12-2009

Timing Analysis of Mixed Time/Event-Triggered Multi-Mode Systems

Linh T.X. Phan
University of Pennsylvania, linhphan@seas.upenn.edu

Samarjit Chakraborty
Institute for Real-Time Computer Systems, TU Munich, Germany, samarjit@tum.de

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

Recommended Citation

Linh T.X. Phan, Samarjit Chakraborty, and Insup Lee, "Timing Analysis of Mixed Time/Event-Triggered Multi-Mode Systems", *30th IEEE Real-Time Systems Symposium (RTSS 2009)* , 271-280. December 2009. http://dx.doi.org/10.1109/RTSS.2009.24

Timing Analysis of Mixed Time/Event-Triggered Multi-Mode Systems

Abstract
Many embedded systems operate in multiple modes, where mode switches can be both time- as well as event-triggered. While timing and schedulability analysis of the system when it is operating in a single mode has been well studied, it is always difficult to piece together the results from different modes in order to deduce the timing properties of a multi-mode system. As a result, often certain restrictive assumptions are made, e.g., restricting the time instants at which mode changes might occur. The problem becomes more complex when both time- and event-triggered mode changes are allowed. Further, for complex systems that cannot be described by traditional periodic/sporadic event models (i.e., where event streams are more complex/bursty) modeling multiple modes is largely an open problem. In this paper we propose a model and associated analysis techniques to describe embedded systems that process multiple bursty/complex event/data streams and in which mode changes are both timeand event-triggered. Compared to previous studies, our model is very general and can capture a wide variety of real-life systems. Our analysis techniques can be used to determine different performance metrics, such as the maximum fill-levels of different buffers and the delays suffered by the streams being processed by the system. The main novelty in our analysis lies in how we piece together results from the different modes in order to obtain performance metrics for the full system. Towards this, we propose both – exact, but computationally expensive, as well as safe approximation techniques. The utility of our model and analysis has been illustrated using a detailed smart-phone case study.

Timing Analysis of Mixed Time/Event-Triggered Multi-Mode Systems

Linh T.X. Phan[1] Samarjit Chakraborty[2] Insup Lee[1]
[1]Department of Computer and Information Science, University of Pennsylvania, USA
[2]Institute for Real-Time Computer Systems, TU Munich, Germany
e-mail: {linhphan, lee}@seas.upenn.edu, samarjit@tum.de

***Abstract*—Many embedded systems operate in multiple modes, where mode switches can be both time- as well as event-triggered. While timing and schedulability analysis of the system when it is operating in a single mode has been well studied, it is always difficult to piece together the results from different modes in order to deduce the timing properties of a *multi-mode* system. As a result, often certain restrictive assumptions are made, e.g., restricting the time instants at which mode changes might occur. The problem becomes more complex when both time- and event-triggered mode changes are allowed. Further, for complex systems that cannot be described by traditional periodic/sporadic event models (i.e., where event streams are more complex/bursty) modeling multiple modes is largely an open problem. In this paper we propose a model and associated analysis techniques to describe embedded systems that process multiple bursty/complex event/data streams and in which mode changes are both time- and event-triggered. Compared to previous studies, our model is very general and can capture a wide variety of real-life systems. Our analysis techniques can be used to determine different performance metrics, such as the maximum fill-levels of different buffers and the delays suffered by the streams being processed by the system. The main novelty in our analysis lies in how we piece together results from the different modes in order to obtain performance metrics for the full system. Towards this, we propose both – exact, but computationally expensive, as well as safe approximation techniques. The utility of our model and analysis has been illustrated using a detailed smart-phone case study.**

I. Introduction

The increasing complexity and costs of modern embedded systems require them to operate in multiple *modes*, where each mode – among other things – may be characterized by a different set of tasks, different data arrival rates, and a different scheduling policy. Mode switches may be both time- as well as event-triggered. Examples of the former consist of servicing time-triggered interrupts, while the latter might be mode switches triggered by events like an incoming call in a mobile phone, or a buffer in the system filling up beyond a certain level.

Modeling and analyzing such systems have therefore been a topic of great interest within the real-time and embedded systems community. Here, the main challenge is to compose timing/performance analysis results from individual modes in order to derive the properties of the overall system. In order to simplify this problem, often some restrictive assumptions are made. For example, in the time-triggered language Giotto [1], it is required that tasks which are interrupted by mode switches should have the same activation rates in both the source as well as the target modes. A result of this restriction is that timing constraints of the overall system are guaranteed if the system is feasible while operating in the individual modes. Hence, it is sufficient to verify feasibility of the individual modes – which is a much easier and well-studied problem – and modes changes need not be explicitly accounted for while performing timing analysis.

The analysis becomes substantially more complex if both time- *and* event-triggered mode changes are allowed. Further, if the system in question is to be described using complex event/data models – i.e., those beyond classical periodic/sporadic event models – then modeling and analyzing mode changes for such systems is currently a largely unexplored problem.

Our contributions: In this paper we propose a model for describing multi-mode systems which process multiple complex/bursty event/data streams, and in which mode changes may be both time- as well as event-triggered. A wide variety of embedded devices have such characteristics. For example, consider a smart-phone which can be used to play streaming audio/video, as well as make and receive phone calls. Depending on the operating mode of the system – i.e., whether an incoming call was received while a video application was running – the priorities of the different tasks may be changed, or certain tasks might be put to sleep or even terminated.

Given the burstiness in the arrival patterns of audio/video data, as well as the high variability in their execution requirements, it is often overly pessimistic to use standard periodic/sporadic event models. Hence, mode changes in such systems need to be modeled in conjunction with more general event models that are better suited to capture bursty event/data streams. Towards this, we rely on the Real-Time Calculus (RTC) framework to describe – in a flexible manner – the arrival pattern and processing requirements of the different streams to be processed by the system. The RTC framework was introduced in [2], [3] and subsequently extended in a number of other papers (e.g., see [4], [5]). It relies on a *count-based abstraction* to model the timing properties of the input streams, as well as the availability of the resources, and is more general than classical event/service models.

The systems we study are assumed to consist of multiple tasks which get triggered by incoming data/event streams. The arrival patterns of these streams, as well as their processing requirements – as mentioned above – are described using the

RTC framework (details of which follow later in the paper). The arriving streams (which are waiting to be processed) are stored in input buffers and processed streams in output buffers that are read out by output devices. Each *mode* in the system is defined by the set of active tasks, the arrival rates of the streams triggering these tasks, and the scheduling policy according to which these tasks are served. Mode changes, as already mentioned, can be both time- as well as event-triggered. For example, certain tasks are activated after pre-defined time periods, while the task priorities might change (hence a mode change) depending on the fill-levels of the different buffers in the system. In this setting, we ask questions of the form: what is the maximum fill level of a given buffer, what is the delay suffered by an individual stream, what is the delay suffered by a stream in a particular mode, etc.

While such questions have been answered before within the RTC framework (e.g., see [2]) for unimodal systems, the known techniques do not extend to the case of multi-mode systems. The main novelty of the analysis techniques that we propose in this paper lies in how we piece together the results from individual modes to derive timing/performance properties of the entire system. Towards this we combine analysis techniques from the RTC framework with state-space exploration methods that are used to analyze state-based models such as timed automata. Besides computing delay and buffer fill-level metrics exactly – which turn out to be computationally expensive – we also propose safe approximations of these metrics in out setting. Finally, we illustrate our model and analysis techniques using a realistic smart-phone case study, where we show that explicitly modeling the multiple modes (rather than approximating the system behavior as a unimodal system) results in tighter estimates of the different performance metrics.

Related work: There has been a number of previous attempts to extend models and timing analysis techniques from the real-time systems literature to accommodate more complex behaviors. For example, the framework presented in [6] allows certain tasks to intentionally change their execution periods, which is a type of mode change. The associated scheduling technique then adapts the periods of the other tasks within allowable limits in order to maintain a schedulable system. Similarly, the model proposed in [7] allows a system to be in multiple modes, where each mode consists of a set of tasks possibly overlapping with other modes. The system uses Rate Monotonic scheduling for all the modes. The problem is then to select suitable parameters for all the tasks, such that the system is schedulable in all modes.

Different *mode change protocols* have been studied in [8], [9] and have been classified in [10]. Here, again, a system consists of multiple modes, where each mode consists of a set of tasks. A mode change is triggered by a mode change request (MCR), and transitions from an old to a new mode take non-zero time. During this transition, the system has tasks from both the old and the new modes, which might produce a temporal overload. However, MCRs cannot arrive during a transition period. The goal is to develop techniques that ensure that no deadlines are violated during the transition periods. Such techniques consist of suitable mode change *protocols* (e.g., to restrict mode changes only at pre-specified time instants, or allow only synchronous mode changes), as well as analysis techniques to *verify* the feasibility of the system in the different modes and during the transition periods.

In contrast to the above approaches, we make the simplistic assumption of instantaneous mode changes. However, we have a richer model for specifying task activation patterns at the different modes (using the RTC framework). Moreover, – unlike the above studies – we also accommodate mixed time- and event-triggered mode changes and our analysis seamlessly handles both. While mixed time- and event-triggered embedded systems have been studied in the past (see, e.g., [11] and [12]) they have either been in the context of bus protocols like FlexRay or in the context of synchronous programming languages (integrating urgent events that require preemption in a synchronous programming environment). We, on the other hand, focus on timing and performance analysis of multi-mode systems where mode changes can be time- and event-triggered (rather than formulating the mode change protocols themselves).

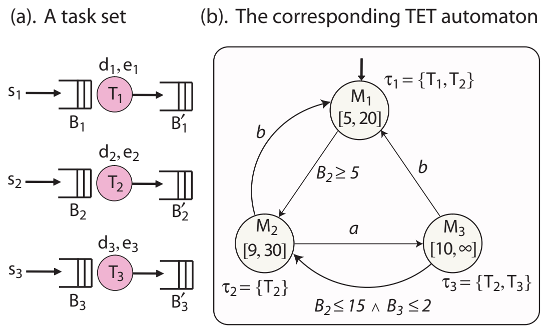
Finally, it may be noted that we recently proposed a *multi-mode* extension to the RTC framework [13], which is a different model compared to what we study in this paper. In multi-mode RTC, arrival patterns of streams and availability patterns of resources are modeled as automata, whose states are annotated with functions denoting arrival/service rates. The arrival and service automata in [13] aim at modeling (independently) complex arrival patterns of event streams and resource availability patterns; however, they follow a relatively simple processing semantics. Here, we are interested in the system as a whole, thereby having a single model for the system does not only reduce the analysis complexity but also provides a better intuition to the system's behavior. The concept of "mode" in this current paper is a more natural one, viz. operating modes of a system. While the arrival/service patterns considered here are simpler than the ones studied in [13], we allow a richer processing semantics with different tasks and scheduling policies explicitly captured and adapted to the dynamic characteristics of event streams and resources.

Organization of the paper: In the next section we describe our model, followed by our analysis techniques in Sections III and IV. Finally, we present a smart-phone case study in Section V and conclude with a discussion on possible extensions of this work. Due to space restrictions, the proofs for all lemmas may be found in [14].

II. The System Descriptions

A. Basic models

The system consists of a finite set of tasks $\mathcal{T} = \{T_1, \ldots, T_n\}$ that are mapped on the same processor, where $n \in \mathbb{N}$, $n \geq 1$. Each task T_i processes an input data stream s_i, item by item, in a first-come first-served basis. Upon arriving at the system, incoming data items from s_i are stored at an input buffer B_i before being processed by T_i, and the processed data items are then written to an output buffer B'_i. Fig. 1(a) shows a system consisting of three tasks $\mathcal{T} = \{T_1, T_2, T_3\}$, with each T_i associated with an input buffer B_i and an output buffer B'_i. The backlog (fill-level) of a buffer B_i at time t, denoted by $B_i(t)$, is the number of items in the buffer at time t. We shall refer to B_i as the name of the buffer as well as the variable whose value gives the current fill-level of the buffer.



$M_1 = \langle \beta_1, \tau_1, \text{TDMA}, 5, \{(\alpha_{11}, 3), (\alpha_{12}, 2)\}\rangle$
$M_2 = \langle \beta_2, \tau_2, \text{-}, 0, \{(\alpha_{22}, 1)\}\rangle$
$M_3 = \langle \beta_3, \tau_3, \text{FP}, 0, \{(\alpha_{32}, 2), (\alpha_{33}, 1)\}\rangle$

Fig. 1: Example of a task set and its TET automaton.

Task models. We assume a data-driven dispatch model for the tasks, where an instance of a task is immediately triggered when a data item arrives at the corresponding input buffer. Each task $T_i \in \mathcal{T}$ has a fixed *relative deadline* of d_i time units and a fixed maximum *execution demand* of e_i processor cycles. In Fig. 1(a), d_i and e_i are the relative deadline and execution demand of T_i, respectively, $\forall 1 \le i \le 3$.

Data streams modeled as arrival functions. An arrival pattern of a data stream is specified as a cumulative function $A(t)$ which gives the number of items arrived in the interval $[0,t)$ The set of all arrival patterns of a stream is captured by an *arrival function* $\alpha = (\alpha^u, \alpha^l)$, where $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$ specify the maximum and minimum *number of data items* that can arrive from the stream in any time interval of length Δ. In other words, $A(t)$ is an arrival pattern of a data stream modeled by α (or simply, of α) iff

$$\alpha^l(\Delta) \le A(\Delta + t) - A(t) \le \alpha^u(\Delta),\ \forall \Delta,\ t \ge 0.$$

Resource availability modeled as service functions. A service pattern of the processor is captured by a cumulative function $C(t)$, with $C(t)$ denoting the number of processor cycles available in the time interval $[0,t)$. The set of all service patterns of the processor is modeled by a *service function* $\beta = (\beta^u, \beta^l)$, where $\beta^u(\Delta)$ and $\beta^l(\Delta)$ give the maximum and minimum *number of processor cycles available* in any time interval of length Δ. Thus, $C(t)$ is a service pattern of β iff

$$\beta^l(\Delta) \le C(\Delta + t) - C(t) \le \beta^u(\Delta),\ \forall \Delta,\ t \ge 0.$$

Execution semantics modeled as TET automata. The execution semantics of a system is described by a multi-mode mixed time- and event-triggered (TET) automaton. A TET automaton is a finite automaton whose states represent different operating modes of the system and transitions represent mode changes. Each mode (state) of the automaton is of the form $\langle \beta, \tau, SP, c, \{(\alpha_i, c_i) \mid T_i \in \tau\}\rangle$, which comprises

- a service function β that bounds the total resource available when the system is in this mode;
- a set of tasks $\tau \subseteq \mathcal{T}$ that are executed at the mode;
- a scheduling policy SP that is used to schedule the tasks in τ. When τ contains only one task and SP is Fixed Priority, we omit SP since the system allocates all its available resource to this task.
- a length c for each TDMA cycle when $SP = \text{TDMA}$, where $c \ge \sum_{T_i \in \tau} c_i$ (and if $SP \ne \text{TDMA}$, c is unused);
- a pair (α_i, c_i) for each $T_i \in \tau$, where α_i is the arrival function of the input stream s_i and c_i denotes either (i) the length of the slot allocated to T_i in each TDMA cycle if $SP = \text{TDMA}$, or (ii) the priority of T_i if $SP = \text{FP}$.

Note that the specification of the scheduling parameters in a mode can be adapted to the scheduling policy used. In our model, arrival and service functions remain constant when the system is in a mode. However, *when the system moves to a new mode, they are reset to the new values associated with the new mode*. Before formally defining TET automata, we first state some relevant notations:

- $INT = \{[a_1, a_2] \mid 0 \le a_1 \le a_2 \wedge a_1, a_2 \in \mathbb{N}\}$.
- $\Phi_{\mathcal{B}}$: the set of all buffer guards φ of the form
$$\varphi = a_1 \le B_i \le a_2 \ \mid\ B_i \ge a_1 \ \mid\ \varphi_1 \wedge \varphi_2$$
where $B_i \in \mathcal{B}$, $[a_1, a_2] \in INT$, and $\varphi_1, \varphi_2 \in \Phi_{\mathcal{B}}$. Since B_i only takes integer values, $a_1 \le B_i \le a_2$ and $B_i \in [a_1, a_2]$ are equivalent and thus used interchangeably.
- φ_i: the guard on the buffer B_i that appears in φ.
- $\varphi_i^{\max}$: the maximum value of B_i that satisfies φ. (If φ_i contains no guards of the form $a_1 \le B_i \le a_2$, $\varphi_i^{\max} = \infty$).
- $\varphi^{\max} = \{\varphi_1^{\max}, \ldots, \varphi_n^{\max}\}$.

Definition 1 (TET Automata). *Given a finite set of tasks $\mathcal{T} = \{T_1, \ldots, T_n\}$ and its associated sets of input streams $S = \{s_1, \ldots, s_n\}$, input buffers $\mathcal{B} = \{B_1, \ldots, B_n\}$, and output buffers $\mathcal{B}' = \{B'_1, \ldots, B'_n\}$. The multi-mode TET automaton that executes $\mathcal{T}$ is a tuple $\mathcal{A} = (\mathcal{M}, M_{in}, Inv, \Sigma, R)$ where*

- *$\mathcal{M} = \{M_1, \ldots, M_m\}$ is the set of modes, with $M_j = \langle \beta_j, \tau_j, SP_j, c_j, \{(\alpha_{j,i}, c_{j,i}) \mid T_i \in \tau_j\}\rangle$, $\tau_j \subseteq \mathcal{T}$, $j = \overline{1,m}$.*
- *$M_{in} \in \mathcal{M}$ is the initial mode of $\mathcal{A}$.*
- *$Inv : \mathcal{M} \to INT$ is the mode invariant function, where $Inv(M_j) = [L_j, U_j]$ specifies the interval during which $\mathcal{A}$ can stay at M_j (i.e., it must stay at M_j for at least L_j and for no more than U_j time units). After U_j time units staying at M_j, the automaton must take an enabled transition to another mode, or it will go to deadlock.*
- *Σ is the set of signals that trigger the mode changes, which can be controlled by an external controller.*
- *$R \subseteq \mathcal{M} \times \Sigma \times \Phi_{\mathcal{B}} \times \mathbb{N} \times \mathcal{M}$ is the transition relation. Each transition in R is of the form (M, a, φ, D, M') where (i) M is an origin mode and M' is a destination mode, (ii) a is an external signal that triggers the transition, (iii) φ is a guard on the fill-levels of the input buffers, which must be satisfied for the transition to be enabled, and (iv) $D \in Inv(M)$ is the time at which the transition is enabled (relative to the instant the automaton enters M).*

We assume that if there are more than one enabled out-going transitions from a mode at the same time, the automaton non-deterministically selects one. All transitions are *not urgent*, unless otherwise specified. When there is a transition that is enabled, we say there is a *mode change request*.

As an example, Fig. 1(b) sketches a TET automaton corresponding to the task set given in Fig. 1(a). In the figure, *a* and *b* denote the external signals that trigger the transitions of the automaton. Table I details the service function, the scheduling policy, the active tasks and their corresponding arrival functions in each mode of the automaton.

	Mode M_1	Mode M_2	Mode M_3
Service	β_1	β_2	β_3
Scheduling	TDMA	-	FP
T_1	$\alpha_{1,1}$ slot = 3		
T_2	$\alpha_{1,2}$ slot = 2	$\alpha_{2,2}$	$\alpha_{3,2}$ prio = 2
T_3			$\alpha_{3,3}$ prio = 1

TABLE I: Mode characteristics for the system in Fig. 1.

Mode change semantics. At the instant when there is a mode change request, the processor may be executing some task. There may also be pending data items in the input buffers which are waiting to be processed. In general, there are various ways how a system could response. In this paper, we assume that when there is a mode change request to a new mode M_j:

1) *The automaton enters M_j instantaneously.*
2) *The arrival functions of the input streams, the service function of the resource, and the scheduling policy are reset (to the ones associated with M_j) immediately.*
3) *New items of the streams processed by the tasks in τ_j can arrive immediately after mode switching and their arrival patterns follow the updated arrival functions.*
4) *The system will continue executing the unfinished task (if any) before scheduling the tasks in τ_j. Note that this unfinished task may or may not appear in τ_j.*
5) *All pending data items in the input buffer of a task $T_k \notin \tau_j$ will be delayed until the system moves to a mode that contains T_k.*
6) *No new data items arrive from the data streams that are not processed by τ_j.*

The rationale behind the above assumptions is that in a streaming environment, the currently processed streams are often paused as the system services more critical tasks and resumed at some point later. It is hence important to maintain the buffer state when the system moves to a new mode. Further, our protocols are more general than the ones where all pending data of a task in the old mode but not in the new mode will also be processed. This can be achieved in our model by including the task into the new mode while setting its arrival function to be zero. Protocols where there is some delay D between the arrival of a new task can also be represented in our model by adding an intermediate mode between the two modes and associating with it the same set of tasks as that of the old mode besides a delay D on the transition from this intermediate mode to the new mode.

Analysis problems. Given a system $Sys = \langle \mathcal{T}, \mathcal{B}, \mathcal{B}', S, \mathcal{A} \rangle$. We would like to compute:

P1 The maximum backlog of a buffer $B_i \in \mathcal{B}$.
P2 The maximum delay experienced by a stream $s_i \in S$.
P3 The maximum delay of any data item arriving at a mode.

Schedulabiliy of a task set can be derived from the computed maximum delays, i.e., T_i meets its deadline iff the maximum delay of s_i is always less than or equal to d_i.

In this paper, we consider specifically Fixed Priority (FP) and Time Division Multiple Access (TDMA); however, the same technique can be applied to other scheduling policies. The main difference would be in the computation of the execution demand at each node with respect to the chosen scheduling policy. The details of FP and TDMA scheduling are explained in [14].

III. Exact Timing Analysis

We first define the concept of execution traces of $\mathcal{A}$. Observe that when the automaton is in a mode, each stream arrives at a different rate which is controlled by the corresponding arrival function. We call $A = (A_1, \ldots, A_n)$ an *arrival pattern of a mode* $M = \left\langle \beta, \tau, SP, c, \left\{ (\alpha_i, c_i) \mid T_i \in \tau \right\} \right\rangle$ iff A_i is an arrival pattern of α_i if $T_i \in \tau$, and $A_i = 0$, otherwise. Since there is only one service function associated with M, C is a *service pattern of mode M* iff it is a service pattern of β.

Definition 2 (Buffer Mapping)**.** *A buffer mapping function Buf of $\mathcal{A}$ is a function that computes the fill-levels of the buffers at time $t+\Delta$ based on the fill-levels of the buffers at time t when the system stays put at a mode. Specifically, define $B_i(t)$ to be the fill-level of B_i after t time units the automaton spent at $M \in \mathcal{M}$. Denote $B(t) = \left\{ B_1(t), \ldots, B_n(t) \right\}$. Suppose A is an arrival pattern of M, C is a service pattern of M, and $t + \Delta \in Inv(M_j)$. Then, $B(t+\Delta) \overset{\text{def}}{=} Buf\left(B(t), A, C, SP, \Delta, t\right)$*

When $t = 0$, $B(\Delta) = Buf\left(B(0), A, C, SP, \Delta, 0\right)$, or simply $Buf\left(B(0), A, C, SP, \Delta\right)$. The function *Buf* varies with the scheduling policy *SP*, however, it is always deterministic. Given below are *Buf* function for FP and TDMA scheduling.

- $Buf\left(B(t), A, C, \text{FP}, \Delta, t\right) \overset{\text{def}}{=} (b_1, \cdots, b_n)$ such that $b_i = \max\left\{0,\ A_i(t+\Delta) - A_i(t) + B_i(t) - \left\lfloor \frac{p_i}{e_i} \right\rfloor \right\}$ $\forall 1 \le i \le n$ where, $p_i = C(t+\Delta) - C(t) - \sum_{c_k < c_i} \left\{ e_k . \left[A_k(t+\Delta) - A_k(t) + B_k(t) \right] \right\}$.
- $Buf\left(B(0), A, C, \text{TDMA}, \Delta\right) \overset{\text{def}}{=} (b'_1, \cdots, b'_n)$ where

$$b'_i = \max\left\{0,\ A_i(\Delta) + B_i(0) - \left\lfloor \frac{p'_i}{e_i} \right\rfloor \right\}$$
$$p'_i = C(c_i) . \left\lfloor \frac{\Delta}{c} \right\rfloor + C\left(\min\left\{ c_i,\ \Delta - c . \left\lfloor \frac{\Delta}{c} \right\rfloor - \sum_{\chi_k < \chi_i} c_k \right\} \right)$$

assuming χ_k is the index of slot c_k in a TDMA cycle.

Definition 3 (Mode Execution)**.** *A tuple (A, C, B, Δ) is an execution of a mode $M = \left\langle \beta, \tau, SP, c, \left\{ (\alpha_i, c_i) \mid T_i \in \tau \right\} \right\rangle$ iff*

- *A is an arrival pattern of M,*
- *C is a service pattern of M,*
- *$B(x) = Buf(B(0), A, C, SP, x)$ for all $0 \le x \le \Delta$, and*
- *$\Delta \in Inv(M)$.*

Definition 4 (TET Execution)**.** *A sequence*

$$tr = \langle M_{k_1}, tr_1 \rangle \rightarrow \langle M_{k_2}, tr_2 \rangle \rightarrow \cdots \langle M_{k_h}, tr_h \rangle$$

is an execution trace of $\mathcal{A}$ iff $M_{k_1} \equiv M_{in}$ and for all $1 \leq j \leq h$:

- $M_{k_j} \in \mathcal{M}$,
- $tr_j = (A_j, C_j, B_j, \Delta_j)$ *is an execution of* M_{k_j},
- $(M_{k_j}, a, \varphi_j, M_{k_{j+1}}) \in \mathcal{R}$ *for some* $a \in \Sigma$ *and* $\varphi_j \in \Phi_{\mathcal{B}}$,
- $B_j(\Delta_j)$ *satisfies* φ_j *and* $B_j(\Delta_j) = B_{j+1}(0)$.

Maximum backlog analysis. To analyze the maximum backlogs of the buffers, we construct an underlying *behavioral automaton* of $\mathcal{A}$, denoted by $Beh(A)$. $Beh(A)$ is a state machine whose states are configurations of the form (M,b,A,C,t), where M is a mode of $\mathcal{A}$, b is a vector of n integers denoting the fill-levels of the buffers in $\mathcal{B}$ after t time units the system staying put at M_j, A is an arrival pattern and C is a service pattern of M defined to time t, and $t \in Inv(M)$. The initial state is the configuration $(M_{in}, b_{in}, A_{in}, C_{in}, 0)$, where M_{in} is the initial mode of $\mathcal{A}$ and b_{in}, A_{in}, C_{in} are all zero functions.

There is a transition from (M,b,A,C,t) to (M',b',A',C',t') if one of the following holds.

- $M' = M$, $t' = t+1$, $A'(\Delta) = A(\Delta)$ and $C'(\Delta) = C(\Delta)$ for all $0 \leq \Delta \leq t$, and $b' = Buf(b,A,C,SP,1,t)$ where SP is the scheduling policy of M.
- $M' \neq M$, $b' = b$, $A'_i(0) = 0$ for all $1 \leq i \leq n$, $C'(0) = 0$, $t' = 0$ and there is a transition (M,a,φ,M') in $\mathcal{A}$ such that (i) b satisfies φ or $\varphi = \emptyset$, and (ii) $t = D$ if there is a time D associated with the transition.

From the reachable states of $Beh(A)$, we can derive the exact maximum backlogs of the buffers. Specifically, the maximum backlog of B_i is the maximum value of $b_i(t)$ for all reachable states (M,b,A,C,t) in $Beh(A)$.

Maximum delay analysis. The maximum delay experienced by a stream can be done similarly by introducing additional variables into the behavioral automaton of the system. These variables are used to keep track of the waiting time of the data items.

Discussions. Observe that if there is no upper bound on the invariant of a mode, $\mathcal{A}$ may stay put at at a mode forever (i.e., the value of t in each configuration of $Beh(\mathcal{A})$ is not bounded). As a result, the behavioral automaton of $\mathcal{A}$ may potentially be infinite. To address this, we construct a regional automaton of $\mathcal{A}$, denoted as $Reg(\mathcal{A})$, which is a time abstract representation of $Beh(\mathcal{A})$. Each state of $Reg(\mathcal{A})$ corresponds to a set of reachable configurations of $Beh(\mathcal{A})$ that have the same TET mode and the same values of buffer fill-level (i.e., the b vector). There is a transition from a state (M,b) to a state (M',b') in $Reg(\mathcal{A})$ if there is a transition from a state (M,b,A,C,t) to a state (M',b',A',C',t') in $Beh(\mathcal{A})$. The analysis can then be done based on $Reg(\mathcal{A})$.

Although the size of $Reg(\mathcal{A})$ is significantly smaller than that of $Beh(\mathcal{A})$, it might still be infinite since the values of the buffer fill-level at the state may not be bounded. To allow feasible computation of $Reg(\mathcal{A})$, we assume that there is an upper bound N on the size of the buffers. The size of the $Reg(\mathcal{A})$ automaton will then be $O(N^n \|\mathcal{M}\|)$. More efficient abstraction approach such as the zone automata used in rectangular hybrid automata can be applied. However, we still require a known upper bound on the size of the buffers for the automaton to be finite.

IV. Approximate Timing Analysis

The analysis approach presented in the previous section, though being exact, can be computationally expensive as the size of the region automaton is exponential in the size of the buffer and the number of tasks. Further, it is only finite assuming an upper bound on the size of the buffer, which must be specified a priori. The exact method can verify if a buffer fill-level exceeds the specified maximum size, however in the case the fill-level of the buffer is unbounded, the analysis goes on forever in searching for the largest backlog value.

In general, there could be scenarios where the maximum backlog of a buffer is unbounded – in which case the system is said to be *unstable*. While stability is often assumed for single-mode systems, the same might not hold for the case of multi-mode systems, unless being verified. As a result of mode switching, the backlog of a buffer can get accumulated when executing along a sequence of modes. If the system does not have enough resource to clear the backlog and/or does not put any constraint on when the mode switching can take place, this backlog may potentially become infinite. As a result, we cannot simply assume that the system is stable. In fact, it is important for the analysis to be able to detect the instability of the system quickly and provide diagnosis feedbacks for the system designer to improve the system.

The above observations have motivated us to develop an alternative method that is much more efficient, capable of identifying system instability as well as computing safe estimations on the system performance properties.

Overview of TET approximate analysis: The key idea of TET approximate analysis is that – instead of enumerating all possible values of the buffer backlog during execution as done in the exact method – we keep track of only the execution paths that lead to the maximum backlogs of the buffers and the maximum backlog values at each mode along the paths. While the exploration of the path is based on the structure of the input TET automaton $\mathcal{A}$, the maximum backlog of a buffer at each mode in an explored path is "computed" directly from the arrival and service functions of the mode using RTC technique, without having to consider every TET execution to search for the value. This exploration results in an abstract tree $\mathcal{G} = (V,E)$ which captures all execution traces in $\mathcal{A}$ that will lead to a maximum buffer backlog. $\mathcal{G}$ can be viewed as an abstraction of the region automaton used in the exact method, and it is used to compute the timing properties of the system.

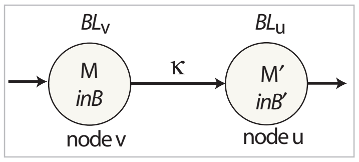
In the two coming sections, we detail the construction of $\mathcal{G}$. Section IV-C and IV-D show how maximum backlog and maximum delay can be computed based on $\mathcal{G}$. It is worth noting that, since the RTC method is not tight in general, the method presented here is inherently an approximate analysis.

A. The key idea and basic results for the construction of $\mathcal{G}$

$\mathcal{G}$ is formed by a set of nodes V and a set of edges E. Each node $v \in V$ consists of a mode $M \in \mathcal{M}$ and a tuple $inB = (inB_1, \ldots, inB_n)$ where inB_i specifies the maximum initial backlog of B_i when the system enters v. In addition, v is associated with an n-tuple BL_v where $BL_{v,i}$ is the maximum backlog of B_i when the system is at v. Each edge $e \in E$ from a node $v = (M, inB)$ to a node $u = (M', inB')$ is associated with a tuple $\kappa = (\kappa_1, \ldots, \kappa_n)$ where

- $\kappa_i = 1$ if the value of inB'_i is always upper bounded by the buffer guard associated with the transition from M to M' in $\mathcal{A}$, or by the values of the arrival and service functions associated with M;
- $\kappa_i = 0$, otherwise.

The attributes associated with a node and a transition of $\mathcal{G}$ are summarized in Fig. 2.



$inB = (inB_1, \ldots, inB_n)$
$BL_v = (BL_{v,1}, \ldots, BL_{v,n})$
$\kappa = (\kappa_1, \ldots, \kappa_n)$

inB_i = maximum initial backlog of B_i when the system enters M
$BL_{v,i}$ = maximum backlog of B_i when the system stays put at M
κ_i = 1, if inB_i' is always bounded due to the buffer guard associated with transition M $\rightarrow$M′ or values of the arrival and service functions at M.
κ_i = 0, otherwise.

Fig. 2: Attributes of a node and a transition in $\mathcal{G}$.

Fig. 3 gives an overview on the construction of $\mathcal{G}$. As highlighted in the figure (by the pointing fingers), at each reachable node $v = \langle M, inB \rangle$, we need to compute BL_v. Further, for each mode M' reachable from M (by taking an enabled outgoing transition), we need to compute the maximum initial backlogs when the system enters M'. Below we describe the computation of these attributes, which will serve as the building blocks for the construction of $\mathcal{G}$ detailed in the next section.

Suppose $M = \left\langle \beta, \tau, SP, c, \{(\alpha_i, c_i) \mid T_i \in \tau\} \right\rangle$. To obtain the maximum buffer backlogs at M and the maximum initial backlogs upon entering M, we compute:

(S1) The service function β_i of the resource allocated to each task $T_i \in \tau$ based on β, SP, α_i, and the initial maximum buffer backlogs when $\mathcal{A}$ enters M.

(S2) The maximum buffer backlogs when $\mathcal{A}$ stays put at M. based on β_i and $Inv(M)$.

(S3) The maximum buffer backlogs when $\mathcal{A}$ moves to M', for every transition from M to M' in $\mathcal{A}$ that is enabled.

Computing (S1). The service function β_i that bounds the portion of resource given to stream s_i when the system is at M is given by a function called *Serv*. Typically, *Serv* is defined based on β, α_i, SP, and the execution demand df_i of B_i (i.e., the number of processor cycles required to process the data items currently in B_i).

Definition 5. *Suppose df_i is execution demand of B_i when the system enters M and let $df = (df_1, \ldots, df_n)$. The service function of the resource allocated to each task $T_i \in \tau$ when the system is at M is given by $Serv(i, df, \alpha, \beta, SP)$.*

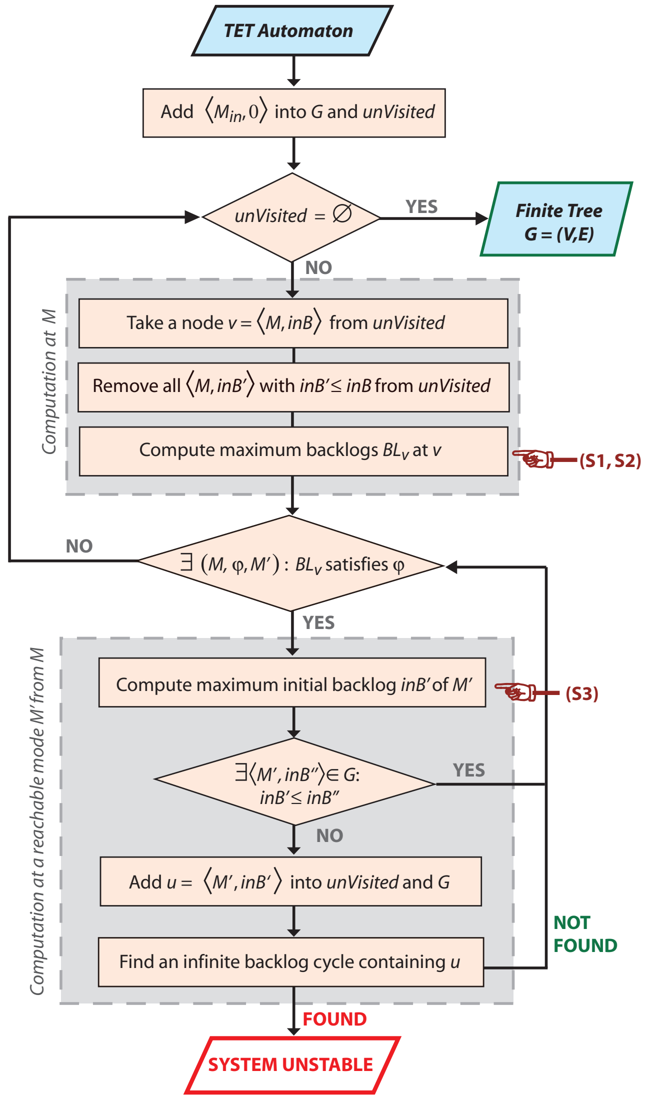

TET Automaton
Add $\langle M_{in}, 0 \rangle$ into G and unVisited
unVisited = ∅
YES
Finite Tree G = (V,E)
NO
Computation at M
Take a node $v = \langle M, inB \rangle$ from unVisited
Remove all $\langle M, inB' \rangle$ with $inB' \le inB$ from unVisited
Compute maximum backlogs BL_v at v
(S1, S2)
NO
$\exists (M, \varphi, M')$: BL_v satisfies φ
YES
Computation at a reachable mode M′ from M
Compute maximum initial backlog inB′ of M′
(S3)
$\exists \langle M', inB'' \rangle \in G$: $inB' \le inB''$
YES
NO
Add $u = \langle M', inB' \rangle$ into unVisited and G
Find an infinite backlog cycle containing u
NOT FOUND
FOUND
SYSTEM UNSTABLE

Fig. 3: Overview on the construction of $\mathcal{G}$.

Lemma IV.1 and IV.2 define *Serv* for FP and TDMA. The proofs for the lemmas are available in [14].

Lemma IV.1. *Let $df = (df_1, \ldots, df_n)$ with df_i denoting the execution demand of B_i when the system enters M. Let $\tau^i \subseteq \tau$ be the set of tasks in τ that have higher priority than T_i, i.e.,*

$$\tau^i = \left\{ T_k \in \tau \mid c_k < c_i \right\}.$$

The lower service function that bounds the amount of resource allocated to a task $T_i \in \tau$ when $SP = \text{FP}$ is given by

$$Serv(i, df, \alpha, \beta, \text{FP}) \stackrel{\text{def}}{=} \Big(\beta^l - \sum_{T_k \in \tau^i} \{ df_k + e_k \alpha_k^u \} \Big)_+$$

where $f_+(x) = \max\{f(x), 0\}$ for all $f : \mathbb{N} \to \mathbb{N}$ and $x \ge 0$.

When SP is TDMA, in each TDMA cycle, T_i may not be allocated resource during a time interval of length $\Delta = c - c_i$, however it is granted full access to the resource during a time interval of length c_i. As a result, one can compute the service function of the resource given to T_i using Lemma IV.2 below.

Lemma IV.2. *The lower service function that bounds the amount of resource allocated to a task $T_i \in \tau$ when $SP =$ TDMA is given by $Serv(i, df, \alpha, \beta, \text{TDMA}) \overset{\text{def}}{=} \overline{\beta}_i$ where*

$$\overline{\beta}_i(\Delta) = \max\left\{ \left\lfloor \frac{\Delta}{c} \right\rfloor . \beta^l(c_i),\ \beta^l(\Delta) - \left\lceil \frac{\Delta}{c} \right\rceil . \beta^l(c - c_i) \right\} \forall\, \Delta \geq 0.$$

Computing (S2). Since the execution demand of T_i is e_i, the following corollary holds.

Corollary IV.3. *Let $df_v = (df_{v,1}, \ldots, df_{v,n})$ with $df_{v,i}$ denoting the maximum execution demand of B_i when the system enters v. Then, $df_{v,i} = e_i.inB_i$ for all $1 \leq i \leq n$.*

Lemma IV.4. *Denote $[L,U] = Inv(M)$. The maximum backlog of B_i when the system stays put at M is given by*

$$BL_{v,i} \overset{\text{def}}{=} \begin{cases} inB_i + \max\limits_{0 \leq \Delta \leq U} \left\{ \alpha_i(\Delta) - \beta_i(\Delta) \right\}, & \text{if } T_i \in \tau \\ inB_i, & \text{otherwise.} \end{cases}$$

where $\beta_i = \lfloor Serv(i, df_v, \alpha, \beta, SP)/e_i \rfloor$.

Computing (S3). Suppose $r = (M, a, \varphi, M')$ is a transition in $\mathcal{A}$. Let $[L,U] = Inv(M) \cap [D,U]$ if there is a time D associated with r, and $[L,U] = Inv(M)$ otherwise. In other words, $[L,U]$ is the interval during r can be taken. Recall that $\beta_i = \lfloor Serv(i, \mathrm{df}_v, \alpha, \beta, SP)/e_i \rfloor$ and $\varphi_i^{\max}$ is the maximum value of B_i that satisfies φ.

Lemma IV.5. *The maximum backlog of B_i when the system enters M' by taking the transition r is given by*

$$inBL(v, \varphi, i) = \min\left\{ outBL_{v,i},\ \varphi_i^{\max} \right\}$$

$$\text{where:} \quad outBL_{v,i} \overset{\text{def}}{=} \max \begin{cases} \max\limits_{0 \leq \Delta < L} \left\{ \alpha_i(\Delta) - \beta_i(\Delta) \right\} \\ inB_i + \max\limits_{L \leq \Delta \leq U} \left\{ \alpha_i(\Delta) - \beta_i(\Delta) \right\} \end{cases}$$

if $T_i \in \tau_j$, and $outBL_{v,i} = inB_i$ otherwise.

B. Construction of the abstract tree $\mathcal{G}$

This section gives a detailed description to the tree construction in Fig. 3. We start with the root of the tree $v_{in} = \langle M_{in}, inB_{in} \rangle$ where M_{in} is the initial mode of $\mathcal{A}$ and inB_{in} is an n-tuple of zeros (since buffers are initially empty). We add v_{in} into V and into the set of unvisited nodes termed *unVisited*. Both V and *unVisited* are empty initially.

Let $v = \langle M, inB \rangle$ be a node in *unVisited* . We first remove all nodes $v' = \langle M, inB' \rangle$ in *unVisited* such that $inB \geq inB'$. This is because corresponding to each path ρ from v to a node w, there is a path ρ' from v' to a node w' passing through the same sequence of modes as that of ρ and the maximum initial backlogs when the system enters w is larger than or equal to the maximum initial backlogs when the system enters w'. In other words, $BL_w \geq BL_{w'}$. As we are computing the maximum backlog, it is sufficient to consider only paths starting from v.

We next compute the maximum backlogs BL_v when the system is at M using Lemma IV.4. Suppose $r = (M, a, \varphi, M')$ is a transition in $\mathcal{A}$ such that BL_v satisfies φ. Then, r is enabled and the maximum backlog of B_i when the system enters M' is given by $inB'_i = inBL(v, \varphi, i)$ (Lemma IV.5). We create a new node $u = \langle M', inB' \rangle$.

Case 1: If there is a node $w = \langle M', inB'' \rangle$ in $\mathcal{G}$ such that $inB'' \geq inB'$, there is no need to explore u. Hence, we delete u and continue with the next transition from M.

Case 2: If no such node w exists, we add u into V and a new edge (v, κ, u) into E. Here, $\kappa_i = 0$ if $\varphi_i^{\max}$ is infinite and

$$inB'_i = inB_i + \max_{L \leq x \leq U} \left\{ \alpha_i(x) - \beta_i(x) \right\}$$

where β_i is the service function of the resource given to s_i, and $[L,U]$ is the interval during which r can be taken (as computed in Section IV-A). Otherwise, $\kappa_i = 1$.

If there is an edge (u_0, u_1) in the path from the root to v such that u_0 contains M and u_1 contains M', we will check if this path contains a zero cycle that leads to an infinite buffer backlog. Specifically, let $\rho = u_0 \xrightarrow{\kappa^1} u_1 \xrightarrow{\kappa^2} u_2 \xrightarrow{\kappa^3} \cdots \xrightarrow{\kappa^h} u_h$ be the path from u_0 to u in $\mathcal{G}$ with $u_k = \langle M_{j_k}, inB^k \rangle$, $M_{j_0} \equiv M$, $M_{j_1} \equiv M'$, $v \equiv u_{h-1}$ and $u \equiv u_h$. Note that, there exists $1 \leq i \leq n$ such that $inB_i^h > inB_i^1$ (otherwise, u has been deleted earlier in Case 1). There are two cases:

i If for all i such that $inB_i^h > inB_i^1$, there exists $\kappa_i^k = 1$ for some $1 \leq k \leq h$, then the fill-level of B_i is always upper bounded. We mark u as a *bounded* node.

ii Otherwise, there is an i where $\kappa_i^k = 0, \forall 1 \leq k \leq h$. We claim that the maximum backlog of B_i will be infinite if we repeats ρ for an infinite number of times [14]. Hence, we report an infinite buffer backlog at B_i, return the path from the root of $\mathcal{G}$ to u, and terminate the construction.

We add u into the set *unVisited* and continue with the next transition from v until all the transitions are explored. We then continue with the next node in *unVisited* set until it is empty.

Lemma IV.6. *The construction of $\mathcal{G}$ is decidable.*

For each mode M that appears in a cycle in $\mathcal{A}$ and $\tau = (M', \varphi, M)$ is an incoming transition from M, let $X_i^\tau = \max_{L \leq x \leq U}\{\alpha_i(x) - \beta_i(x)\}$, where β_i is the service allocated to T_i assuming zero initial buffer backlogs for the buffers, and $[L,U]$ is the interval during which τ can be taken. Define $K_i^\tau = 1$ if $\varphi_i^{\max} = \infty$ and $\varphi_i^{\max}/X_i^\tau$, otherwise. Further, let K_i be the maximum value of K_i^τ for all incoming transition τ of M. From the construction, we imply that K_i is the maximum number of values for the initial maximum backlog of B_i when $\mathcal{A}$ enters M. Thus, the number of times M appears in a node in $\mathcal{G}$ is at most $K = \prod_{i=1}^n K_i$. As a result, the size of the tree $\mathcal{G}$ is $O(K\|\mathcal{M}\|)$. Since the number of modes in the automaton is relatively small, the algorithm is highly scalable.

C. Computing maximum backlog

The maximum backlog of the buffer B_i at a mode M_j is the maximum value of $BL_{v,i}$ for all $v \in V$ that contain M_j:

$$BL(B_i, M_j) = \max_{v \in V}\left\{ BL_{v,i} \mid v = \langle M_j, inB \rangle \wedge inB \in \mathbb{N}^n \right\}$$

The maximum backlog of B_i experienced by the stream is:

$$BL(B_i) = \max\left\{ BL(B_i, M_j)\ \mid\ M_j \in \mathcal{M} \right\}$$

D. Computing maximum delay

We now present our method for computing the maximum delay of an event stream s_i at a mode $M \in \mathcal{M}$. Let

$$tr = \langle M_{in}, tr_{in} \rangle \to \cdots \to \langle M_{k_1}, tr_1 \rangle \to \langle M_{k_2}, tr_2 \rangle \to \cdots$$

be an execution trace of $\mathcal{A}$ that results in the maximum delay of s_i at mode M, where $M_{k_1} \equiv M$ and $tr_{k_j} = (A_j, C_j, B_j, \Delta_j)$ for all $j \geq 1$. Suppose ev is an event/item of s_i which has the longest delay among all items of s_i that arrive when the system is at mode M. The maximum delay of s_i at M is then the delay $delay(M,i)$ experienced by ev.

Since we only consider systems that have finite buffer backlogs, $delay(M,i)$ is finite. Thus, ev will be fully processed at some Δ_h time units after the system enters a mode M_{k_h} with $1 \leq h \leq \infty$. Denote λ as the amount of time from the instant $\mathcal{A}$ enters M to the instant ev arrives. Then,

$$delay(M,i) + \lambda = \Delta_1 + \cdots + \Delta_h \tag{1}$$

Further, $delay(M,i)$ is the amount of time needed to process the $B_{j,i}(0)$ items initially in the buffer B_i when the system enters M and the data items that arrive during the λ time units the system is at M. Since the number of items that arrive from s_i and the amount of resource allocated to s_i in M are independent of the data that arrive from s_i in previous modes, the delay of ev is largest implies $B_{j,i}(0)$ equals the maximum backlog of B_i whenever the system enters M, i.e.,

$$B_{j,i}(0) = \max_{\langle M, inB \rangle \in V} \left\{ inB_i \mid \langle M, inB \rangle \in V \right\} \overset{\text{def}}{=} inB^{\max}_{M,i}$$

Let $v_{k_1} = \langle M, inB^1 \rangle \in V$ be a node containing M at which $inB^1_i = inB^{\max}_{M,i}$. Then $\langle M_{k_1}, tr_1 \rangle \to \cdots \to \langle M_{k_h}, tr_h \rangle$ corresponds to a path $\pi = v_{k_1} \to \cdots \to v_{k_h}$ in $\mathcal{G}$ with $v_{k_j} = \langle M_{k_j}, inB^j \rangle$ for all $1 \leq j \leq h$. Moreover, the amount of resource given to s_i when the system is at v_p is bounded by $\overline{\beta}_{p,i} = Serv(i, \mathrm{df}_{v_p}, \alpha_p, \beta_p, SP_p)$, where $\mathrm{df}_{v_p} = (\mathrm{df}_{v_p,1}, \ldots, \mathrm{df}_{v_p,n})$ and $\mathrm{df}_{v_{k_j},i} = e_i inB^j_i$ for all $1 \leq i \leq n$. Since ev is fully processed after the system being at M_{k_h} for Δ_h units of time, the total execution demand of the data items that arrive before ev and ev must be no more than the total resource given by the system along path π in the worst case. In other words,

$$\left(\alpha^u_{k_1,i}(\lambda) + inB^1_i\right) e_i \leq \overline{\beta}_{k_1,i}(\Delta_1 - \lambda) + \sum_{j=2}^{h} \overline{\beta}_{k_j,i}(\Delta_j) \tag{2}$$

From Eqs (1) and (2), we imply that

$$delay(M,i) \leq \max\Big\{ delay(\pi,i) \mid M_{k_1} \equiv M \wedge inB^1 = inB^{\max}_{M,i} \wedge \pi = \langle M_{k_1}, inB^1 \rangle \to \cdots \to \langle M_{k_h}, inB^h \rangle \in \mathcal{G} \Big\},$$

where: $$delay(\pi,i) = \max_{\lambda \in Inv(M)} \left\{ \min_{\Delta_j \in Inv(M_{k_j})} \left\{ \sum_{j=1}^{h} \Delta_j - \lambda \;\middle|\; \left(\alpha^u_{k_1,i}(\lambda) + inB^1_i\right) e_i \leq \overline{\beta}_{k_1,i}(\Delta_1 - \lambda) + \sum_{j=2}^{h} \overline{\beta}_{k_j,i}(\Delta_j) \right\} \right\}$$

The overall maximum delay experienced by a stream s_i is then the maximum value of $delay(M,i)$ for all $M \in \mathcal{M}$.

Note that both the computations of backlog and delay are safe approximation in that the actual maximum backlog of a buffer and maximum delay of a stream will be less than or equal to the computed values.

V. Case Study

In this section we present a case study to illustrate the applicability of TET automata in realistic multi-mode real-time systems. In particular, we look into a smart-phone which is equipped with advanced features such as emails, Internet connectivity via Wi-Fi, and media multitasking. We show how the proposed model can be used to describe such a system and to derive various performance properties using our analysis methods. Based on the obtained analysis results, designers are able to tune the system design to minimize resource usage while assuring quality of service requirements.

We assume that the system consists of five main tasks (Table II) executing on a single processor architecture (Fig. 4).

Task	Functionality
T_{kernel}	Kernel threads that should be running at all time, for instance, system clocks and I/O user inputs.
$T_{monitor}$	Connection monitoring activities such as telemetry and wireless data collection, which is activated periodically.
T_{update}	System updates and maintenance routines such as display updating, which is executed periodically every D_u time units.
T_{audio}	Audio processing of voice data when users receive/make calls.
T_{video}	Video processing when users stream video using WiFi.

TABLE II: Different tasks of a smart-phone.

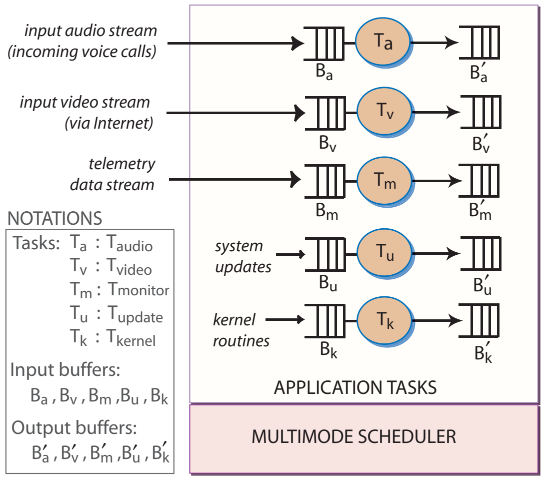


Fig. 4: The system architecture of a smart-phone application.

As in many complex real-time applications, the smart-phone has a dynamic scheduler. It is a multi-mode system, with each operating mode executing a different subset of the above tasks. The processor frequency is scaled to best match with the processing requirement of each mode. The scheduling of the tasks active at each mode is also chosen with respect to the tasks' execution patterns and the overall performance objectives. In this case study, we assume that Fixed Priority is employed for all modes. Further, voice calls have higher performance requirements than that of video streaming. However, the scheduler will allocate more resources to the video stream to take advantage of idle periods during the call and hence there is only little voice data (e.g., background noise). On the whole, we have seven different modes:

- *Active:* The system is idle and executes only T_{kernel} and $T_{monitor}$, with T_{kernel} having higher priority than $T_{monitor}$.

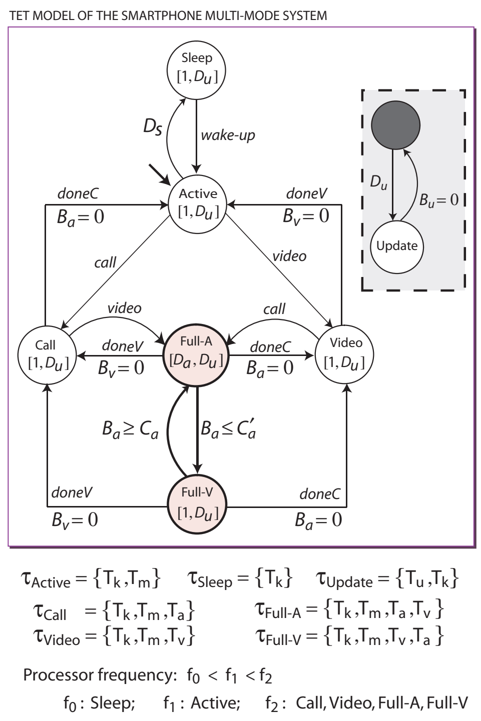
- *Call:* The system enters this mode when the user makes/receives a call. Here, three tasks T_{kernel}, $T_{monitor}$ and T_{audio} are executed, with T_{kernel} having highest priority and $T_{monitor}$ having the lowest priority.
- *Video:* The system is in this mode when the user watches streaming video. In this mode, three tasks T_{kernel}, $T_{monitor}$ and T_{video} are executed, with T_{kernel} having the highest priority and T_{video} having the lowest priority.
- *Full-A:* The system is in this full-on mode when the user watches video while taking/making a phone call. Thus, four tasks T_{kernel}, $T_{monitor}$, T_{audio}, andT_{video} get executed, in decreasing order of priority.
- *Full-V:* The system enters this mode if the fill-level of the audio stream is below a certain threshold and it has spent at least D_a time units in *Full-A*. This happens when there is an idle period during the conversation and the audio traffic contains mainly background noise. Hence, we assign a higher priority to the video stream. The system will switch back to *Full-A* mode if the fill-level of the audio buffer exceeds a value C_a (i.e., the end of the idle period).
- *Update:* In this mode, only two tasks T_{kernel} and T_{update} are executed, with T_{update} having higher priority.
- *Sleep:* The system enters this power saving mode if there is no activity after a duration of D_s time units. It will execute only T_{kernel} and periodically wakes up to perform system update and returns back to sleep.

When the system is in *Sleep* mode, the processor runs at the smallest frequency f_0. When it is in *Active* mode, it runs at the normal frequency f_1. When there are computationally expensive tasks such as audio and/or video processing, the processor operates at the maximum frequency f_2.

In designing such a system, an important question that must be addressed is to decide how much buffer space is required to ensure there will be no buffer overflows. A typical way of doing this is to compute the maximum backlogs of the different buffers in the system; these computed values can then be used as the minimum buffer capacities that need to be allocated to each task to avoid buffer overflows.

TET model of the smart-phone multi-mode system. We now demonstrate how TET automata can easily capture the dynamic behavior of the given multi-mode system. Based on the resulting model, we compute the maximum backlogs of the buffers using our proposed analysis methods. Note that it might be possible to represent the system using a model that has no mode modeling capability by viewing the system as having a single mode (with the worst combination of the properties of the original modes). However, this approach often leads to overly pessimistic results as we will demonstrate later. In this case study, we are specifically focus on the input buffers of the voice and video streams (due to space restrictions).

Fig. 5 depicts the TET automaton for the smart-phone system. For ease of presentation, the mode *Update* is not shown in the automaton. There should be a transition to and from each mode of the automaton to mode *Update*, as shown



TET MODEL OF THE SMARTPHONE MULTI-MODE SYSTEM
Sleep $[1, D_u]$
D_S
wake-up
Active $[1, D_u]$
doneC
$B_a = 0$
doneV
$B_v = 0$
call
video
Call $[1, D_u]$
Full-A $[D_a, D_u]$
Video $[1, D_u]$
video
call
doneV
$B_v = 0$
doneC
$B_a = 0$
$B_a \geq C_a$
$B_a \leq C'_a$
Full-V $[1, D_u]$
doneV
$B_v = 0$
doneC
$B_a = 0$
D_u
$B_u = 0$
Update
$\tau_{Active} = \{T_k, T_m\}$ $\tau_{Sleep} = \{T_k\}$ $\tau_{Update} = \{T_u, T_k\}$
$\tau_{Call} = \{T_k, T_m, T_a\}$ $\tau_{Full\text{-}A} = \{T_k, T_m, T_a, T_v\}$
$\tau_{Video} = \{T_k, T_m, T_v\}$ $\tau_{Full\text{-}V} = \{T_k, T_m, T_v, T_a\}$
Processor frequency: $f_0 < f_1 < f_2$
f_0: Sleep; f_1: Active; f_2: Call, Video, Full-A, Full-V

Fig. 5: The TET model of the multimode scheduler (cf. Fig. 4).

in the dashed box. In the figure, the tasks at each mode are listed from the highest priority to the lowest priority. D_u is the period of T_{update}, D_a is the minimum duration the system must stay at mode *Full-A*, and D_s is the amount of inactivity time for the system to go to *Sleep* mode. In the figure, B_a and B_u denote the input buffer of the audio (voice) stream and the stream of update jobs, respectively. Further, $C_a, C'_a \in \mathbb{N}$ are the pre-specified thresholds of the buffer fill-level for which the system switches between the two full-on modes. We assume that when the system is in the *Update* mode, it keeps the same processor frequency as the previous mode it came from.[1]

Experimental setup and results. Since T_{kernel}, $T_{monitor}$ and T_{update} are periodic tasks, their arrival functions are computed directly from the chosen periods. We assume that the arrival functions for these tasks remain constant for all the modes.

The input audio and video streams in general can be bursty and hence, their arrival functions take arbitrary forms. To obtain the arrival functions of the video stream, we use a set of representative video clips and simulate their executions on a customized version of the SimpleScalar instruction set simulator [15]. From the execution traces, we measured the execution demands of the decoding tasks for each macroblock and derived a function $A(t)$ which gives the number of macroblocks arriving at the input buffer B_v during the time interval $[0,t]$. This function is then used to compute the arrival function $\alpha_v(\Delta)$ of the video input stream. The arrival function for the audio stream is obtained in a similar fashion.

[1]To capture this, instead of having only one shared *Update* mode in the automaton, we have one copy $Update_M$ for each mode M in the automaton that is shown in Fig. 5 and set the frequency to be the same as that of M.

In our experiment, the arrival function of the video stream is chosen to be the same for both the modes *Full-A* and *Full-V*. The arrival function of the audio stream at mode *Full-V* is smaller than at mode *Full-A*. This is because the system will only give higher priority to the audio stream when there is less input data from the audio stream (to maintain a high level of service for voice calls at all time).

The service function of the system is computed based on the frequency of the processor. We assume that the processor does not run any other tasks besides the given task set. Hence, its service function is given by $\beta_f^l(\Delta) = \beta_f^u(\Delta) = f.\Delta$, where f is the frequency of the processor. In our experiment, we set $f_1 = 200MHz$, $f_2 = 400MHz$, and $f_3 = 600MHz$. The threshold at which the system moves to mode *Full-V* is set to $C_a' = 500$.

For the analysis, we implemented the approximation methods outlined in Section IV. Additionally, we performed an analysis without mode modeling using the original RTC model where we computed the maximum backlog of the video buffer for each mode individually and took the worst value.

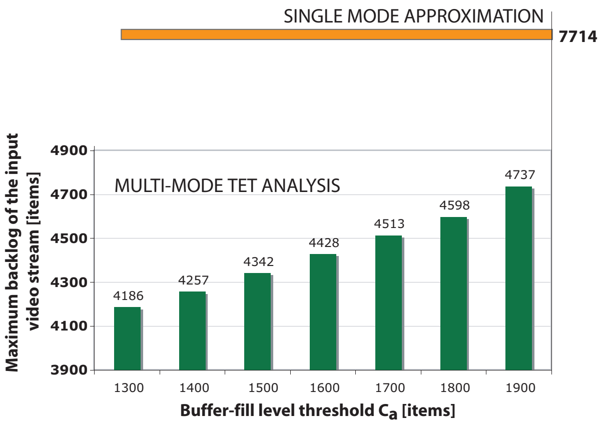


Fig. 6: The maximum of the video input buffer B_v (cf. Fig. 4).

Fig. 6 plots the resulting maximum backlog of the input video buffer B_v for different values of C_a (see Fig. 5) using TET model, and using the single-mode RTC approximation. The bar graphs represent the results of TET analysis, whereas the horizontal line represents the result from the single-mode approximation. When comparing the TET results with the single-mode approximation, it may be seen that explicitly modeling the dynamic behavior of the system results in significantly tighter estimates, thereby leading to better resource dimensioning (which, here, is the on-chip buffer memory).

Further, it may be observed that the maximum backlog of B_v given by the TET analysis does not remain constant but increases with C_a. Based on these computed results, one may easily determine the amount of buffer memory that should be provisioned for the video stream, for any chosen threshold value of C_a. Alternatively, this information can also help the system developer in tuning the scheduler to minimize resource usage (e.g., by selecting the best value of C_a that minimizes the total memory requirement of the system). Such tuning would not be possible with the simplistic single-mode model.

VI. Concluding Remarks

We have proposed a model called TET automata and a set of associated techniques for analyzing multi-mode systems where mode changes are both time- and event-triggered. Our first approach uses automata verification techniques that tackle stringent cases requiring an exact timing analysis. A second technique was developed – incorporating both Real-Time Calculus and automata state exploration – to provide approximate performance metrics for the whole system based on the results derived from the individual modes. This combination of techniques from two different domains produced a solution which is more efficient and at the same time is guaranteed to give safe estimates. The applicability and benefits of our proposed model have been demonstrated in a smart-phone multi-mode system, where mode changes are driven by both time and fill-level of various buffers in the system. It would be interesting to explore how the TET model can be lifted to an interface-theoretic setting to enable more efficient/lightweight compositional analysis and correct by construction design.

Acknowledgment

This research was supported in part by AFOSR FA9550-07-1-0216, NSF CNS-0720703, and NSF CNS-0721541.

References

[1] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: a time-triggered language for embedded programming," *Proceedings of the IEEE*, vol. 9, no. 1, pp. 84–99, 2003.

[2] S. Chakraborty, S. Künzli, and L. Thiele, "A general framework for analysing system properties in platform-based embedded system designs," in *DATE*, 2003.

[3] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli, "A framework for evaluating design tradeoffs in packet processing architectures," in *39th Design Automation Conference (DAC)*, 2002.

[4] E. Wandeler, A. Maxiaguine, and L. Thiele, "Quantitative characterization of event streams in analysis of hard real-time applications," *Real-Time Systems*, vol. 29, no. 2-3, pp. 205–225, 2005.

[5] E. Wandeler and L. Thiele, "Workload correlations in multi-processor hard real-time systems," *Journal of Computer and System Sciences (JCSS)*, vol. 73, no. 2, pp. 207–224, 2007.

[6] G. C. Buttazzo, G. Lipari, and L. Abeni, "Elastic task model for adaptive rate control," in *RTSS*, 1999.

[7] Y. Shin, D. Kim, and K. Choi, "Schedulability-driven performance analysis of multiple mode embedded real-time systems," in *Design Automation Conference (DAC)*, 2000.

[8] G. Fohler, "Changing operational modes in the context of pre run-time scheduling," *IEICE Transactions on Information and Systems*, vol. E76-D, no. 11, pp. 1333–1340, 1993.

[9] L. Sha, R. Rajkumar, J. Lehoczsky, and K. Ramamritham, "Mode change protocols for priority-driven preemptive scheduling," *Real-Time Systems*, vol. 1, no. 3, pp. 244–264, 1989.

[10] J. Real and A. Crespo, "Mode change protocols for real-time systems: A survey and a new proposal," *Real-Time Systems*, vol. 26, pp. 161–197, 2004.

[11] T. Pop, P. Eles, and Z. Peng, "Design optimization of mixed time/event-triggered distributed embedded systems," in *CODES+ISSS*, 2003.

[12] N. Scaife and P. Caspi, "Integrating model-based design and preemptive scheduling in mixed time- and event-triggered systems," in *16th Euromicro Conference on Real-Time Systems (ECRTS)*, 2004.

[13] L. Phan, S. Chakraborty, and P. Thiagarajan, "A multi-mode real-time calculus," in *RTSS*, 2008.

[14] L. T. X. Phan, S. Chakraborty, and I. Lee, "Timing analysis of mixed time/event-triggered multi-mode systems," http://www.cis.upenn.edu/~linhphan/papers/rtss09.pdf, 2009.

[15] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An infrastructure for computer system modeling," *IEEE Computer*, vol. 35, no. 2, pp. 59–67, 2002.