpreted with Figure 9(c). The decoder drops fewer video frames with a larger buffer. As a result, more frames in the buffer may lead to overload. Proper buffer size estimation and/or buffering techniques (e.g., skip decoding/encoding some frame when buffer overflow occurs) can balance between frame drops (due to limited buffering capacity) and deadline misses (timing violation due to system overload)

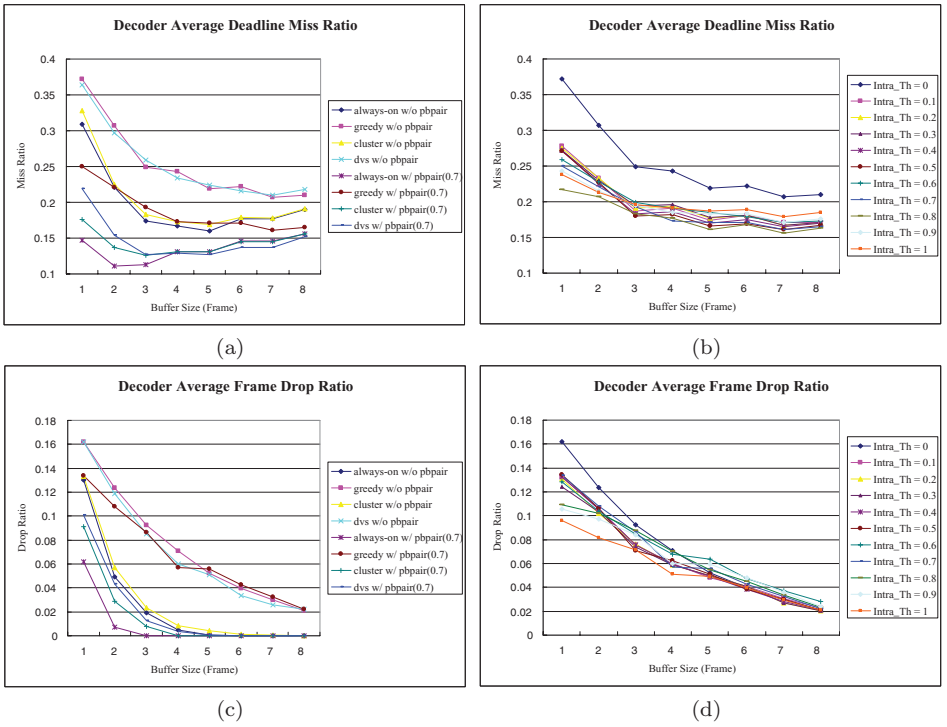


Fig. 9. Effect of cross-layer optimization: QoS perspective (packet loss rate = 10%) on varying (a), (c) policy and buffer size ($Intra\_Th = 0.7$); (b), (d) $Intra\_Th$ and buffer size (greedy policy).

[Qiu et al. 2001; Wu et al. 2009]. This is, however, beyond the scope of this article. In Figure 9(c), Cluster policy outperforms DVS and Greedy from the perspective of average frame drops because Cluster policy attempts to finish decoding/encoding for buffered frames as soon as possible. The effect of parameter setting within the same policy selection (e.g., Greedy) is presented in Figures 9(b) and 9(d). It should be noted that the comparative evaluations such as Figures 8 and 9 can guide informed tuning of a system. In Figure 9(b), with $Intra\_Th$, parameter 1 and larger buffer sizes possibly miss more deadlines since more intracoding could lead to higher decoding energy consumption to the point where encoding energy reduction cannot compensate for it. More detailed experimental results on QoS aspects can be found in Kim et al. [2007b].

Note that runtime is linearly proportional to the single trace generation time (i.e., Maude rewriting time from initial state). If we consider the rate of 50 frames each (5 frames per-second) for both encoding and decoding, single trace generation takes around 400 to 500ms on a 2.8GHz Pentium 4 processor running Linux. Therefore, it is infeasible to produce all possible traces to evaluate policy and parameter changes in dynamic situations. This led us to use statistical approaches with quantifiable confidence for our evaluation/decision.

5.1.2. Statistical Property Checking. As explained in Section 3.2, we implemented suitable forms of sequential testing and black-box testing for our purposes. Statistical property checking enables quick detection of problematic situations (e.g., battery expiration) that can arise due to the selection of policy/parameter settings. As an example of sequential testing, we checked the formula

$$\mathcal{P}_{<\theta}\ [batteryEmpty]$$

```
1: decoderAvgConsecutiveLost =
2:   [nTrace=100] Reject Ho (PValue=0.042)
3:   [nTrace=110] Fail to reject Ho (PValue=0.702)
4:   [nTrace=110] (d = 0.01094) > (delta = 0.01)
5:   [nTrace=121] (d = 0.01077) > (delta = 0.01)
6:   [nTrace=133] (d = 0.01059) > (delta = 0.01)
7:   [nTrace=146] (d = 0.01020) > (delta = 0.01)
8:   [nTrace=160] (d = 0.00997) <= (delta = 0.01)
9: Expected value = 1.56061 with error bound 5.0%, confidence interval 0.997%
```

Fig. 10. Statistical quantitative analysis (H_0: null hypothesis on data normality, δ: confidence interval, α: error bound).

by testing if the probability p that the path formula *batteryEmpty* holds is less than the threshold θ. We use arguments $\alpha = 0.05$ (false negative), $\beta = 0.05$ (false positive), $\theta = 0.1$ (threshold), and $\delta = 0.01$ (indifference region), respectively. Sequential testing accepts the hypothesis $H_1 : p \ll \theta - \delta$ with 133 traces, that is, the *batteryEmpty* property checker in Figure 3 gives *false* for all traces. With the same 133 traces that were generated for sequential testing, black-box testing also confirms the formula with an error of 8.20E-7. Further explanation for sequential testing [Wald 1945] and black-box testing [Sen et al. 2004] can be found in our technical report [Kim et al. 2007b]. The runtime for either method of statistical property checking is 10 to 20ms (on a 2.8GHz Pentium 4 processor running Linux) in addition to the sample generation, which indicates that this is a feasible proposition for the on-the-fly adaptation.

5.1.3. Statistical Quantitative Analysis. Figure 10 shows our statistical quantitative analysis results with arguments of α (error bound) and δ (confidence interval) as 5% and 1%, respectively. In Figure 10, the observable *decoderAvgConsecutiveLost* can not pass the normality test with 100 initial samples since its p-value (0.042) is lower than significance level (0.05). Hence, we need to generate more samples (10% in this experiment) as shown in *line 3*. Even if the sample data follows a normal distribution, we may need more samples for limiting the confidence interval by δ. Line 4–8 in Figure 10 present such a case. Finally, the resulting confidence interval d (0.997%) is less than the desired value δ (1%).

The aforementioned static analysis based on a statistical approach allows an informed selection of policy/parameter when a user resides in the same zone.

5.2. Policy/Parameter Tuning: Dynamic Analysis

As depicted in Figure 1, the controller interacts with the system realization to pretest selected policies/parameters and to obtain information on dynamic system behavior for iterative tuning of policies/parameters. Here, we explain the experimental results on iterative tuning by model refinement (Section 3.3.1) and proactive control (Section 3.3.2).

Let us take an example. At time t_0, the formal specification models PBPAIR execution with [BCET, WCET] as [109ms, 202ms][7], and provides the analysis results for four different policy/parameter selections (A, B, C, D) to encode FOREMAN.QCIF video clip (S_1 to S_4 in Figure 11). Since our system realization reports dynamic execution of PBPAIR as shown in Figure 7, we can refine [BCET, WCET] to be [66ms, 125ms], leading to formal analysis results S_5 to S_8 in Figure 11. Furthermore, we adjust the

[7]These profiled values are from Kim et al. [2006] for various video inputs. We also use 0.73 as *Intra_Th* to generate a similar compression ratio with other error-resilient coding schemes when the packet error rate is 10%. In Kim et al. [2006], the amount of possible energy reduction affected by communication overhead is very small compared to that of the encoding tasks running on a PDA, since the wakeup time of the network interface on the PDA is not short enough to apply DPM for real-time multimedia applications. Increased length of bitstreams due to intracoding may result in higher communication overhead on some devices, which is not modeled in our current specification. However, we believe that our model can be extended to cope with other devices without affecting the core approach.

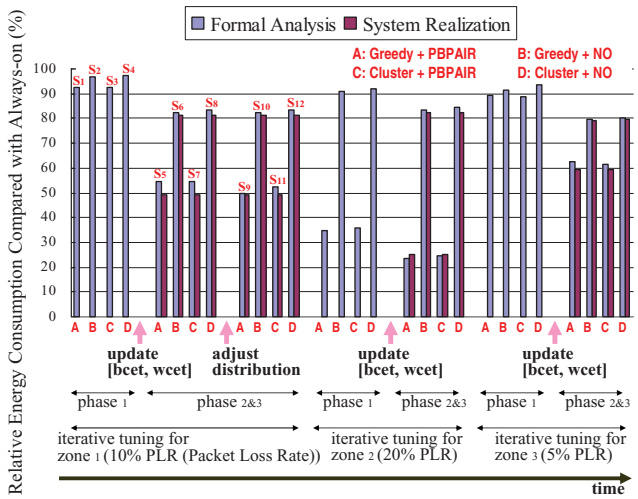


Fig. 11. Dynamic analysis (buffer size = 2 frames, $Intra\_Th = 0.73$).

parameter of the frame encoding time distribution model in the formal specification since many frame encoding times are close to BCET as shown in Figure 7(a). Instead of simply providing simulated [BCET, WCET] as the parameter of the normal distribution model explained in Section 3.1, we use the actual average execution time observed from the system realization. Formal analysis results S_9 to S_{12} in Figure 11 show a better approximation (i.e., closer to the estimation based on dynamic execution behavior from the system realization) due to this adjustment.

It should be noted that $phase_1$ (initial formal analysis) and $phase_3$ (enhanced formal analysis) take much less time than $phase_2$ (running a system realization). The goodness of a policy/parameter selection, however, remains the same through $phase_1$ to $phase_3$. This indicates that lightweight formal verification at runtime can be effectively used in adaptation by rapidly narrowing down the search space of potential policies and parameters. Furthermore, the quality of adaptation can be improved by combining formal analysis with observed system execution behavior.

6. PREVIOUS AND RELATED WORK

Previous work on probabilistic model checking for stochastic systems includes PRISM [Kwiatkowska et al. 2005], SMART [Siminiceanu and Ciardo 2007], MRMC [Katoen et al. 2009], and RAPTURE [Jeannet et al. 2002]. Jansen et al. [2007] present comprehensive experimental results on runtimes and memory consumptions of different probabilistic model checkers including MRMC, PRISM, Ymer [Younes et al. 2006], and VeStA [Sen et al. 2004]. Given a formal description of a system (e.g., variants of Markov chains for PRISM, Petri Nets for SMART), those tools generate the state space, verify temporal logic properties, and provide efficient numerical solutions for stochastic analysis. A quantitative approach with detailed models of the system components (e.g., kinetic battery model) and a tailored numerical algorithm to compute the battery lifetime distributions has been studied [Cloth and Haverkort 2008]. Unlike these methods, we focus on statistical, simulation-based methods and black-box testing only aiming at approximate but sufficiently good results.

Ymer [Younes et al. 2006] implements such a statistical technique, based on discrete event simulation and sequential acceptance sampling for CSL model checking. The system is modeled by CTMC or generalized semi-Markov processes (GSMPs).

Properties are expressed using CSL. Ymer also integrates numerical techniques to solve nested CSL queries by including the hybrid engine of the PRISM tool for CTMC model checking. A more recent study on statistical model checking using perfect simulation [Rabih and Pekergin 2009] proposes the combination of perfect sampling using Markov Chain Monte Carlo methods and statistical hypothesis testing. Perfect sampling could also be combined with our approach to deal with steady-state properties. In summary, the expressive power of the Maude language (extended with probability and time) enables the concise specification of a wide spectrum of applications that are beyond the scope of Markovian models and thanks to its logical basis facilitates the use of a variety of formal methods, e.g., for model validation.

Kumar et al. [2003] and Agha et al. [2006] introduce PMaude (Probabilistic Maude), a rewriting-based specification language for modeling probabilistic concurrent and distributed systems. The associated tool, VeStA [Sen et al. 2004], was developed to statistically analyze various quantitative aspects of models such as those specified in PMaude using a query language QuaTEx [Agha et al. 2006] based on CSL. However, this approach does not provide any procedure to determine the sample size required to achieve normality. Moreover, the authors approximate the expected average by applying *Student's T*-distribution. As the sample size n grows, the T-distribution approaches the normal distribution. Therefore, we extended their approach by an on-demand sample generation that can compute the sample size sufficient to guarantee the normality of data and utilize the normal distribution to obtain the error bound and confidence interval.

Probabilistic model checking (PMC) is used in Norman et al. [2005] to analyze the problem of DPM. The authors obtain the optimal DPM policy by formulating the optimization problem as a discrete time Markov chain (DTMC) model and solve it using an equation solver. Once a policy has been constructed, its performance is validated using the PMC tool PRISM. Even though PMC enables one to experiment with the effectiveness of a selected DPM algorithm in a quantitative way, the challenge of determining how to actually implement a good power manager that considers complex system dynamics still remains, since their work is essentially a validation process for a specific policy determined using an equation solver. Their analysis of stochastic systems is carried out using numerical solution techniques that are far more memory intensive. On the other hand, our approach is to start with an executable formal model specifying a space of possible behaviors and analyze these possible behaviors using lightweight statistical techniques.

Model-predictive control approaches also aim to address power management issues. Abdelwahed et al. [2004] propose predictive learning to shutdown a device by exhaustively searching over a limited look-ahead horizon. In Lu et al. [2002], a closed-loop feedback control based on queuing theory is presented to optimize CPU frequency. Their solution quality depends on the future events forecasted by a mathematical model (e.g., a filter). The applicability of these approaches, however, is limited to systems having a small number of control inputs with synthetic workloads. A control-based middleware framework for QoS adaptations [Li and Nahrstedt 1999] has been proposed for fair resource sharing in a visual tracking application. However, this work focused on the analysis of the adaptation process itself rather than optimizing the overall system utility factor.

Acquaviva et al. [2004] propose an incremental methodology to analyze the effect of a simple time-out-based DPM scheme from a functional model without timing and performs a noninterference check for behavior. Then, the authors extend it to a Markovian model and the effect of DPM is evaluated by standard numerical techniques. Last, they extend it to a general model by using profiled information from real-world measurements and simulate it to compare the result with that of the Markovian model. Our

framework can be seen as a generalization of their work. First, since we use Maude formal executable specifications that can have any distribution in timing by controlling the tick rule, our formal model corresponds to their general model. Their mathematical soundness is guaranteed only when the model follows an exponential timing distribution. More importantly, our primary focus is online adaptation based on abstract formal models complemented by a system realization, not the validation at design time.

7. CONCLUSIONS

We have presented a unified framework, xTune, to develop formal analytical methods for understanding cross-layer optimization issues in highly distributed systems that incorporate resource-limited devices, and we have shown how to integrate these methods into the design and adaptation processes for such systems. We propose iterative system tuning for mobile embedded systems and apply it in a case study treating multimedia communication. The integration of formal analysis with the observation of dynamic system execution for analysis of system behavior and optimizing choice of policies and parameters results in a feedback loop that includes the formal models, simulation, and monitoring of running systems. On another note, this article substantially complements our previous work on experimentally based cross-layer strategies Forge Project[8] and conclusively shows that xTune provides a uniform methodology for deriving, analyzing, and validating cross-layer adaptation.

As a next phase of this project, we investigated the impact of compositions between application and OS layers [Kim et al. 2008]. This effort aims to provide a policy/parameter selection mechanism by coordinating among local optimizers through *constraint refinement*. As an example, if the application layer optimizer refines its parameter, the OS layer optimizer refines its parameter by taking the application layer parameter ranges as constraints. Then the OS layer results are again transmitted to the application layer optimizer for further refinement. Thus, the constraint language serves as a common interface among different local optimizers, leading to improvements of solution quality, robustness, and speed of convergence. Compositional optimization through constraint refinement enables a controller to coordinate existing local optimizers, which can accommodate different objectives by treating them as black boxes.

The formal rewriting logic specification underlying our case study satisfies the conditions of Agha et al. [2006], which guarantee the absence of nondeterminism. An interesting question is if our approach can be generalized to support uncertainty in probabilities/parameters and true nondeterministic choices, a more dynamic form of uncertainty. Formally, the case of uncertain probabilities/parameters can be treated by using a parameterized rewriting logic specification along the lines of a probabilistic specification in the sense of Jonsson and Larsen [1991], which constrains probabilities to a set (typically an interval). Runtime monitoring can help to narrow down the choice of such parameters. True nondeterminism, which is arbitrarily resolved by the environment/adversary at system runtime, would, in the general case, require the exhaustive exploration of all possibilities. An interpretation of a rewriting logic specification as a Markov decision process (instead of a Markov chain) would provide a possible formal setting. State space exploration is supported by the Maude engine, but due to the resulting state space explosion the degree of nondeterminism that can be handled in this way is limited.

The problem of model checking in the presence of uncertainties and nondeterminism has been studied on the basis of interval-valued discrete-time Markov chains [Sen et al. 2006], but model checking for an expressive logic such as probabilistic computational

[8]http://forge.ics.uci.edu.

tree logic (PCTL) might be unnecessary and too expensive for runtime use. A more practical way to deal with both uncertain parameters and nondeterminism is to perform the probabilistic analysis under a specific subset of possible environments/adversaries, for example, to explore best/worst-case behavior. Clearly, the justification for utilizing such a subset needs to be established independently based on the formal specification. Tools support for this process would be another interesting direction for future work.

REFERENCES

Abdelwahed, S., Kandasamy, N., and Neema, S. 2004. Online control for self-management in computing systems. In *Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'04)*. 368.

Acquaviva, A., Aldini, A., Bernardo, M., Bogliolo, A., Bonta, E., and Lattanzi, E. 2004. Assessing the impact of dynamic power management on the functionality and the performance of battery-powered appliances. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'04)*. 731.

Agha, G. A., Meseguer, J., and Sen, K. 2006. PMaude: Rewrite-based specification language for probabilistic object systems. *Electr. Notes Theor. Comput. Sci. 153,* 2, 213–239.

Aziz, A., Sanwal, K., Singhal, V., and Brayton, R. K. 1996. Verifying continuous-time Markov chains. In *Proceedings of the 8th International Conference on Computer Aided Verification (CAV'96)*. 269–276.

Baier, C., Katoen, J.-P., and Hermanns, H. 1999. Approximate symbolic model checking of continuous-time Markov chains. In *Proceedings of the International Conference on Concurrency Theory*. 146–161.

Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., and Talcott, C. 2007. All about Maude, a high-performance logical framework. In Lecture Notes in Computer Science, vol. 4350, Springer, Berlin.

Cloth, L. and Haverkort, B. R. 2008. Quantitative evaluation in embedded system design: Predicting battery lifetime in mobile devices. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'08)*. ACM, New York, NY, 90–91.

Han, Q. and Venkatasubramanian, N. 2001. AutoSeC: An integrated middleware framework for dynamic service brokering. *IEEE Distrib. Syst. Online 2,* 7.

Hogg, R. and Craig, A. 1995. *Introduction to Mathematical Statistics* 5th Ed. Pearson.

Im, C., Ha, S., and Kim, H. 2004. Dynamic voltage scheduling with buffers in low-power multimedia applications. *Trans. Embedd. Comput. Syst. 3,* 4, 686–705.

Jansen, D. N., Katoen, J.-P., Oldenkamp, M., Stoelinga, M., and Zapreev, I. S. 2007. How fast and fat is your probabilistic model checker? An experimental performance comparison. In *Proceedings of the Haifa Verification Conference*. Lecture Notes in Computer Science, vol. 4899, Springer, 69–85.

Jarque, C. and Bera, A. 1987. A test for normality of observations and regression residuals. *Int. Statist. Rev. 55,* 2, 163–172.

Jeannet, B., D'Argenio, P. R., and Larsen, K. G. 2002. RAPTURE: A tool for verifying Markov decision processes. In *Proceedings of the International Conference on Concurrency Theory*.

Jonsson, B. and Larsen, K. G. 1991. Specification and refinement of probabilistic processes. In *Proceedings of the Logic in Computer Science Symposium (LICS)*. IEEE Computer Society, 266–277.

Katoen, J.-P., Zapreev, I. S., Hahn, E. M., Hermanns, H., and Jansen, D. N. 2009. The ins and outs of the probabilistic model checker MRMC. In *Quantitative Evaluation of Systems (QEST)*, IEEE Computer Society, 167–176. www.mrmc-tool.org.

Kim, M., Dutt, N., and Venkatasubramanian, N. 2006. Policy construction and validation for energy minimization in cross layered systems: A formal method approach. In *Proceedings of RTAS'06* (WiP Session). 25–28.

Kim, M. and Ha, S. 2001. Hybrid run-time power management technique for real-time embedded system with voltage scalable processor. In *Proceedings of the ACM Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*. ACM, 11–19.

Kim, M., Oh, H., Dutt, N., Nicolau, A., and Venkatasubramanian, N. 2006. PBPAIR: An energy-efficient error-resilient encoding using probability based power aware intra refresh. *ACM SIGMOBILE Mob. Comput. Comm. Rev. 10,* 3, 58–69.

Kim, M., Stehr, M.-O., Talcott, C., Dutt, N., and Venkatasubramanian, N. 2007a. Combining formal verification with observed system execution behavior to tune system parameters. In *Proceedings of FORMATS'07*. Lecture Notes in Computer Science, vol. 4763, 257–273.

Kim, M., Stehr, M.-O., Talcott, C., Dutt, N., and Venkatasubramanian, N. 2007b. Modeling and exploiting cross-layer optimization in distributed embedded systems. Tech. rep. SRI-CSL-07-02, SRI International.

KIM, M., STEHR, M.-O., TALCOTT, C., DUTT, N., AND VENKATASUBRAMANIAN, N. 2007c. A probabilistic formal analysis approach to cross layer optimization in distributed embedded systems. In *Proceedings of FMOODS'07*. Lecture Notes in Computer Science, vol. 4468, 285–300.

KIM, M., STEHR, M.-O., TALCOTT, C., DUTT, N., AND VENKATASUBRAMANIAN, N. 2008. Constraint refinement for online verifiable cross-layer system adaptation. In *Proceedings of the Design, Automation and Test in Europe Conference and Exposition (DATE'08)*.

KUMAR, N., SEN, K., MESEGUER, J., AND AGHA, G. 2003. A rewriting based model for probabilistic distributed object systems. In *Proceedigs of FMOODS'03*. Lecture Notes in Computer Science, vol. 2884, 32–46.

KWIATKOWSKA, M., NORMAN, G., AND PARKER, D. 2005. Quantitative analysis with the probabilistic model checker PRISM. *Electr. Notes Theor. Comp. Sci. 153,* 2, 5–31.

LI, B. AND NAHRSTEDT, K. 1999. A control-based middleware framework for quality of service adaptations. *IEEE J. Select. Areas Comm. 17,* 9, 1632–1650.

LORCH, J. R. 2001. Operating systems techniques for reducing processor energy consumption. Ph.D. thesis, University of California, Berkeley.

LU, Z., HEIN, J., HUMPHREY, M., STAN, M., LACH, J., AND SKADRON, K. 2002. Control-theoretic dynamic frequency and voltage scaling for multimedia workloads. In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*. 156–163.

MCCABE, G. P. AND MOORE, D. S. 2005. *Introduction to the Practice of Statistics* 5th Ed. W.H. Freeman.

MESEGUER, J. 1992. Conditional Rewriting Logic as a unified model of concurrency. *Theor. Comput. Sci. 96,* 1, 73–155.

MOHAPATRA, S., CORNEA, R., OH, H., LEE, K., KIM, M., DUTT, N. D., GUPTA, R., NICOLAU, A., SHUKLA, S. K., AND VENKATASUBRAMANIAN, N. 2005. A cross-layer approach for power-performance optimization in distributed mobile systems. In *Proceedings of IPDPS'05*.

MOHAPATRA, S., DUTT, N., NICOLAU, A., AND VENKATASUBRAMANIAN, N. 2007. DYNAMO: A cross-layer framework for end-to-end QoS and energy optimization in mobile handheld devices. *IEEE J. Select. Areas Comm. 25,* 4, 722–737.

NORMAN, G., PARKER, D., KWIATKOWSKA, M., SHUKLA, S., AND GUPTA, R. 2005. Using probabilistic model checking for dynamic power management. *Formal Aspects Comput. 17,* 2, 160–176.

OLSEN, C. M. AND NARAYANASWAMI, C. 2003. A work dependent OS timing scheme for power management: Implementation in Linux and modeling of energy savings. Internet draft, IBM.

ÖLVECZKY, P. C. AND MESEGUER, J. 2007. Semantics and pragmatics of real-time Maude. *Higher-Order Symbol. Comput. 20,* 1-2, 161–196.

QIU, Q., WU, Q., AND PEDRAM, M. 2001. Dynamic power management in a mobile multimedia system with guaranteed quality-of-service. In *Proceedings of DAC'01*. 834–839.

RABIH, D. E. AND PEKERGIN, N. 2009. Statistical model checking using perfect simulation. In *Proceedings of ATVA*. Lecture Notes in Computer Science, vol. 5799, Springer, 120–134.

SEN, K., VISWANATHAN, M., AND AGHA, G. 2004. Statistical model checking of black-box probabilistic systems. In *Proceedings of CAV'04*. 202–215.

SEN, K., VISWANATHAN, M., AND AGHA, G. 2006. Model-checking Markov chains in the presence of uncertainties. In *Proceedings of TACAS*. Lecture Notes in Computer Science, vol. 3920, Springer, 394–410.

SIMINICEANU, R. I. AND CIARDO, G. 2007. Formal verification of the NASA runway safety monitor. *Int. J. Softw. Tools Technol. Transf. 9,* 1, 63–76.

TMN 10. 1998. TMN 10 (H.263+), ver. 3.2.0. Image Process. Laboratory, University of British Columbia.

WALD, A. 1945. Sequential tests of statistical hypotheses. *Annals Mathem. Statis. 16,* 2, 117–186.

WU, C.-C., CHEN, K.-T., CHANG, Y.-C., AND LEI, C.-L. 2009. An empirical evaluation of VoIP playout buffer dimensioning in Skype, Google talk, and MSN messenger. In *Proceedings of NOSSDAV'09*.

YOUNES, H., KWIATKOWSKA, M., NORMAN, G., AND PARKER, D. 2006. Numerical vs. statistical probabilistic model checking. *Int. J. Softw. Tools Technol. Transf. 8,* 3, 216–228.

YUAN, W., NAHRSTEDT, K., ADVE, S. V., JONES, D. L., AND KRAVETS, R. H. 2006. Grace-1: Cross-layer adaptation for multimedia quality and battery energy. *IEEE Trans. Mobile Comput. 5,* 7, 799–815.

ZHU, Y. AND MUELLER, F. 2005. Feedback EDF scheduling exploiting hardware-assisted asynchronous dynamic voltage scaling. In *Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*. 203–212.

Received February 2010; revised August 2010; accepted November 2010