Fifth Workshop on Next Generation Real-Time Embedded Systems

NG-RES 2024, January 17–19, 2024, Munich, Germany

Edited by
Patrick Meumeu Yomsi
Stefan Wildermann

Editors

Patrick Meumeu Yomsi
CISTER Research Centre, ISEP, Porto, Portugal
pmy@isep.ipp.pt

Stefan Wildermann
Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany
stefan.wildermann@fau.de

ACM Classification 2012
Computer systems organization → Multicore architectures; Networks → Network protocols; Computer systems organization → Distributed architectures

ISBN 978-3-95977-313-3

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-313-3.

Publication date
March, 2024

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/OASIcs.NG-RES.2024.0

ISBN 978-3-95977-313-3 **ISSN 1868-8969** **https://www.dagstuhl.de/oasics**

OASIcs – OpenAccess Series in Informatics

OASIcs is a series of high-quality conference proceedings across all fields in informatics. OASIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/oasics

Contents

Invited Papers

Regular Paper

Fifth Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2024).
Editors: Patrick Meumeu Yomsi and Stefan Wildermann
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

On behalf of the Technical Program Committee, we are pleased to welcome you to the Proceedings of the 5th Edition of the Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2024), which was held on January 19, 2024 in Munich, Germany. The NG-RES workshop series focuses on real-time embedded systems, with an emphasis on distributed and parallel aspects. NG-RES serves as a platform for collaboration between the networking and multicore real-time communities and promotes cross-fertilization and multidisciplinary approaches to embedded systems design.

Key topics of interest for NG-RES 2024 included, but are not limited to:

- Application of formal methods to distributed and/or parallel real-time systems
- Programming models, paradigms and frameworks for real-time computation on parallel and heterogeneous architectures
- Applications of approximate computing in real-time systems
- Compiler-assisted solutions for distributed and/or parallel real-time systems
- Middlewares for distributed and/or parallel real-time systems
- Networking protocols and services (e.g., clock synchronization) for distributed real-time embedded systems
- Scheduling and schedulability analysis for distributed and/or parallel real-time systems
- System-level software and technologies (e.g. RTOSs, hypervisors, separation kernels, virtualization) for parallel and heterogenous architectures

We express our gratitude to everyone involved in the organization of the workshop. Our special thanks go to the General Chair Federico Terraneo, Submission and Web Chair Daniele Cattaneo and sincere appreciation to the members of the Program Committee. Their dedicated support was crucial in putting together the workshop program and we thank you all.

Finally, a big thank you goes to all the authors who contributed to NG-RES 2024 with their work. Their valuable contributions make this workshop possible. We hope you will enjoy the event!

Patrick Meumeu Yomsi and Stefan Wildermann

Fifth Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2024).
Editors: Patrick Meumeu Yomsi and Stefan Wildermann
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of Authors

Vejey Pradeep Suresh Achari (2)
Keele University, Keele, UK

Issam Boukhennoufa (2)
University of Essex, Colchester, UK

Khalil Esper (4)
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany

Miguel Gutiérrez Gaitán (3)
Department of Electrical Engineering, Pontificia Universidad Católica de Chile, Santiago, Chile;
Faculty of Engineering, Universidad Andres Bello, Santiago, Chile;
CISTER/ISEP, Polytechnic Institute of Porto, Portugal

Mohammadhassan Gholami Derouei (1)
Università degli Studi di Modena e Reggio Emilia, Italy;
Minerva Systems srl, Modena, Italy

Anish Jindal (2)
Durham University, Durham, UK

Zeba Khanam (2)
BT Security Research, Adastral Park, UK

Harrison Kurunathan (3)
CISTER/ISEP, Polytechnic Institute of Porto, Portugal

Andrea Marongiu (1)
Università degli Studi di Modena e Reggio Emilia, Italy

Amit Kumar Singh (2)
University of Essex, Colchester, UK

Marco Solieri (1)
Minerva Systems srl, Modena, Italy

Ramiro Sámano-Robles (3)
CISTER/ISEP, Polytechnic Institute of Porto, Portugal

Jürgen Teich (4)
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany

Eduardo Tovar (3)
CISTER/ISEP, Polytechnic Institute of Porto, Portugal

Paolo Valente (1)
Università degli Studi di Modena e Reggio Emilia, Italy

Fifth Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2024).
Editors: Patrick Meumeu Yomsi and Stefan Wildermann
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

HMB: Scheduling PREM-Like Real-Time Tasks at High Memory Bandwidth

Mohammadhassan Gholami Derouei
Università degli Studi di Modena e Reggio Emilia, Italy
Minerva Systems SRL, Modena, Italy

Paolo Valente
Università degli Studi di Modena e Reggio Emilia, Italy

Marco Solieri
Minerva Systems SRL, Modena, Italy

Andrea Marongiu
Università degli Studi di Modena e Reggio Emilia, Italy

Abstract

Current homogeneous and heterogeneous computing systems reach high performance through parallelization. Yet, parallel execution of tasks entails non-trivial latency-vs-throughput issues when it comes to concurrent accesses to shared memory. In this respect, effective bandwidth regulation solutions do exist, and provide a basic mechanism to control the latency of memory accesses. Such solutions, though, are often cumbersome to deploy and to configure to guarantee both bounded latency and high utilization of the memory bandwidth. The problem is that memory latency varies non-linearly with the number and type of concurrent accesses, and the latter may in turn vary with time, often unpredictably. For this reason, previous attempts at memory regulation in scheduling solutions resulted either in poor real-time execution guarantees, or in severe underutilization of the memory bandwidth. In this paper, we outline High Memory Bandwidth (HMB), a scheduling solution that guarantees bounded response times to real-time task sets through memory regulation, while also reaching a high utilization memory bandwidth. Since the complete solution is complex, just like the problem it addresses, this preliminary work defines in full detail only the core mechanism. This mechanism builds on the notion of memory access slowdown experienced by any processor performing back-to-back memory operations; this slowdown is due to the interference generated by other processors also accessing the memory at the same time. The core mechanism assumes that each processor can tolerate a certain amount of slowdown before the timing behavior of the task(s) it is running is compromised. Each processor has a priority assigned: the higher the priority, the more stringent the timing requirements. The slowdown can be controlled by regulating with precision the maximum amount of system bandwidth each processor is allowed to use, based on its priority. The proposed mechanism finds the maximum bandwidth each processor can use such that the highest number of processors simultaneously accessing memory is found (thus avoiding memory bandwidth underutilization) while guaranteeing that the slowdown of each processor is kept within the tolerated limits.

2012 ACM Subject Classification Computer systems organization → Real-time operating systems

Keywords and phrases Heterogenous systems, Parallel execution, Shared memory, Bandwidth regulation, Memory access, Real-time execution, Memory bandwidth utilization, High Memory Bandwidth (HMB), Memory access slowdown, Memory interference, Memory-centric scheduling

Digital Object Identifier 10.4230/OASIcs.NG-RES.2024.1

Category Invited Paper

Acknowledgements Authors are grateful to Tomasz Kloda at TUM, and Benjamin Rouxel at UNIMORE, with whom they enjoyed inspiring discussions in the conception of this work.

1 Introduction

Memory contention on parallel systems

Modern systems feature multiple execution units, including CPU cores and accelerator cores, running in parallel. These units may inherently perform memory accesses simultaneously. To support parallelism in memory accesses, these systems are equipped with cache hierarchies. Caches help to make memory accesses as local as possible, thereby reducing the likelihood of accessing shared memories and interconnects. However, in many cases, conflicting accesses to shared resources remain unavoidable. For instance, these conflicting accesses become unavoidable when there is insufficient space in local caches to accommodate the cumulative memory footprint of the processors (e.g. CPU cores, execution units) utilizing those caches. Additionally, such conflicts arise when distant processors need to communicate through shared memory. Contention arises across all shared resources along the path from the processors to the actual memory banks, including the interconnect, shared caches, memory bus, and others. This contention, in turn, results in a more or less significant and unpredictable inflation of memory latency, impacting the duration of the memory access performed by the competing processors. [10]. This is an evident problem in real-time applications. Increased memory latencies can slow down the execution of tasks that involve memory accesses. This may make it impossible for the tasks to meet their deadlines, thereby rendering the task set unfeasible. Several solutions have been proposed in the scientific literature to eliminate or control this slowdown, which can be broadly grouped into two classes: (i) exclusive memory accesses; and (ii) limited memory accesses.

Exclusive memory access: low bandwidth and high predictability

The first type of solution is based on allowing one processor (or, very few processors) at a time to access shared memory [20, 21]. This approach either eliminates or reduces interference to such a low level that no significant slowdown occurs. However, the available bandwidth from shared memories is typically sized to meet the cumulative average bandwidth demand of the set of processors connected to that memory. Consequently, the total bandwidth offered by the memory is often higher or much higher than the bandwidth that a single processor may request. In the end, if only one or relatively few processors access memory at the same time, the memory bandwidth may be underutilized, potentially to a severe extent. For example, the ratio between the memory bandwidth available to a single CPU core and the one available to the whole system (or the CPU complex, respectively) ranges between: 5.0% (or 11%) on an A57 core in the NXP i.MX 8QM platform, 13% (or 24%) on a Carmel core in the Nvidia Xavier AGX, 15% (or 35%) on a A53 core in the AMD Xilinx Zynq UltraScale+ [4].

Limited memory access: either high bandwidth, or high predictability

The other type of solutions follows a somewhat opposite approach, as it allows multiple processors to access memory in parallel [11, 18, 22]. In this case, slowdowns are controlled by imposing a *limit* on the maximum bandwidth at which each processor can access memory. This approach is effective as long as the sum of the per-processor bandwidths remains low compared to the total memory bandwidth. In this particular case, memory contention is negligible, and per-processor bandwidths add up linearly – the execution latencies of parallel tasks are nearly identical to when executed in isolation. Consequently, the only factor producing the slowdown is essentially the bandwidth limit itself. Therefore, in this case, the slowdown experienced by each processor can be conveniently controlled.

Conversely, operating at high utilization implies making memory work at or close to bandwidth saturation. However, in this regime, memory behavior becomes non-linear. The slowdown for a specific processor changes if other processor begin accessing memory and varies based on the types of memory accesses and bandwidth limits of the same processor and the competing ones. Several factors contribute to this behavior, such as varying conflicts on different memory banks and interconnect components, or contention affecting the cache hierarchy at the interconnect level. [4] Ultimately, slowdowns become complicated to predict and impossible to control through any static assignment of bandwidth limits. For these reasons, memory limitation mechanisms in real-time scenarios are typically employed conservatively, aiming to maintain memory bandwidth well below saturation. However, in such a configuration, memory bandwidth is once again underutilized, similar to the previous case.

Closing the gap with dynamic parallel access

How can we reconcile high predictability in system behavior with efficient memory bandwidth exploitation? In other words, how can we guarantee bounded slowdowns while fully utilizing memory bandwidth? In this paper, we propose a general solution to achieve such a goal. It builds on two main ingredients.

PREM task model. We adopt a PREM-like real-time task model, featuring a task set with per-task deadlines and a processor set where tasks are to be executed, and memory limits can be enforced. PREM-like tasks are identified by memory phases during which they execute contiguous memory accesses. In our model, we assume that memory phases can fall into two types: data prefetch, where only reads are executed, or data writeback, where only writes are performed. This is similar to other models such as the Logical Execution Time (LET) model of task execution, which distinguishes logical timing requirements from the actual physical platform execution. In the LET model, a task is sequential code with its own memory space and lacks internal synchronization points [8].

Dynamic memory policy. We assume to be given an *execution policy* that provides task allocation and scheduling – at any time, it assigns tasks to processors of interest. We do not introduce any further hypothesis on the execution policy, except for the fact it has to be memory-agnostic, i.e. allocation and scheduling choices must not depend on any memory concept like limits, contention or service times. We introduce a second policy, called the *memory policy*, which is responsible for the dynamic adjustment of the bandwidth limits of all active processors. This policy is a function of how many processors are accessing memory at the same time, and of which memory accesses they are performing between reads and writes. Limits are adjusted in such a way that the slowdowns experienced by each task are low enough to let the task still meet its deadlines, under the scheduling policy at hand.

The assumption regarding PREM-like tasks represents an important simplifying hypothesis as it enlarges the granularity of memory regulation decisions to a tractable size. Indeed, it costs time to detect a change in the memory access pattern of processors, compute new limits, and initiate the enforcement of these new limits. The highest time constant in play is the communication delay between the processors. For instance, in the case of a single CPU, which has lower delays compared to a heterogeneous system, we can estimate the delay to be in the order of 10 microseconds, assuming the communication is triggered by an inter-processor interrupt [15]. For this reason, our solution is feasible only for PREM-like tasks whose memory phases are longer than this base delay. For non-PREM tasks, it would

be challenging or even impossible to determine the type of each generic memory access right before it happens and to dynamically adjust bandwidth limits for each instantaneous change in the type of memory accesses. Nevertheless, other dynamic approaches could be considered, such as measuring per task/processor bandwidths online and adjusting limits on the fly. However, such extensions are beyond the scope of this initial proposal.

Given the PREM task set and a system where delays make our solution feasible, the core challenge lies in computing the bandwidth limits. All potential combinations of parallel access patterns must be considered, and their number grows exponentially with the number of processors and available limit values. However, only a limited set of configurations that drive memory bandwidth close to saturation are of interest. The number of these configurations is significantly lower than the total number of possible combinations. Therefore, a critical feasibility aspect is defining an algorithm that discards all useless configurations, eliminating the need to store them all and potentially avoiding their evaluation altogether.

We remark that, by construction,we structured the problem to maintain orthogonality between the execution and memory policies. This provides complete freedom in defining the execution policy. Specifically, one can establish either a global or a partitioned task scheduling policy with regulated parallel accesses to memory. A partitioned scheme is likely the preferred option for an initial solution, as it is typically characterized by simpler analysis and implementation.

2 System Model

This work focuses on a multi-processor system featuring a shared last-level cache and shared memory. The proposed idea in this work does not hinge on any specific task model or scheduling policy. Nevertheless, for illustrative purposes, the following system model is employed to present the idea. Each processor is assigned a partitioned subset of tasks. Tasks within the system follow a sequence: they prefetch data in the Read-memory phase, perform computations in a single computation phase, and write back the results in the Write-memory phase. On each processor, tasks are scheduled based on a dynamic-priority, non-preemptive policy.

2.1 Processors

The main memory is shared among m identical processors, each assigned a distinct static priority. These priorities govern the allocated bandwidth for parallel access to the main memory. A detailed explanation of how these priorities can be utilized to govern the parallel memory accesses will be provided in Section 4. Processors are indexed following their priorities, with P_1 possessing the highest priority and P_m the lowest.

2.2 Tasks

This work considers a partitioned system where each task is statically assigned to a single processor. The assignment of the i-th task to the k-th processor is denoted by the notation $\tau_i \in P_k$. Each task, denoted as τ_i exhibits a dynamic behavior by releasing an infinite sequence of jobs sporadically. Each individual job within this sequence is subject to a specified minimum inter-arrival time denoted as T_i. Each job of τ_i must be executed and completed within a fixed time limit from its release, specified by D_i, the relative deadline. We employ a PREM-like task model to simplify governing the memory requests. This assumption is significant because it increases the granularity of memory policing decisions to

a manageable level. Detecting changes in memory access patterns, calculating new limits, and enforcing these limits all take time. Due to this constraint, our solution is practical only for tasks resembling PREM, where the duration of memory phases exceeds the base delay. For non-PREM tasks, accurately predicting the type of each generic memory access just before it occurs and dynamically adjusting bandwidth limits for every instantaneous change in memory access type would be challenging, if not impossible. In this PREM-like model, each task consists of three phases: a read-memory phase, a computation phase, and a write-memory phase. During the read-memory phase, the task prefetches data from the main memory. In the computation phase, the task performs computations exclusively on the prefetched data without making any requests to access the main memory. The result is then written back to the main memory during the write-memory phase. In both memory phase, the tasks executes only memory accesses.

Tasks are scheduled on each processor according to a dynamic-priority non-preemptive scheduling policy and are indexed by priority, with τ_1 having the highest priority and τ_n the lowest, where n is the number of tasks allocated to the processor under study.

2.3 Memory

Memory functions as a globally shared resource, accessible to all processors with identical memory access latencies. In this model, the processors are symmetric, and they have the potential to saturate the bandwidth, meaning their cumulative demand may exceed the total memory bandwidth. Consequently, due to memory interferences, the duration extension of memory phases for each processor becomes unpredictable without regulation. Additionally, we assume that the order in which the memory controller serves memory requests simultaneously issued by different processors is unknown.

3 The Scheduling Policy

In this section, we present a partitioned memory-centric scheduling policy and illustrate it with an example. We will delve into the basic idea and the principal rules of the proposed policy. Our applied scheduling policy consists of two main components: the *execution policy* to distribute the task set among the processors and to schedule the execution order of each subset of the task set on each processor, and the *memory policy* for regulating the bandwidth to control access to the main memory. These two parts are explained as follows.

Execution Policy. Each task in the task set is statically assigned to a single processor, and no migration is allowed. The task set is first sorted based on their relative deadlines. Subsequently, tasks are assigned one by one to the processors according to their priority. Assuming there are n_T tasks in the task set and m processors in the system, the task with the closest relative deadline is assigned to the highest priority processor, denoted as P_1. The task with the second closest relative deadline is assigned to P_2, and this assignment continues until each processor is assigned a task. Once every processor has a task, the $m + 1^{th}$ closest relative deadline task is assigned to P_1 again. This procedure repeats until all tasks in the task set are assigned to processors. On each processor, tasks run based on a non-preemptive Earliest Deadline First (EDF) algorithm. Considering that all the required data for the execution of each task is prefetched during the read-memory phase, using a preemptive scheduling policy may increase the response time of each task by prefetching the same data several times. Therefore, choosing a non-preemptive policy can be more efficient. Moreover, by applying the EDF, the system can enjoy the benefits of dynamic task priorities.

Memory Policy. All the memory requests are governed globally through bandwidth regulation. When a new memory request arrives, or an existing memory access finishes, depending on the number of parallel memory accesses, and the workload on the targeted processor and the interfering ones, the supervisor should check the corresponding cell in the table of regulation factors($\mathcal{RF}$) to identify the amount of bandwidth allocated to each active processor.

4 Implementing The Policy

4.1 Memory bandwidth regulation

4.1.1 Regulation mechanisms

Memory bandwidth limits are realized by *memory regulation mechanisms* that can be implemented in different ways – they can be either hardware assisted, or provided by software components, either external from, or internal to, the workload to be limited.

HW regulation. Hardware regulation is commercially available with Memory Bandwidth Allocation, part of the Resource Director Technology by Intel [9], that is mainly featured on higher end Xeon family processors. A similar technology is provided also by Arm with the Memory System Resource Partitioning and Monitoring (MPAM) set of IPs [2], but we are not aware of any commercially available chip including it. The common principle is that CPU cores can be assigned with a given quality of service for memory transactions, and that such limit is enforced by regulating the traffic originated from the last-level cache or the system-level cache.

External SW regulation. Software regulation can be conveniently offered by a component of the platform software. It can be the case of the operating system like in MemGuard [21], the hypervisor like in MinervaSys Jailhouse [6] or some system-resident firmware like in MemPol [22]. The underlying principle is the same – a processor is allotted a predefined maximum number of memory accesses (budget) to execute within a fixed period. Should the processor exhaust its budget before the period concludes, the regulation mechanism halts the processor until the period concludes, at which point the processor receives a new budget for the subsequent period.

Internal SW regulation. This method, also called Voluntary Throttling (VolT), entails augmenting the code of the memory phases of the PREM tasks so to introduce a number of NOPs (No Operation instructions) periodically during its memory phase(s) [4]. The number of NOPs is externally configurable and provides the mean to regulate the throttling length (or frequency) of throttling. The code augmentation can be inserted at compile time, which is especially convenient when code PREMization is already automated [7].

4.1.2 Regulation factors

In order of us to abstract from the implementation details, we define the notion of regulation factor, that acts as a knob to adjust the bandwidth allocation for a process. A regulation factor of 0% indicates that no bandwidth is allocated for the service, while a factor of 100% implies that the entire available bandwidth is dedicated to the service.

▶ **Definition 1** (Regulation Factor (RF)). *For each processor configured with a memory limit, we define its regulation factor (RF) as the ratio between the* limited memory bandwidth *measured in isolation by performing back-to-back Reads (or Writes) operations, and the* (unlimited) memory bandwidth *measured in the same conditions while removing the memory limitation on the processor. For the sake of simplicity, we assume RFs to range among percentage integers between* 0 *(no bandwidth) to* 100 *(full bandwidth).*

We remark that the target measurement is performed in isolation because the attainable bandwidth is, in general, influenced by the number of processors operating in parallel, and the workload on both the affected processor and any interfering processors. Consequently, let us also stress that the actual bandwidth experienced by a processor is influenced, not entirely determined, by its regulation factor.

Observe that the regulation primitives greatly differ among the various memory regulation mechanisms. A concrete manner is thus needed to compute the function that maps any configuration of the mechanisms-specific knobs to its corresponding regulation factors. In most cases, only an experimental method enables to obtain a precise definition, as the one provided in Subsubsection 4.1.3 for VoLT.

4.1.3 Example: VOLT regulation factors

In a VoLT system the only available knob is the number of NOPs injected in the code. To compute the RFs, we need to experimentally find, for each workload, for each kind of processor, and for each regulation factor, the number of NOPs that produce the limited bandwidth of our interest. For instance, when the regulation factor is 10%, it means that the number of NOPs is such that if the processor executes this specific workload (Read or Write) in isolation, it receives 10% of the unlimited bandwidth. This table can be constructed experimentally following the algorithm outlined in Algorithm 1.

For each memory access type, the algorithm initializes two counters: N_{old} and N_{new}, representing the initial and current number of NOPs, respectively. Then, it sweeps through a range of RF values from 100 to 0 with a step of -10, representing increasing regulation levels. At each RF iteration, a $Match$ flag is set to false, indicating that a precise match

Algorithm 1 Construct the translating table between regulation factors and limited bandwidth values.

Input : Unlimited bandwidth values UnB_R, UnB_W for Read and Write memory accesses
Output : Translating table $\mathcal{RF}2\mathcal{BW}$ between regulation factors and actual bandwidth values:

```
1  foreach TMR ∈ {R, W} do
2      N_old = 1
3      N_new = 2
4      for RF = 100 to 0 with step −10 do
5          Match = False
6          while !Match do
7              BW_old = measure the bandwidth with N_old NOPs
8              BW_new = measure the bandwidth with N_new NOPs
9              if |BW_new − RF/100 × UnB_TMR| < |BW_old − RF/100 × UnB_TMR| then
10                 N_old++
11                 N_new++
12             else
13                 N(TMR, RF) = N_old
14                 Match = True
15 return Result
```

between RF and bandwidth has not yet been established. The algorithm then enters a loop that repeatedly measures bandwidth using two different NOP configurations. The first measurement (BW_{old}) reflects the current NOP count (N_{old}), while the second measurement (BW_{new}) employs N_{old} incremented by one ($N_{new} = N_{old} + 1$).

The algorithm compares the absolute differences between the measured bandwidth values and the expected bandwidth value for each RF, calculated by multiplying RF by the unlimited bandwidth for the corresponding memory access type (UnB_{TMR}). If the difference for the newer NOP setting ($|BW_{new} - \frac{RF}{100} \times UnB_{TMR}|$) is smaller than that for the older setting ($|BW_{old} - \frac{RF}{100} \times UnB_{TMR}|$), it suggests that the newer NOP configuration provides a more accurate bandwidth estimation for that particular RF. In such a case, the algorithm increments both counters ($N_{old}++$, $N_{new}++$), effectively refining the bandwidth estimation resolution. The algorithm continues iterating within this loop until the Match flag is set to true, indicating that the optimal match between RF and bandwidth has been identified. Upon reaching this point, the algorithm stores the corresponding NOP count (N_{old}) in the translating table $\mathcal{RF}2\mathcal{BW}$ for the specified memory access type (TMR) and RF value. This process is repeated for both R and W memory access types to construct the complete translating table.

It's crucial to recognize that since the number of NOPs is limited to integer values, the bandwidth values derived from regulation factors are not entirely precise – they represent the closest approximation of the true bandwidth achievable with a specific regulation level.

4.2 Construct the table of slowdown measurements

The table of regulation factors ($\mathcal{RF}$) comprises factors for regulating memory accesses corresponding to each workload pattern. To fill in the cells of this table, a series of slowdown measurements must be conducted for every conceivable set of regulation factors on active processors. This enables us to deduce the factors that more effectively align with our timing constraints.

As these slowdown measurements merely pertain to the specifications of the hardware in use, rather than the actual task set, we optimize system performance by employing a separate table, designated as the *table of slowdown measurements ($\mathcal{SM}$)*, to store the recorded slowdown values for each scenario of memory accesses and a specific set of regulation factors on the active processors. The construction of the slowdown measurements table is a one-time necessity. Subsequently, based on the task set specifications, we can then select the set of suitable factors.

In the remainder of this section, we systematically introduce the algorithms for constructing the aforementioned tables, step by step. But first, we shall define some preliminary concepts.

4.2.1 Formal definitions

The set of notations used is briefly explained in the Table 1.

To determine the appropriate set of throttling factors for effective bandwidth regulation, it is essential to undertake a series of slowdown measurements encompassing all possible combinations of throttling factors for active processors. In this section, we aim to outline a systematic approach for conducting these measurements while avoiding redundant cases. But before presenting the algorithm a few concepts should be clarified.

Table 1 The table of notations.

Notation	Definition
$\mathbf{C}(j, \mathbf{S}_i, \mathbf{RF}(j))$	a generic configuration consists of a set of regulation factors,$\mathbf{RF}(j)$, its corresponding slowdown measurement$\mathbf{SD}(j, \mathbf{S}_i, \mathbf{RF}(j))$, and the actual bandwidth of processors, or **UB**
j	number of processors accessing the memory in parallel
m	total number of processors in the system
$N(TMR, RF)$	number of NOPs to achieve a specific regulation factor
RF_k	the regulation factor assigned to the $k-th$ highest priority processor among those accessing the memory at the same time
RF	a generic regulation factor to regulate the bandwidth
$\mathbf{RF}(j)$	a generic vector of regulation factors used to measure the slowdown values in case of j parallel memory accesses following $\mathbf{S}_i$ scenario corresponding to s measured slowdown value
$\mathcal{RF2BW}$	the translating table between regulation factors and actual bandwidth values
$\mathbf{S}_i$	each scenario of parallel memory requests
$\mathcal{S}(j)$	set of all possible scenarios of parallel memory accesses for each value of j
$\mathbf{SD}(j, \mathbf{S}_i, \mathbf{RF}(j))$	the vector of measured slowdown values corresponding to j parallel memory accesses following $\mathbf{S}_i$ scenario using $\mathbf{RF}(j)$
$\mathcal{SM}$	table of slowdown measurements
$\mathbf{SM}[\mathbf{S}_i(j), \mathbf{C}(j, \mathbf{S}_i), \mathbf{RF}(j)]$	sub-table of slowdown measurements corresponding to j parallel memory accesses following $\mathbf{S}_i$ scenario
UB	the vector of actual bandwidth of the processors
ε	a very small positive value

▶ **Definition 2** (Slowdown Measurement). *Processor slowdown measurement involves quantifying the reduction in the speed of a processor as it performs tasks. This measurement is typically expressed as a percentage decrease in processing speed compared to the processor's original performance. In the context of this discussion, slowdown measurements refer to the decrease in speed of processors when accessing the memory as a consequence of throttling the bandwidth.*

In practical scenarios, the memory phases of different processors may partially overlap. However, to consider the worst-case scenario, these measurements involve executing all conceivable combinations of memory phases in parallel continuously. In this scenario, the system encounters the most severe slowdown due to memory interference.

▶ **Definition 3** (Set of workload combinations($\mathcal{S}(j)$)). *For any given number j of parallel memory accesses, the set of all the scenarios of memory accesses, or in other words, the different patterns of Reads and Writes coming from different processors in parallel, is called the set of workload combinations and is denoted by $\mathcal{S}(j)$. Each scenario within this set is denoted by $\mathbf{S}_i$.*

For example, in the case of two parallel memory requests, there will be four distinct scenarios of memory accesses: R_hR_l, R_hW_l, W_hR_l, W_hW_l, where 'h' refers to the memory access of the higher-priority processor and 'l' to the lower-priority one. Therefore, $\mathcal{S}(2)$ will have, 2^2 elements. Following the same reasoning, in general $\mathcal{S}(j)$ includes 2^j elements.

▶ **Definition 4** (Configuration of parallel memory accesses($\mathbf{C}$)). *For any given number j of parallel memory accesses, and any scenario of memory accesses $\mathbf{S}_i$, a configuration of parallel memory accesses is a vector that concatenates a vector of regulation factors for the corresponding active processors* $\mathbf{RF}$*, its corresponding measured slowdown* $\mathbf{SD}(j, \mathbf{S}_i, \mathbf{RF})$*, and the vector of actual bandwidth of the active processors* $\mathbf{UB}(j, \mathbf{S}_i, \mathbf{RF})$*. Or*

$$\mathbf{C}(j, \mathbf{S}_i, \mathbf{RF}(j)) = [\mathbf{RF}(j); \mathbf{SD}(j, \mathbf{S}_i, \mathbf{RF}(j)); \mathbf{UB}(j, \mathbf{S}_i, \mathbf{RF}(j))]$$

▶ **Definition 5** (Slowdown measurements table($\mathcal{SM}$)). *Slowdown measurements table is a table of subtables, each denoted by* $\mathbf{SM}[\mathbf{S}_i(j), \mathbf{C}(j, \mathbf{S}_i), \mathbf{RF}(j)]$*, corresponding to each number of parallel memory accesses (*j*) and each scenario of memory access (*$\mathbf{S}_i$*). Each subtable is represented as a two-dimensional array, where the rows represent different scenarios of memory accesses(*$\mathbf{S}_i$*), and the columns represent different valid configurations of parallel memory accesses.*

4.2.2 Slowdown interplay

When employing each set of regulation factors, the amount of slowdown experienced by each processor may vary depending on the specific combination of processors concurrently accessing memory, the overall number of active processors, and the type of memory access (read or write) on both the targeted processor and its competitors. Since this study operates under the assumption of a homogeneous system, the identity of the processors simultaneously requesting memory is irrelevant; the only relevant factors are their relative priorities, and their total number, represented by j.

4.2.3 Algorithm core ideas

The algorithm to construct the slowdown measurement table receives the number of processors in the system as the input and outputs the table of slowdown measurements. For each number of parallel memory requests and each scenario, $\mathbf{S}_i \in \mathcal{S}(j)$, the algorithm commences by measuring the slowdown values and the actual bandwidth of all the active processors for the full-throttling case. Subsequently, starting from the processor with the lowest priority and progressing to the higher-priority ones, the throttling factor of each processor is reduced by 10 percent, and the measurements are then repeated to track every possible configuration of parallel memory accesses.

The main objective is to maximize bandwidth utilization, therefore we need to maintain bandwidth as close to saturation as possible avoiding over-allocation.

Within the saturation zone, the relationship between utilized bandwidth and throttling factors is non-linear. Counter-intuitively, adjusting the throttling factor of one processor can lead to fluctuations in the actual bandwidth values of other processors. For instance, decreasing the throttling factor of one processor might increase the actual bandwidth of other processors. On the opposite end, in the underutilization zone, the bandwidth is out of saturation. Therefore, the behavior of the system becomes essentially linear. In this zone, decreasing the regulation factors of one processor will either decrease or maintain the bandwidth of that processor without affecting the bandwidth of other processors. Considering this behavior, we can develop a discarding technique to improve the efficiency of the algorithm for storing slowdown measurements. This technique involves retaining only configurations close to saturation.

4.2.4 Discarding configurations

There are two primary reasons for discarding a configuration: either when we have already identified a better configuration in terms of bandwidth utilization in the saturation zone, or when the current configuration results in under-utilization of the available bandwidth.

If reducing the regulation factor results in negligible changes to the measured slowdown, it implies that the previous configuration was saturating the bandwidth. Consequently, we can discard the previous configuration and retain the current one. Given the uniform decrease of

the regulation factors, this comparison can be conducted for every two consecutive measurements to eliminate configurations that result in over-allocation of bandwidth. Conversely, if a regulation requires less bandwidth than an already existing configuration for the same workload combination, it indicates that this configuration does not provide us with a better solution and there is no need to retain it. Considering the gradual reduction of regulation factors, this can be examined by comparing the actual bandwidth of all the processors in two consecutive measurements. If none of the processors gain a better bandwidth, it means this configuration under-utilizes the bandwidth. Therefore, we should discard it.

4.2.5 Termination

By the following termination rule, we can determine whether continuing the measurement will yield beneficial results or if we can terminate it. If modifying a single regulation factor solely affects the bandwidth of the corresponding processor, while the bandwidth of all other processors remains unchanged, it indicates that there is no interference between memory accesses, and the system is operating in the under-utilization zone. Therefore, we can terminate the current loop and proceed to the next outer loop. This nested table structure provides a concise and organized representation of the system's slowdown values, enabling efficient retrieval and analysis of the impact of regulation factors.

4.3 Construct the table of regulation factors

Given that slowdown values are influenced by the selected regulation factors, we can control slowdown values by adjusting these factors. With a predefined maximum tolerable slowdown value, aligned with the timing constraints of the task set, we can choose regulation factors to facilitate parallel memory accesses. This concept can be implemented through a table, which provides the appropriate set of Regulation factors for each scenario of parallel memory accesses under predefined slowdown constraints.

▶ **Definition 6** (Table of Regulation factors ($\mathcal{RF}$)). *The table $\mathcal{RF}$ is structured as a collection of sub-tables. For each number of parallel memory accesses, denoted as j, there exists a sub-table. Within each sub-table, for every possible scenario of memory accesses, it encapsulates the set of Regulation factors for active processors, tailored to meet the slowdown constraints imposed by the task set.*

To present the algorithm to construct this table, we should clarify a few notations.

▶ **Definition 7** (Possible Configurations ($\mathcal{PC}$)). *The set of candidate configurations corresponding to j parallel memory accesses with workload combination $\mathbf{S}_i$ is shown by $\mathcal{PC}(j, \mathbf{S}_i)$.*

▶ **Definition 8** (Maximum Bandwidth Utilization ($\mathcal{MBU}$)). *The set of regulation factors' vectors with the same maximum bandwidth utilization by $\mathcal{MBU}(j, \mathbf{S}_i)$.*

This algorithm receives as inputs: the table of slowdown measurements($\mathcal{SM}$), and the set of maximum tolerable values for slowdowns in line with the timing constraints of the task set, denoted by $\mathcal{MTS}$. It provides as output the regulation factors table $\mathcal{RF}$. Alongside this algorithm, for each number of parallel memory accesses, first, the set of all possible scenarios of parallel memory requests, or $\mathcal{S}(j)$, is generated. Then for each scenario, all the configurations stored at sub-table $\mathbf{SM}[\mathbf{S}_i(j), \mathbf{C}(j, \mathbf{S}_i)]$ are compared to their corresponding maximum tolerable value in $\mathcal{MTS}$. If the measured slowdown value is smaller or slightly larger than the maximum tolerable value, the algorithm appends this configuration to $\mathcal{PC}(j, \mathbf{S}_i)$.

Algorithm 2 Construct the slowdown measurements table.

Input : number m of processors
Output : table $\mathcal{SM}$ of slowdown measurements

for $j = 2 : m$ **do**
  Generate: $\mathcal{S}(j)$
  foreach $\mathbf{S}_i \in \mathcal{S}(j)$ **do**
    $\mathbf{RF}_{\text{old}} = [100, 100, \ldots, 100]$
    Perform slowdown and used bandwidth measurements for unlimited bandwidth case $\mathbf{SD}(j, \mathbf{S}_i, \mathbf{RF}_{\text{old}})$, $\mathbf{UB}$
    $\mathbf{SD}_{\text{old}} = \mathbf{SD}(j, \mathbf{S}_i, \mathbf{RF}_{\text{old}})$
    $\mathbf{UB}_{\text{old}} = \mathbf{UB}$
    for $RF_1 = 100$ *to* 0 *with step* -10 **do**
      for $RF_2 = RF_1$ *to* 0 *with step* -10 **do**
        ...
        for $RF_{j-1} = RF_{j-2}$ *to* 0 *with step* -10 **do**
          $UnderUtilization = \text{False}$
          $RF_j = RF_{j-1}$
          while $!UnderUtilization$ **do**
            $\mathbf{RF}_{\text{new}} = [RF_1, RF_2, \ldots, RF_j]$
            Perform measurements $\mathbf{SD}(j, \mathbf{S}_i, \mathbf{RF}_{\text{new}}), \mathbf{UB}$
            $\mathbf{SD}_{\text{new}} = \mathbf{SD}(j, \mathbf{S}_i, \mathbf{RF}_{\text{new}})$
            $\mathbf{UB}_{\text{new}} = \mathbf{UB}$
            `// over-utilization check:`
            if $\mathbf{SD}_{new} \approx \mathbf{SD}_{old}$ **then**
              $\mathbf{C}(j, \mathbf{S}_i, \mathbf{RF}_{\text{old}}) = \mathbf{C}(j, \mathbf{S}_i, \mathbf{RF}_{\text{new}})$
              $\mathbf{RF}_{\text{old}} = \mathbf{RF}_{\text{new}}$
              $\mathbf{SD}_{\text{old}} = \mathbf{SD}_{\text{new}}$
              $\mathbf{UB}_{\text{old}} = \mathbf{UB}_{\text{new}}$
            `// non-optimal solution check:`
            if $\mathbf{UB}_{new} \leq \mathbf{UB}_{old}$ **then**
              Discard the new configuration
            `// termination condition:`
            if $(\mathbf{UB}_{new}[1 : j-1] \approx \mathbf{UB}_{old}[1 : j-1]$ && $\mathbf{UB}_{new}[j] \leq \mathbf{UB}_{old}[j])$ $\|$ $RF_j == 0$ **then**
              $UnderUtilization = \text{True}$
            else
              $RF_j -= 10$
return *Result*

Whenever $\mathcal{PC}(j, \mathbf{S}_i)$ is empty, the algorithm reports: **"Not enough bandwidth"**, which means we should suspend the memory access coming from the lowest priority processor to make sure this case will not happen. However, if $\mathcal{PC}(j, \mathbf{S}_i)$ is not empty, the algorithm looks for the configuration that yields the maximum used bandwidth(or **UB**). If this solution is not unique, among them, this algorithm picks the one with the maximum regulation factors.

Algorithm 3 Construct the table of regulation factors.

Input : Number m of processors, set $\mathcal{MTS}$ of maximum tolerable values for slowdowns in line with the timing constraints of the task set, table $\mathcal{SM}$ of slowdown measurements

Output : Table $\mathcal{RF}$ of regulation factors

for $j = 2 : m$ **do**
  Generate: $\mathcal{S}(j)$
  for $\forall \mathbf{S}_i \in \mathcal{S}(j)$ **do**
    Move to sub-table $\mathbf{SM}[\mathbf{S}_i(j), \mathbf{C}(j, \mathbf{S}_i)]$ **in** $\mathcal{SM}$
    for $\forall \mathbf{C} \in \mathbf{SM}[\mathbf{S}_i(j), \mathbf{C}(j, \mathbf{S}_i)]$ **do**
      if $\mathbf{SD} \leq (\mathcal{MTS}(j, \mathbf{S}_i) + \varepsilon)$ **then**
        Append C to $\mathcal{PC}(j, \mathbf{S}_i)$
    if $\mathcal{PC}(j, \mathbf{S}_i) \neq \emptyset$ **then**
      $MBU(j, \mathbf{S}_i) = \max_{\mathbf{C} \in \mathcal{PC}(j, \mathbf{S}_i)}$ **UB**
      $\mathcal{MBU}(j, \mathbf{S}_i) = \{\mathbf{C} \in \mathcal{PC}(j, \mathbf{S}_i) | \mathbf{UB} \approx MBU(j, \mathbf{S}_i)\}$
      if $\mathcal{MBU}(j, \mathbf{S}_i) \neq \emptyset$ **then**
        $\mathcal{RF}(j, \mathbf{S}_i) = \max_{\mathbf{C} \in \mathcal{MBU}(j, \mathbf{S}_i)}$ **RF**
    if $\mathcal{PC}(j, \mathbf{S}_i) = \emptyset$ **then**
      Report: "Not enough bandwidth"
return *Result*

4.4 An illustrative example

To illustrate the policy, the following example can provide clarity. Let's consider a task set as described in the table below. All tasks in this set are activated synchronously. Our objective is to efficiently schedule this task set on a system with three processors. These tasks are prioritized based on their periods and scheduled accordingly. We also assume that no preemption is allowed. In table Table 2 the duration of the read memory phase, write memory phase, computation time, period(or minimum inter-arrival time), and the relative deadline of task τ_i are denoted by, rm_i, wm_i, c_i, T_i, and D_i, respectively. All the values are measured in microseconds:

Table 2 Task Set Characteristics.

τ_i	$rm_i(\times 10\mu s)$	$c_i(\times 10\mu s)$	$wm_i(\times 10\mu s)$	$T_i(\times 10\mu s)$	$D_i(\times 10\mu s)$
τ_1	3	4	2	20	10
τ_2	2	3	1	25	15
τ_3	2	2	1	30	20
τ_4	1	1	1	35	25
τ_5	1	4	1	40	30

Following the partitioning policy, τ_1, and τ_4 are assigned to P_1, τ_2, and τ_5 to P_2, and τ_3 is assigned to P_3. In Gantt charts, red, grey, and blue blocks represent the read-memory phase, the computation phase, and the write-memory phase, respectively. Initially, we assume no parallel memory access is allowed, limiting each processor to accessing memory one at a time according to their priorities. Tasks are partitioned using the same policy, and on each processor, tasks are executed using a non-preemptive Earliest Deadline First (EDF) scheduling algorithm.

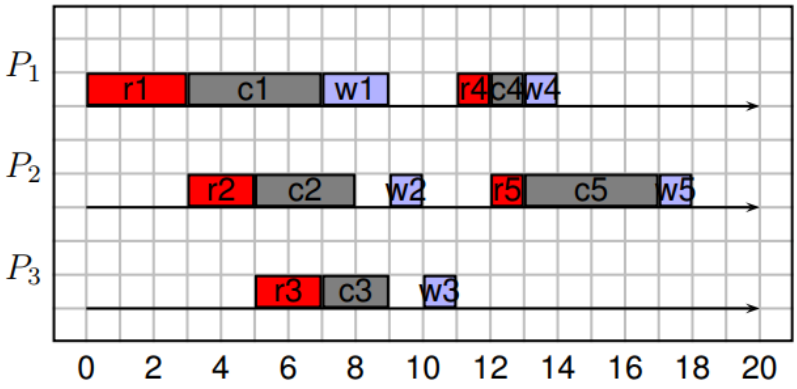


Figure 1 Partitioned scheduling algorithm, no parallel memory access.

Next, we aim to schedule the identical task set employing our policy. Broadly speaking, the asynchronous execution capability of the write memory phase renders it faster than the read memory phase. Therefore, we consider distinct values for unlimited bandwidth in read and write operations. As an illustrative numerical example, let's assume the system has an unlimited bandwidth of 2.5 GB/s for read-memory phases and 8 GB/s for write-memory phases. The regulation factors for two and three parallel memory accesses are outlined in Table 3 and Table 4, respectively. In practice, these tables must be filled following the algorithms.

Table 3 Table of regulation factors for two parallel memory accesses.

Workload pattern	Regulation factors for P_h	Regulation factors for P_l
RR	100	100
RW	90	70
WR	80	60
WW	60	40

Table 4 Table of regulation factors for three parallel memory accesses.

Workload pattern	Regulation factors for P_1	Regulation factors for P_2	Regulation factors for P_3
RRR	100	100	100
RRW	90	70	50
RWR	80	60	40
WRR	70	50	40
RWW	80	40	30
WRW	60	30	30
WWR	60	30	30
WWW	60	30	10

According to the tables of regulation factors, the Gann chart will be:

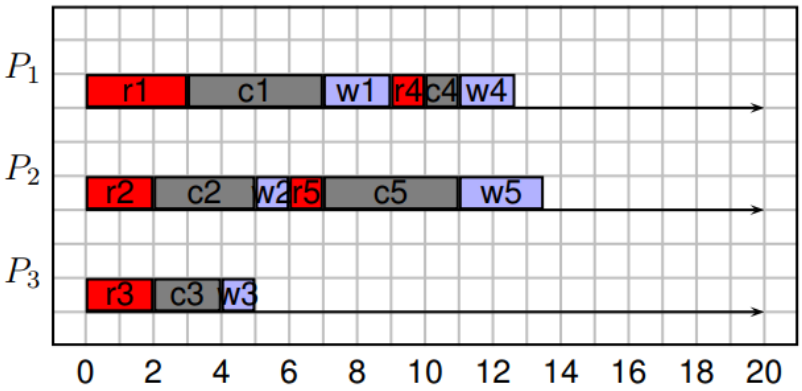


Figure 2 Partitioned scheduling algorithm, using HMB.

As illustrated in the example, three read-memory phases can be executed in parallel with negligible extension of the memory phase duration. However, when two write-memory phases access the memory in parallel (as is the case with w_4 and w_5), according to the table, 60% of the unlimited bandwidth of the processor will be allocated to the higher priority processor, and 40% to the lower priority one. Consequently, the duration of these memory phases will be extended accordingly. Nevertheless, upon comparing the Gantt charts, there is a remarkable improvement in the overall time-span of the task set following our proposed policy.

5 Related Works

The impact of memory contention in contemporary systems has been extensively explored in prior scientific literature. [10] Previous studies have focused on investigating the decline in Worst-Case Execution Time (WCET) for applications contending for memory, particularly in multi-core embedded systems [13]. Proposals for memory-bandwidth partitioning schemes aimed at ensuring temporal isolation have been introduced [12]. In [21], the authors introduced a memory bandwidth reservation system named MemGuard. This system was proposed, designed, and implemented with the primary aim of providing bandwidth reservation to ensure temporal isolation, and maximizing the utilization of the reserved bandwidth. In [14], the memory utilization is periodically sampled, While using standard MemGuard's interrupts – and associated overheads – to regulate cores and to trigger the sampling. While partitioning represents a straightforward and robust solution, it encounters challenges related to underutilizing the bandwidth. Moreover, it offers less refined control over task execution compared to the PREM approach. Similar challenges are observed in alternative hardware-level partitioning solutions and mechanisms for enforcing bandwidth allocation documented in existing literature [5, 18]. As an example in [22], known as MemPol, in introduced that operates a regulation mechanism from outside the cores, monitoring performance counters for the application core's activity in main memory at a microsecond scale. In contrast, our work adopts an internal mechanism to regulate the bandwidth offering a more flexible scheme to maximize bandwidth utilization. A substantial body of literature addresses the application of PREM model to multi-core systems [1, 3, 17, 19]. However, a primary limitation of these studies is their restriction to permitting only one memory access at a time, leading to bandwidth underutilization. In [20], the authors extended PREM by accommodating more

than one task to access memory in parallel. Through experimentation, they demonstrated that the latency of main-memory accesses increases at a rate less than linear when multiple cores simultaneously access memory. Their model supports k parallel memory accesses, where k is a statically configurable number determined based on hardware specifications. The primary drawback of this model lies in its rigidity. As shown in [4], allowing k cores to utilize bandwidth without constraint, whether needed or not, may lead to bandwidth overutilization while selecting $k-1$ could result in bandwidth underutilization. Addressing this issue, the main advantage of our model lies in the dynamic allocation of bandwidth to processors based on workload patterns. This allows for the adaptive selection of the number of parallel memory accesses, optimizing resource utilization. Despite differences in scope, a comparable work to ours is [16], which introduces the Envelope-aWare Predictive model, abbreviated as E-WarP. It aims to provide both the technological foundations and theoretical bases for a workload-aware analysis of real-time systems.

6 Conclusion

6.1 Discussion

Contemporary homogeneous and heterogeneous computing systems attain enhanced performance levels through parallelization. However, the parallel execution of tasks introduces complex trade-offs between latency and throughput, particularly in the context of simultaneous accesses to shared memory. Numerous approaches aim to mitigate memory interference issues, with bandwidth regulation being popular. In the literature, various viable solutions for bandwidth provide basic mechanisms to address the latency of memory accesses. However, their primary drawbacks include the complexity of deployment and rigidity. It has been observed that existing solutions may lead to underutilization of the available bandwidth.

The challenge lies in the non-linear behavior of memory latency based on the number and type of concurrent accesses, which can fluctuate over time in an unpredictable manner. Past attempts to integrate memory regulation into scheduling solutions have, as a consequence, either failed to provide guarantees for real-time execution or led to significant underutilization of memory bandwidth.

In this paper, we introduce High Memory Bandwidth (HMB), a scheduling solution designed to guarantee bounded response times of real-time task sets through memory regulation while ensuring a high utilization of memory bandwidth. The intricate nature of both the problem and its potential solution necessitates a comprehensive approach, one that this preliminary work only begins to unfold. In this first step, we focus on the core mechanism of HMB, providing a detailed explanation of its inner workings and how it addresses the challenges posed by real-time task sets. Our goal is to lay a solid foundation for future research and development, paving the way toward a scheduling solution that seamlessly integrates the demands of real-time systems with the efficient utilization of memory resources.

The core concept of this work lies on the notion memory slowdown, a phenomenon that occurs when a processor's memory access performance is hindered by the concurrent memory access patterns of other processors. This slowdown, particularly during bursts of back-to-back memory accesses, can significantly impact the execution time of real-time tasks, potentially violating their timing constraints. To address this challenge, HMB employs a novel mechanism that dynamically allocates memory bandwidth among processors based on their priority levels. By carefully balancing the needs of high-priority processors, which typically have stricter timing requirements, HMB ensures that their memory accesses are prioritized, minimizing slowdown and maintaining their responsiveness.

HMB's efficiency stems from its ability to optimize memory bandwidth utilization while adhering to the priority-based allocation scheme. It continuously evaluates the system's memory access patterns and dynamically adjusts bandwidth caps to accommodate the demands of high-priority processors without compromising overall efficiency. This intricate balance enables HMB to achieve both bounded response times for real-time tasks and high memory bandwidth utilization, a remarkable feat in the context of resource-constrained real-time systems.

6.2 Further works

In this short work-in-progress paper, our primary focus has been on expounding the core concept of HMB and its underlying mechanism for bandwidth regulation. The next crucial step involves conducting a comprehensive series of experiments to rigorously validate the feasibility and evaluate the efficiency of this proposed mechanism in real-world scenarios.

During the implementation phase, potential overheads can arise at multiple levels, demanding careful consideration and optimization. At the hardware level, we must carefully evaluate the size of tables required to effectively implement HMB and ensure that data transfer speeds are sufficient to support the proposed bandwidth allocation scheme. Additionally, we need to analyze the overhead introduced by the algorithm and the dispatcher at the execution level. To ensure feasibility, it is imperative that the overall overhead remains lower than the shortest memory phase.

To objectively assess the effectiveness of HMB, we can compare its performance to similar works in this domain, such as [20] and [16]. By comparing against these established solutions, we can gain valuable insights into the relative strengths and weaknesses of HMB, paving the way for further refinements and improvements.

References

1 Ahmed Alhammad, Saud Wasly, and Rodolfo Pellizzoni. Memory efficient global scheduling of real-time tasks. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 285–296, 2015. doi:10.1109/RTAS.2015.7108452.

2 Arm. *MPAM Architecture Reference Supplement*, 2022. URL: https://developer.arm.com/documentation/ddi0598/latest.

3 Stanley Bak, Gang Yao, Rodolfo Pellizzoni, and Marco Caccamo. Memory-aware scheduling of multicore task sets for real-time systems. In *Proceedings of the 2012 IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, RTCSA '12, pages 300–309, USA, 2012. IEEE Computer Society. doi:10.1109/RTCSA.2012.48.

4 Gianluca Brilli, Roberto Cavicchioli, Marco Solieri, Paolo Valente, and Andrea Marongiu. Evaluating controlled memory request injection for efficient bandwidth utilization and predictable execution in heterogeneous socs. *ACM Trans. Embed. Comput. Syst.*, 22(1), December 2022. doi:10.1145/3548773.

5 Jon Perez Cerrolaza, Roman Obermaisser, Jaume Abella, Francisco J. Cazorla, Kim Grüttner, Irune Agirre, Hamidreza Ahmadian, and Imanol Allende. Multi-core devices for safety-critical systems: A survey. *ACM Comput. Surv.*, 53(4), August 2020. doi:10.1145/3398665.

6 Jailhouse commmunity, Technical University of Munich, Università di Modena e Reggio Emilia, Boston University, and Minerva Systems SRL. MinervaSys Jailhouse, 2019–2023. URL: https://gitlab.com/minervasys/public/jailhouse.

7 Björn Forsberg, Marco Solieri, Marko Bertogna, Luca Benini, and Andrea Marongiu. The predictable execution model in practice: Compiling real applications for cots hardware. *ACM Transactions on Embedded Computing Systems (TECS)*, 20(5):1–25, 2021.

8 Arkadeb Ghosal. *A Hierarchical Coordination Language for Reliable Real-Time Tasks*. PhD thesis, EECS Department, University of California, Berkeley, January 2008. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-10.html.

9 Intel. *Resource Director Technology Refrence Manual*, 2019.

10 Tamara Lugo, Santiago Lozano, Javier Fernández, and Jesus Carretero. A survey of techniques for reducing interference in real-time applications on multicore platforms. *IEEE Access*, 10:21853–21882, 2022. doi:10.1109/ACCESS.2022.3151891.

11 Jan Nowotsch, Michael Paulitsch, Daniel Buhler, Henrik Theiling, Simon Wegener, and Michael Schmidt. Multi-core interference-sensitive wcet analysis leveraging runtime resource capacity enforcement. *2014 26th Euromicro Conference on Real-Time Systems*, pages 109–118, 2014. URL: https://api.semanticscholar.org/CorpusID:12573936.

12 Jan Nowotsch, Michael Paulitsch, Daniel Bühler, Henrik Theiling, Simon Wegener, and Michael Schmidt. Multi-core interference-sensitive wcet analysis leveraging runtime resource capacity enforcement. In *2014 26th Euromicro Conference on Real-Time Systems*, pages 109–118, 2014. doi:10.1109/ECRTS.2014.20.

13 Rodolfo Pellizzoni, Andreas Schranzhofer, Jian-Jia Chen, Marco Caccamo, and Lothar Thiele. Worst case delay analysis for memory interference in multicore systems. In *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pages 741–746, 2010. doi:10.1109/DATE.2010.5456952.

14 Ahsan Saeed, Dakshina Dasari, Dirk Ziegenbein, Varun Rajasekaran, Falk Rehm, Michael Pressler, Arne Hamann, Daniel Mueller-Gritschneder, Andreas Gerstlauer, and Ulf Schlichtmann. Memory utilization-based dynamic bandwidth regulation for temporal isolation in multi-cores. In *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 133–145, 2022. doi:10.1109/RTAS54340.2022.00019.

15 Gero Schwäricke, Tomasz Kloda, Giovani Gracioli, Marko Bertogna, and Marco Caccamo. Fixed-priority memory-centric scheduler for cots-based multiprocessors. In *Euromicro Conference on Real-Time Systems*, 2020. URL: https://api.semanticscholar.org/CorpusID:220275158.

16 Parul Sohal, Rohan Tabish, Ulrich Drepper, and Renato Mancuso. E-warp: A system-wide framework for memory bandwidth profiling and management. In *2020 IEEE Real-Time Systems Symposium (RTSS)*, pages 345–357. IEEE, 2020.

17 Muhammad R. Soliman and Rodolfo Pellizzoni. PREM-Based Optimal Task Segmentation Under Fixed Priority Scheduling. In Sophie Quinton, editor, *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, volume 133 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:23, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ECRTS.2019.4.

18 Prathap Kumar Valsan, Heechul Yun, and Farzad Farshchi. Taming non-blocking caches to improve isolation in multicore real-time systems. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–12, 2016. doi:10.1109/RTAS.2016.7461361.

19 Gang Yao, Rodolfo Pellizzoni, Stanley Bak, Emiliano Betti, and Marco Caccamo. Memory-centric scheduling for multicore hard real-time systems. *Real-Time Systems*, 48, November 2012. doi:10.1007/s11241-012-9158-9.

20 Gang Yao, Rodolfo Pellizzoni, Stanley Bak, Heechul Yun, and Marco Caccamo. Global real-time memory-centric scheduling for multicore systems. *IEEE Transactions on Computers*, 65(9):2739–2751, 2016. doi:10.1109/TC.2015.2500572.

21 Heechul Yun, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha. Memory bandwidth management for efficient performance isolation in multi-core platforms. *IEEE Transactions on Computers*, 65(2):562–576, 2015.

22 Alexander Zuepke, Andrea Bastoni, Weifan Chen, Marco Caccamo, and Renato Mancuso. Mempol: Policing core memory bandwidth from outside of the cores. In *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 235–248. IEEE, 2023.

A Multi-Modal Distributed Real-Time IoT System for Urban Traffic Control

Zeba Khanam
BT Security Research, Adastral Park, UK

Vejey Pradeep Suresh Achari
Keele University, Keele, UK

Issam Boukhennoufa
University of Essex, Colchester, UK

Anish Jindal
Durham University, Durham, UK

Amit Kumar Singh
University of Essex, Colchester, UK

Abstract

Traffic congestion is one of the growing urban problem with associated problems like fuel wastage, loss of lives, and slow productivity. The existing traffic system uses programming logic control (PLC) with round-robin scheduling algorithm. Recent works have proposed IoT-based frameworks that use traffic density of each lane to control traffic movement, but they suffer from low accuracy due to lack of emergency vehicle image datasets for training deep neural networks. In this paper, we propose a novel distributed IoT framework that is based on two observations. The first observation is major structural changes to road are rare. This observation is exploited by proposing a novel two stage vehicle detector that is able to achieve 77% vehicle detection accuracy on UA-DETRAC dataset. The second observation is emergency vehicle have distinct siren sound that is detected using a novel acoustic detection algorithm on an edge device. The proposed system is able to detect emergency vehicles with an average accuracy of 99.4%.

2012 ACM Subject Classification Computer systems organization → Real-time system architecture

Keywords and phrases Vehicle Detection, Deep Neural Network, Traffic Control, Edge Computing, Emergency Vehicle Detection, Sliding Window

Digital Object Identifier 10.4230/OASIcs.NG-RES.2024.2

Category Invited Paper

1 Introduction

One of the major problems plaguing urban cities is traffic congestion. This problem stems from lopsided growth in road infrastructure in comparison to traffic volume. The visible devastating effects on travelers and urban cities in general are increased global carbon footprint, low productivity, fuel wastage; resource depletion, loss of lives, and economic downfall. To address this pressing issue, governments have invested heavily in infrastructure upgradation with complex civil construction of bridges, roads, and new lanes addition. However, this solution has further complicated the issue. Big cities like London that have been implementing this solution are currently reeling under issues like urban sprawl, pollution, and stalling [8]. As the number of on-road vehicles is projected to increase manifold, this problem is going to intensify. With the global emphasis to cut the scope 1 emission due to road transportation for a sustainable future, an intelligent urban traffic system is the need of the hour.

Most of the cities deploy a traditional traffic control system that uses an array of programming logic controllers (PLC) and a round-robin scheduling algorithm to allow traffic for each lane to pass in a circular fashion [7]. However, few pilot studies have been conducted on the deployment of Internet of Things (IoT)-driven intelligent traffic control systems. Most of these studies have been part of the deployment of connected and automated vehicles and self-driving cars [16, 14]. Given the uncertainty around the future of self-driving cars and the urgency of decarbonization targets, there is a need for an IoT system for current on-road vehicles. Few recent IoT frameworks have used recent technologies like infrared cameras and GPS [3], RFID [6], Bluetooth and Zigbee [15]. However, these technologies require the deployment of an array of sensors in vehicles. These solutions require large investment and energy to revamp the existing vehicles and the accuracy of detection of traffic density estimation is not substantially high due to their susceptibility to noise from the environment [12].

To overcome the aforementioned problems, we proposed a novel framework, IoT based Intelligent Urban Traffic System (I^2UTS) for traffic light control that leveraged the existing CCTV network for designing the IoT-based framework for traffic light control [1]. CCTV videos have proven to be efficient and economical in past. There are several works that have explored the potential of CCTV camera networks as input sensors for solving traffic problems like predicting accidents, monitoring spatio-temporal behavior of pedestrians, and the detection of firearms and knives [4]. I^2UTS framework used CCTV video and state-of-the-art CNN to estimate the traffic density. Traffic density was a major input to control traffic lights. To address the privacy concern and computational resource requirements, Yolo v3 with a darknet backbone was deployed over edge device, Raspberry Pi, alongside our novel scheduling algorithm. Even though, I^2UTS was able to achieve 68.10% vehicle detection accuracy, it inherited problems associated with end-to-end convolutional neural networks (CNNs) that rendered practical difficulties in real-time traffic network analysis.

The first major problem associated with I^2UTS was the inability to find datasets with emergency vehicles. One of the novel contributions of our previous research was to account for the different traffic signal times incase emergency vehicles like police cars, firefighter trucks, and ambulances are part of the traffic scheduling algorithm. Since, the proposed CNNs Yolo v3- Efficient Net is dependent on labeled data, the availability of labeled dataset of the emergency vehicle was a challenge. Finding such a dataset is cumbersome given the rare occurrence of emergency vehicles in traffic conditions. The proposal of emergency vehicle datasets has been an area of research. Researchers have turned to various resources like google search alongside manual annotation of 1500 images, manual filtering of Kaggle dataset images, and youtube streams [13]. Given the amalgamation of various sources of image acquisition, these datasets suffer from large viewpoint variations, which renders them unsuitable for our IoT framework where the input sensor, CCTV camera, has a fixed viewpoint. Apart from viewpoint variation, another problem in the detection of emergency vehicles is weather variation. Inclement weather conditions present in the images often decrease vehicle detection accuracy significantly [5]. Since, the speed of non-emergency vehicles is comparatively slow, their occurrence on CCTV images for a fixed time frame is higher. This increases their detection accuracy alongside the availability of a large set of labeled images in varying weather as a training input for YOLO V3. These problems clearly illustrate the inhibition of using RGB cameras for emergency vehicles. This serves as a major motivation to re-investigate our previous work I^2UTS for emergency vehicle detection. Emergency vehicles all over the world use loud-noise making noise, sirens, to alert the traffic of passage. In this work, we propose a multi-modal distributed IoT framework that detects the distinguished sound property of emergency vehicles alongside the image-based traffic density estimation.

The second major problem associated with I^2UTS is the high variance in mean average precision (mAP) across different classes of vehicle detectors. Although, the major outlier was the class *"van"* where mAP was 37.49%. Further, False Discovery Rate (FDR) was 63.23%, which clearly indicates that the proposed YOLOV3-Effecient net has high false positive (FP) in comparison to true positive (TP) for *"van"*. YOLO is a preferred single-stage object detector in comparison to its counterparts two-stage detectors like RCNN due to its faster performance on edge devices. However, YOLO struggles to localize smaller objects/vehicles and identify the optimum number of clusters. This is the main reason that I^2UTS has a high misclassified bus stops as vans due to similar features. Single-stage detectors classify and localize objects in a single shot using dense sampling whereas two-stage detector consists of an additional preliminary stage of region proposal. The region proposal stage indeed increases the performance of object detector but are computationally expensive. However, the CCTV camera on road is fixed and viewpoint variation in the images captured is hardly possible. Changes in road infrastructure are also minimal. Considering, this advantage, we propose a novel two-stage detector road-based YOLO (*"R-YOLO"*) that confines the search of vehicles to the road with an increased performance accuracy comparable to two-stage object detector(like RCNN) and low computational resource usage like single stage detector (like YOLO).

The remainder of the paper is organised as follows. Section II describes the proposed IoT framework. The experimental evaluation and results of the proposed distributed system is detailed in Section III. Finally, the conclusion is presented in Section IV.

2 Proposed Distributed IoT framework

The proposed distributed IoT based urban traffic management system contains two edge parts: 1) S_1: Vehicle Detector that uses Deep Neural Network, R-CNN. 2) S_2: Emergency Vehicle Detector that uses acoustic sliding window approach to detect siren's of emergency vehicle.

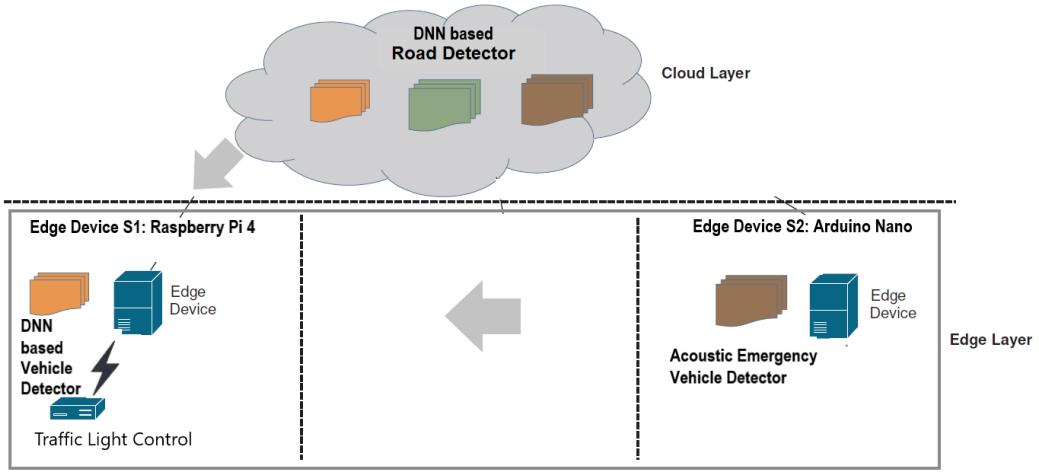


Figure 1 A multi-modal IoT Distributed Framework.

2.1 Edge Devices and Sensors

The previous works deployed NVIDIA Jetson TX2 that is a computationally powerful edge device with the dual support of CPU and GPU [2]. However, the cost of Jetson Nano is 24 times more than average cost of other edge devices. Given the economic feasibility contraint of the system, we use Raspberry PI 4 as the edge device S_1 for vision algorithms with moderate memory of 4 GB and powerful Quad core cortex-A72 (Arm-8) 1.5GHz processor.

The acoustics emergency vehicle detector uses Arduino Nano as Edge device S_2. Arduino Nano is an open source micro-controller based on ATmega328P architecture. With a flash memory of 32 KB, 16 Analog Pins and 22 I/O pins, SEN0232 noise meter is used. SEN0232 uses an instrument circuit and a low noise microphone, with a measuring decibel value ranges from 30dBA to 130dBA, accurately measuring noise level of the surrounding environment.

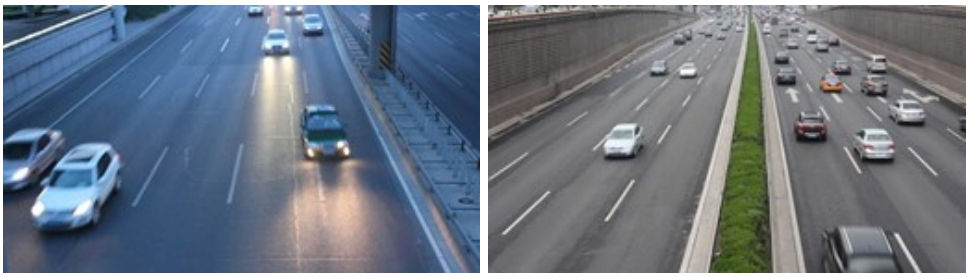
2.2 System Overview

A higher level overview of the distributed system is as follows.

1. **Cloud Layer: Road Detector**
   a. Train Faster-RCNN network on cloud to detect the vehicles.
   b. Detect the object road and Estimate the road mask.
   c. Train the YOLO v3 network with Efficient net as a backbone on cloud to detect the road.
   d. Transfer the trained weights to the Faster RCNN deployed on the edge device S_1 for testing.
   e. Transfer road mask to the edge device S_2 for road extraction.
2. **Edge S_2: Acoustic Emergency Vehicle Detector**
   a. Detect Siren Sound from continuous noise level monitoring.
   b. Send the emergency vehicle trigger to Edge Device S_1.
3. **Edge S_1: Vehicle Detector**
   a. Estimate the traffic density of each road lane by detecting vehicles using road mask and YOLOv3 vehcile detector.
   b. Use the parameters generated in previous steps as an input to the proposed traffic control algorithm.

2.2.1 Dataset

Many prior studies have utilized the KITTI and COCO datasets to train the YOLOv3 network. However, a more recent dataset, UA-DETRAC [18], developed by the University of Albany for object detection and tracking, is more profound as a benchmark dataset for real-world multi-object tracking. This dataset comprises traffic camera videos, recorded at 25 fps over a span of 10 hours, with a resolution of 960 x 540 pixels. The footage originates from 24 distinct locations in Beijing and Tianjin, China, offering a diverse and challenging environment. UA-DETRAC includes approximately 1.21 million labeled bounding boxes representing 8250 vehicles across four classes: car, van, bus, road and others. Figure 2 illustrates some CCTV images from the dataset. An additional strength of utilizing UA-DETRAC is its representation of multi-class weather conditions and variations in day and night illumination. This dataset is used for both the purposes-road and vehicle detection. For both the tasks, the dataset is divided into training, validation, and testing sets following a 70:15:15 ratio, respectively.



(a) (b)

Figure 2 Sample CCTV images from UA-DETRAC [18].

2.2.2 DNN for Road Detection

For road detection and classification, we use YOLO v3 CNN model [9] trained on the backbone of Efficient Net [17]. Unlike the traditional YOLO v3 model [9] which has used DarkNet-53 network as backbone, our network has a high object detection accuracy at low inference latency. Changes in road infrastructures are minimal, so we can assume CCTV images have advantage of viewpoint invariation. Since vehicles runs on road, the bounding boxes of vehicles are embedded within the bounding box of road, pooling of region of interests, we select the road bounding box. Once the bounding box is selected, we calculate *road mask*. Road Mask is a binary image that is applied to CCTV images to mask out remaining image (converts the pixels to black) except road object. Road Mask image is helpful in limiting regions of interest. Fig 3b, 3d, and 3f show the CCTV image obtain after applying road mask.

2.2.3 DNN for Vehicle detection

For vehicle detection and classification into five annotation classes: bus, car, vans, road, and others, we use Faster R-CNN model [10]. Faster R-CNN in comparison to traditional regional based CNN [11] is manifold times faster due to region proposal network. However, its pooling characteristic is also extremely beneficial. Faster RCNN has a better accuracy than single shot detectors like YOLO. However, the inference speed is extremely slow in comparison to YOLO, rendering it unsuitable for real-time object detection on edge devices. To overcome this, we limit the scope by inputing the image with road mask. Therefore, Faster R-CNN will now only detect object on road, significantly reducing the image area, number of objects, in turn reducing the processing time. Few instances of vehicle detection are illustrated in Figure 3.

2.2.4 Acoustic Emergency Vehicle Detection Framework

Most sirens are rated at around 124 dB when measured 10 feet in front of the sound source. As the distance from the siren doubles, the sound pressure of the siren will drop by approximately 6dB. This concept is known as the “inverse square law.” In our system, Data is collected with a frequency of 50Hz and a sliding window technique is employed to detect the emergency vehicle. A sliding window computes the area of the sound noise level over a certain period of time as shown in Figure 4. When the area exceeds a predefined value the detection algorithm dispatches the emergency light sequence. When it does not, normal sequence is carried out. The area value is computed using the formula:

$$Area = Frequency \times sound\ level \tag{1}$$

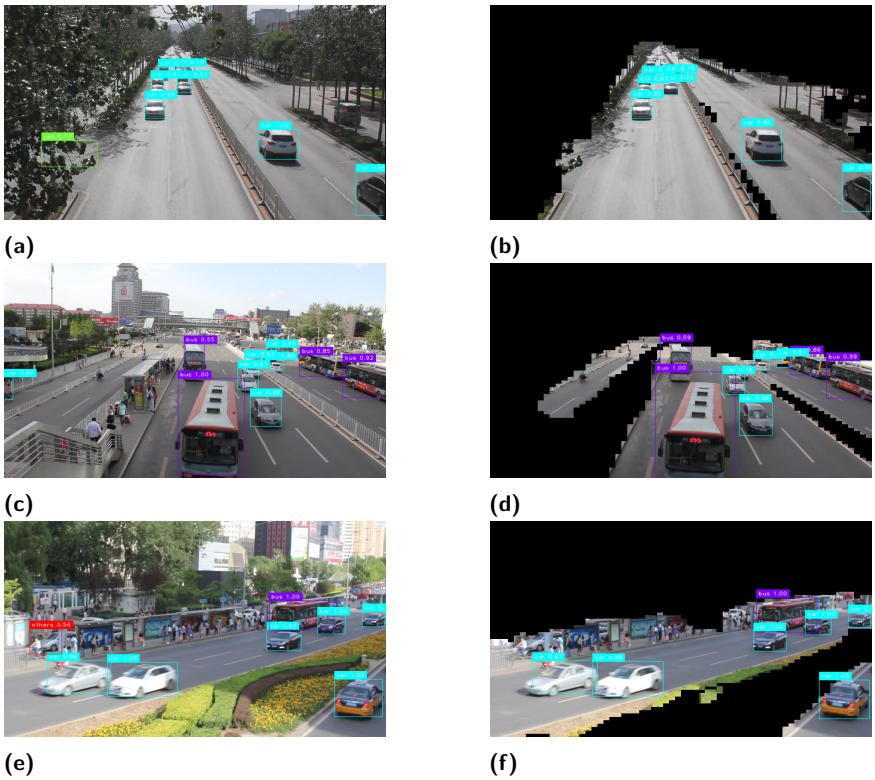


Figure 3 Examples of vehicle detection by a), c), e) by I2UTS and in b), d), f) by our proposed system on same CCTV images from UA-DETRAC [18]. Proposed system performs masking while leaving road.

The system is continuously computing the area over the window length, noise values are summed together over a defined window. The highest sound level is set 100dB, this value has been chosen to take into account different distances of the ambulance from the sound sensor. It has been chosen as it is the average noise level when taking into account 80 feet (25m) as the noise level varies between 124dB and 76dB.

Figure 4 Sample Sliding Window for Emergency vehicle's siren.

Once the siren is detect, a trigger is generated in Edge device S_2. This is sent to edge device S_1 as shown in Figure 1. Edge Device S_1 estimates the vehicle density of each lane using vision algorithm explained in Section 2.2.3. Traffic Control Algorithm proposed by us in our earlier work in I^2UTS uses both these parameters to calculate traffic light sequence again on edge device S_1. The mutli-modal nature of our framework that encapsulates both acoustic and vision sensor and processing to build a resilient system.

3 Experimentation and Results

In this section, we evaluate the performance of our proposed distrubuted IoT framework. The weights of the Faster RCNN and YOLO v3- Effecient Net are trained on a cloud server with Intel i7-9th generation as main processor alongside Nvidia 1660Ti GPU on Linux 18.04 operating system. CUDA 10.1 with Cudnn 7 libraries were used for parallel computation on GPU. The edge device S_1, Raspberry Pi 4, has Quad core cortex-A72 (Arm-8) 1.5GHz processor, 4GB RAM and OpenGL ES 3.0 graphics with Raspbian Buster as the operating system.

The experimental setup for edge system for edge comprises of a SEN0232 sound meter that is connected through a SPI serial connection to an Arduino Nano. This edge device is further connected to Raspberry Pi either wirelessly or through USB. Raspberry Pi is edge device that is responsible for managing the traffic light sequence.

3.1 Emergency Vehicle Detection

Experiments were conducted to find the optimal window lengths. To do so multiple window sizes were chosen starting from 0.5s to 10s. After each trial the window length is incremented by 0.5s. For each window size, ten different sounds are played, of which three correspond to sirens sound of emergency vehicles (police, ambulance, fire-truck) while the rest are different urban noises. The detection time is measured, as well as detection accuracy. The overall operation is repeated 100 times. The detection time D_t is the difference between the time at which the siren is first detected T_d and the time at which the sound is played T_p.

$$D_t = T_d - T_p \tag{2}$$

The accuracy of detection is defined as the number of the truly predicted siren sounds P_s and the truly predicted urban noises P_u divided by the number all the tests N.

$$Accuracy = \frac{P_s + P_u}{N} \tag{3}$$

Table 1 Accuracy and detection time per window size.

Window Length(s)	0.5	2	3.5	5	6.5	8	9.5
Detection Accuracy (%)	63.6	88.4	94.5	99.5	97.4	99.2	99.4
Detection Time (s)	0.54	2.032	3.524	5.031	6.521	8.041	9.523

Results are shown in Table 1, accuracy is very low for short window sizes. The main reason behind this is that it detects numerous urban noises as being siren sounds. Short window sizes make the detection algorithm act as a threshold detection. Accuracy increases with the window length to attain a maximum of 99.5% at 5s and it stays at the same level relatively. From 1000 segments, only 5 sound segments were miss-classified, and these

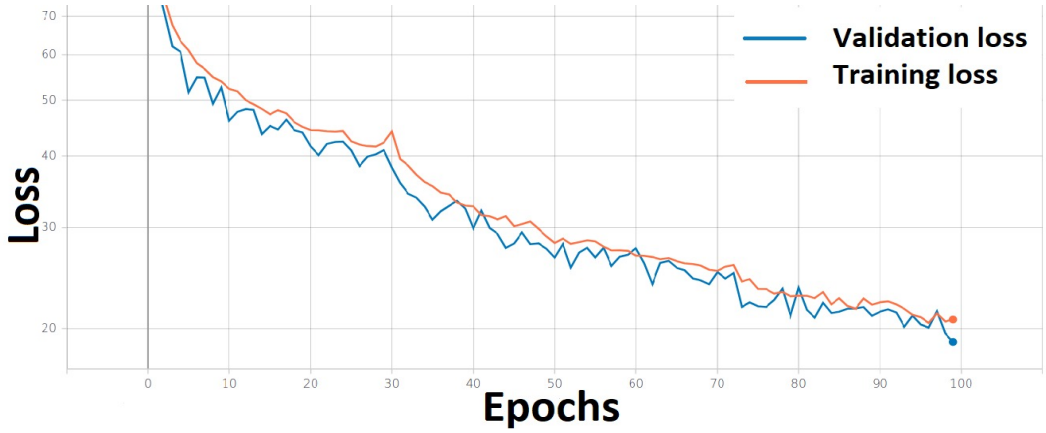


Figure 5 Number of Epoch v/s Loss Difference [1].

segments belong to the urban noises' sounds. The detection time for the different experiments takes an average time of 0.03s ± 0.007s. For 5s the maximum accuracy is reached, and it is selected to be utilised. Another reason is that it is more effective to have a quicker detection.

3.2 Vehicle Detection

While training and fine-tuning the hyper-parameters of both the DNNs for vehicle and road detection, we ensure model reaches optima by neither overfiting nor underfiting. The most important hyperparameter that we need to fine-tune will be number of epochs. To determine this, we plot number of epochs with respect to validation and training loss difference. Figure 5 shows that the loss value's difference is lowest around 100^{th} epoch.

The other important metric to measure efficiency of vehicle detection system is inference time. The inference time of our DNN on the edge divice S_1, Raspberry Pi varied between 1.55 – 2.3 sec per frame. This is comparable to state-of-the-art I^2UTS framework which had inference time of 1.45 – 1.57 sec per frame. The power consumption of edge device (Raspberry Pi 4) per second on different loads is presented in Table 2. The input voltage and current to edge device was DC 5.1V and 3A.

Table 2 Power consumption of IoT device on different loads.

Parameters	Current (Amps)	Voltage (Volts)	Power (Watts)
IoT device not connected to monitor	0.76	5.8	3.12
IoT device connected to monitor	0.78	5.8	3.45
IoT device running only detector	1.34	5.23	6.87
IoT device running detector with connected monitor	1.4	5.23	7.182

The highest power consumption observed was 7.18 W when the detector ran alongside a monitor, constituting only half of the input power supplied. When connected to a traffic camera, the power consumption reduced to 6.87 W. The mean average precision (mAP) for vehicle detection DNN is 79.5% in comparison to 65.10% achieved by state-of-the-art framework I^2UTS. If both the metrics inference time and accuracy are looked together, we can easily say that our proposed novel two-stage detector R-YOLO is able to achieve better accuracy in similar inference time.

4 Conclusion

This paper proposes a distributed IoT framework for urban traffic management system using the CCTV camera and sound sensor. The framework uses the two important observations in urban traffic control: 1) Structural Changes to road are minimum. 2) Emergency Vehicles have distinct sound. The novel two stage detector exploits the first observation by detecting road in first stage and vehicles in second stage. The detector achieves 79.5% accuracy that can further be enhanced by training the network on multiple datasets with CCTV footage with viewpoint and illumination variation. The second observation is implemented using acoustic sliding window detection algorithm achieving 99.4% accuracy.

References

1 Vejey Pradeep Suresh Achari, Zeba Khanam, Amit Kumar Singh, Anish Jindal, Alok Prakash, and Neeraj Kumar. I 2 UTS: An IoT based intelligent urban traffic system. In *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*, pages 1–6. IEEE, 2021.

2 Stephan Patrick Baller, Anshul Jindal, Mohak Chadha, and Michael Gerndt. DeepEdgeBench: Benchmarking deep neural networks on edge devices. In *2021 IEEE International Conference on Cloud Engineering (IC2E)*, pages 20–30. IEEE, 2021.

3 Sayalee Deshmukh and SB Vanjale. Iot based traffic signal control for reducing time delay of an emergency vehicle using gps. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pages 1–3. IEEE, 2018.

4 Michał Grega, Andrzej Matiolański, Piotr Guzik, and Mikołaj Leszczuk. Automated detection of firearms and knives in a CCTV image. *Sensors*, 16(1):47, 2016.

5 Jose Carlos Villarreal Guerra, Zeba Khanam, Shoaib Ehsan, Rustam Stolkin, and Klaus McDonald-Maier. Weather classification: A new multi-class dataset, data augmentation approach and comprehensive evaluations of convolutional neural networks. In *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 305–310. IEEE, 2018.

6 Yeong-Lin Lai, Yung-Hua Chou, and Li-Chih Chang. An intelligent IoT emergency vehicle warning system using RFID and Wi-Fi technologies for emergency medical services. *Technology and health care*, 26(1):43–55, 2018.

7 Vamsi Paruchuri, Sriram Chellappan, and Rathinasamy B Lenin. Arrival time based traffic signal optimization for intelligent transportation systems. In *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, pages 703–709. IEEE, 2013.

8 Sonam Pathak and Manish Pandey. Smart cities: Review of characteristics, composition, challenges and technologies. In *2021 6th International Conference on Inventive Computation Technologies (ICICT)*, pages 871–876. IEEE, 2021.

9 Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint*, 2018. `arXiv:1804.02767`.

10 Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

11 Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1137–1149, 2016.

12 David Dorantes Romero, Anton Satria Prabuwono, A Hasniaty, et al. A review of sensing techniques for real-time traffic surveillance. *Journal of applied sciences*, 11(1):192–198, 2011.

13 Shuvendu Roy and Md Sakif Rahman. Emergency vehicle detection on heavy traffic road from CCTV footage using deep convolutional neural network. In *2019 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, pages 1–6. IEEE, 2019.

14 Mohd Saifuzzaman, Nazmun Nessa Moon, and Fernaz Narin Nur. Iot based street lighting and traffic management system. In *2017 IEEE region 10 humanitarian technology conference (R10-HTC)*, pages 121–124. IEEE, 2017.

15 Rajeshwari Sundar, Santhoshs Hebbar, and Varaprasad Golla. Implementing intelligent traffic control system for congestion control, ambulance clearance, and stolen vehicle detection. *IEEE Sensors Journal*, 15(2):1109–1113, 2014.

16 Mehal Zaman Talukder, Sheikh Shadab Towqir, Arifur Rahman Remon, and Hasan U Zaman. An IoT based automated traffic control system with real-time update capability. In *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–6. IEEE, 2017.

17 Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

18 Longyin Wen, Dawei Du, Zhaowei Cai, Zhen Lei, Ming-Ching Chang, Honggang Qi, Jongwoo Lim, Ming-Hsuan Yang, and Siwei Lyu. UA-DETRAC: A new benchmark and protocol for multi-object detection and tracking. *Computer Vision and Image Understanding*, 193:102907, 2020.

DynaVLC – Towards Dynamic GTS Allocation in VLC Networks

Harrison Kurunathan
CISTER/ISEP, Polytechnic Institute of Porto, Portugal

Miguel Gutiérrez Gaitán[1]
Department of Electrical Engineering, Pontificia Universidad Católica de Chile, Santiago, Chile
Faculty of Engineering, Universidad Andres Bello, Santiago, Chile
CISTER/ISEP, Polytechnic Institute of Porto, Portugal

Ramiro Sámano-Robles
CISTER/ISEP, Polytechnic Institute of Porto, Portugal

Eduardo Tovar
CISTER/ISEP, Polytechnic Institute of Porto, Portugal

Abstract

Envisioned to deliver superior Quality of Service (QoS) by offering faster data rates and reduced latency in 6G communication scenarios, pioneering communication protocols like the IEEE 802.15.7 are poised to facilitate emerging application trends (e.g. metaverse). The IEEE 802.15.7 standard that supports visible light communication (VLC) provides determinism for time-critical reliable communication through its guaranteed time-slots mechanism of the contention-free period (CFP) while supporting non-time-critical communication through contention-access period (CAP). Nevertheless, the IEEE 802.15.7 MAC structure is fixed and statically defined at the beginning of the network creation. This rigid definition of the network can be detrimental when the traffic characteristics evolve dynamically, for example, due to environmental or user-driven workload conditions. To this purpose, this paper proposes a resource-aware dynamic architecture for IEEE 802.15.7 networks that efficiently adapts the superframe structure to traffic dynamics. Notably, this technique was shown to reduce the overall delay and throughput by up to 45% and 30%, respectively, when compared to the traditional IEEE 802.15.7 protocol performance under the same network conditions.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Networks → Network protocols; Theory of computation → Design and analysis of algorithms

Keywords and phrases IEEE 802.15.7, VLC networks, network tuning

Digital Object Identifier 10.4230/OASIcs.NG-RES.2024.3

Category Invited Paper

Funding This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology) within the CISTER Research Unit (UIDB/04234/2020).

1 Introduction

With 6G expected on the horizon by 2025, new technologies must be adopted to ensure the flawless usage of 6G [5]. Likewise, with the ever-growing network traffic demand and the need to support high bandwidth applications, researchers are venturing into new communication possibilities, including Visible Light Communication (VLC) and the Terahertz (THz) band. VLC, particularly, is deemed to be well-suited to meet the criteria of emerging applications toward 6G, including Virtual Reality (VR) and augmented Reality (AR), among others.

[1] Corresponding author

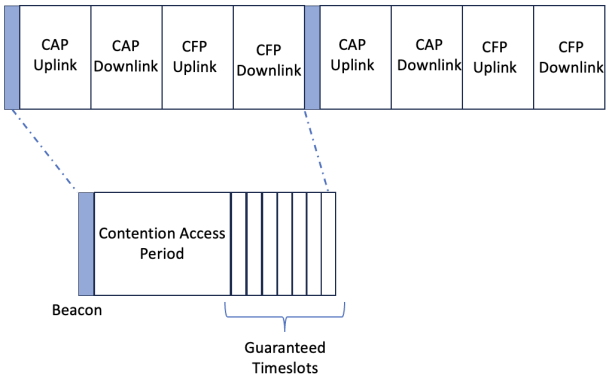


Figure 1 The superframe structure of IEEE 802.15.7 with the contention-free period of guaranteed time-slots enabling time-critical communication for VLC applications.

VLC's distinctive features such as its immunity to electromagnetic interference, high data rates, and the capability to operate in unlicensed bands [7], have placed it among the potential candidates to be included in the competitive arsenal of 6G communication technologies.

IEEE 802.15.7 [13] is a communication protocol poised to realize communication in VLC networks. The architecture of this standard supports high data rates of up to 96 Mbps with almost 300 THz of unlicensed spectrum. This makes it ideal to support bandwidth-hungry applications, potentially using existing illumination infrastructure. The standard also offers predictable protocol features enabling the support of critical and demand strict timeliness requirements through its Guaranteed Time Slot (GTS) mechanism, which operates in a periodically synchronized superframe structure (Fig. 1).

Among the key parameters of the protocol is the *superframe order* (SO) which defines the duration of the active period of the superframe, a.k.a. the *superframe duration* (SD). Within this scheme, beacons are transmitted between subsequent superframes enabling time synchronization and MAC management. The time interval between beacons is known as the *beacon interval* (BI). All these parameters can be properly set statically at the beginning of the network to govern overall communication performance. This approach, although suitable in the case of highly stationary network scenarios, prevents achieving adequate Quality of Service (QoS) when traffic characteristics evolve dynamically, for example, due to environmental or user-driven workload conditions.

In fact, in several potential VLC scenarios for 6G, such as healthcare monitoring [9], underwater networks [1] or vehicular communication, to name a few, the data traffic and/or the number of nodes that connect (or disconnect) to a central coordinator can vary frequently, e.g., based on local environmental circumstances [2], mobility of the nodes from one area to another [9], and/or due to multiple nodes reaching the same area and creating a bottleneck, which implies more traffic to be accommodated. The aforementioned static settings are just examples of how a dynamic traffic behavior can lead to inevitable compromises on QoS on metrics such as (worst-case) delay or throughput. This raises a need for a novel VLC network architecture that can adapt protocol features on the fly to varying conditions.

In this paper, we propose an adaptive MAC architecture called the DynaVLC that will dynamically toggle the network parameters and make them suitable to the underlying traffic behavior. This method can adapt efficiently to scenarios where the data traffic demand grow either higher or lower while satisfying QoS requirements such as latency or throughput. This tuning technique can be facilitated by managing entities such as the network coordinators

typically set to be aware of the network demand requirements to be served by the GTS. More concretely, making the network coordinators demand-aware can be done, for example, by integrating an RPL (Routing Protocol for Lossy Networks) layer over the VLC MAC layer.

We summarize the main contributions presented in this paper as follows:

- We provide a novel dynamic MAC structure tuning architecture called DynaVLC for IEEE 802.15.7 networks that yields better QoS performance.
- We introduce the so-called CAP reduction and modeling of the GTS under the DynaVLC architecture for several scenarios involving varying network demand.
- We derive the worst-case bounds and perform an in-depth performance analysis of the proposed structure covering both throughput and delay analysis.

The rest of the paper is organized as follows: Section 2 provides related works on some of the adaptive techniques devised for VLC networks and general communication protocols. Section 3 presents the CAP reduction technique as one of the key elements of the DynaVLC architecture to increase the number of GTS in the superframe. Section 4 introduces the system model and discusses the topologies and scenarios taken to demonstrate this architecture. Section 5 presents our novel DynaVLC architecture, and Section 6 analyzes its performance. Conclusions and a wrap-up with some discussion of future scope are presented in Section 7.

2 Related Works

The research work in [3] proposes a flexible superframe structure that enables sleep modes for priority data handling in IEEE 802.15.7-based real-time sensor networks. This method enables a hybrid mode in the contention access period and contention-free period (CFP) adaptively. The method works by shifting periods and sending priority data with lower bandwidth and delay. While the method holds promise in static/stationary conditions, the improvements do not show to be suitable for evolving traffic conditions.

A different approach is proposed by researchers in [17] who propose a priority MAC based on a multi-parameter for IEEE 802.15.7 VLC networks. They make use of common parameters such as the backoff times (NB), backoff exponent (BE), and contention window (CW) to enable priority-driven multilevel differentiated service. Moreover, using a discrete-time Markov chain model, the authors analyzed the impact of their multi-parameter traffic differentiation on throughput. More recently, a comparison between the traditional IEEE 802.15.7 frame and a novel energy-efficient superframe was done in [4]. This work also considered different inputs such as the biosensors' battery life as well as adaptive data requirements to vary MAC parameters accordingly. However, in both of these works variations in traffic data were not considered. The data traffic was set as a constant and only the impact of the variation of the MAC parameters such as the BO and SO were considered.

In one of our previous works, we presented the worst-case bounds delay of IEEE 802.15.7 using network calculus [13]. In this work, we explored the possibility of a technique called CAP reduction functionality from IEEE 802.15.4 and carried out a detailed performance analysis. This technique increases the number of GTS slots in the traditional IEEE 802.15.7 MAC frame, thus increasing the scalability for critical communication nodes in the network. Based on these results, we recently presented in [14] the possibility of having a multichannel structure to enhance the allocation for GTS in IEEE 802.15.7 frames. While both of these works focused on increasing the number of GTS timeslots, both the methods are statically defined, and thus cannot adapt to traffic dynamics in evolving network scenarios.

Adaptive superframe is a concept that has been researched for several network protocols like the IEEE 802.15.4 and IEEE 802.15.6. The underlying idea is that superframes are flexible to support GTS requirements [8, 11] where the active period or the CFP is adapted

as per the requested data. They also can be adapted to support priority data [15, 6] and support specific QoS like the energy efficiency [16]. Along this line of thought, in this work, we propose a novel technique called DynaVLC for IEEE 802.15.7 VLC networks where the superframe structure can be adjusted based on the oncoming traffic needs. The end goal is to significantly improve network throughput and reduce the overall worst-case delay toward deterministic 6G application scenarios.

3 Background to the CAP reduction architecture

CAP reduction is a technique where two or more superframes can be joined together as a multi-superframe and the CAP period between them can be removed and replaced with a CFP period. This technique was first introduced for the IEEE 802.15.4e - DSME network protocol [10] and then extended to the IEEE 802.15.7 protocol in [13].

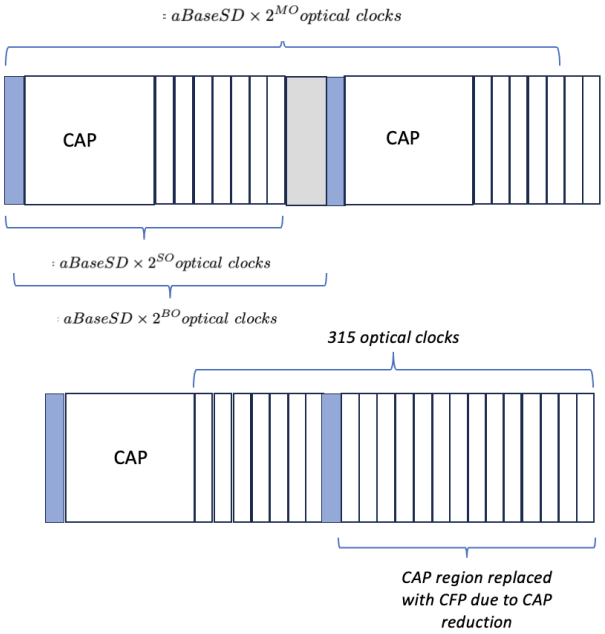


Figure 2 The superframe structure representation where BO=3, MO=3 and SO=2, also showing the structure with CAP reduction comprising of 21 GTS timeslots to support critical deterministic communication.

To have CAP reduction in the classes IEEE 802.15.7 protocols, we must introduce first a concept called multi-superframes that can be enabled through multi-superframe order (MO) and multi-superframe duration (MD). These parameters define the length of all the individual superframes within the multi-superframe. The aforementioned parameters can be formally represented as follows:

$$BI = aBaseSD \times 2^{BO} optical\ clocks \qquad for\ 0 \leq BO \leq 14 \tag{1}$$

$$SD = aBaseSD \times 2^{SO} optical\ clocks \qquad for\ 0 \leq SO \leq BO \leq 14 \tag{2}$$

$$MD = aBaseSD \times 2^{MO} optical\ clocks \qquad for\ 0 \leq SO \leq MO \leq BO \leq 14. \tag{3}$$

Table 1 Network configurations and their respective application scenarios.

Application	BO	SO	MO	CAP reduction	SD size
Delay sensitive	6	0	2	enabled	60
Large scale	10	1	8	enabled	128
Energy critical	6	1	1	disabled	128
Reliability	8	2	2	disabled	240

The number of multi-superframes within a beacon interval is given by $2^{(BO-MO)}$, and the number of superframes within the multi-superframe by $2^{(MO-SO)}$. To illustrate this scheme we can take the configuration presented in Figure 2, which is a network infrastructure representation where BO=3, MO=3 and SO=2. This is a case where two superframes are stacked within a single multi-superframe. Note that after the network is initiated with these parameters the infrastructure remains unchanged and the setup repeats periodically. For clarity, we briefly describe the most relevant parameters as follows:

$aBaseSD$ is defined as the minimum duration of a superframe and is set to 60 optical clocks at the initial order of the superframe (i.e., SO=0). Formally, this value is defined as:

$$aBaseSD = Slot\ Duration \times T_s \tag{4}$$

where T_s is the size of the timeslot in the superframe. Note that T_s in a superframe is made up of the data frames and idle frames. Data frames encompass the data transmissions and the idle frames encompass acknowledgments, long interframe spacing ($LIFS$), short interframe spacing ($SIFS$), and reduced interframe spacing ($RIFS$). Then, to develop the worst-case bounds analysis, we must include the GTS transmission, its respective acknowledgments and the CAP region within the multi-superframe. As every VLC superframe comprises 16 timeslots, the size of a single timeslot is denoted as:

$$T_s = \frac{SD}{16} = aBaseSD \times 2^{SO-4}. \tag{5}$$

For the parameters that define the multi-superframe architecture of Figure 2, the size of every single timeslot will be 15 optical cycles, the superframe duration will be 240 optical cycles and the entire multi-superframe will be 480 optical cycles. Among these optical cycles 210 optical cycles that correspond to 14 GTS timeslots across the multi-superframe support critical deterministic communication. Now for the same structure when we employ CAP reduction, the CAP region of the second superframe is replaced with a CFP and the inactive period is removed, thus drastically increasing the number of GTS timeslots in the network. In such a case, there will be 315 optical cycles corresponding to 21 GTS timeslots to support critical deterministic communication. Now when multichannel communication can exist over this architecture, i.e., over three multi-channels, there will be a total of 63 GTS timeslots over 315 optical cycles.

The setting of the network parameters and the CAP reduction can also be made application-specific. For instance, in a delay-sensitive network that carries priority traffic, we need an architecture with minimal SD size so that the next packet can be sent with minimal latency. With CAP enabled, the delay due to waiting for the inactive period and the adjacent CAP region can be avoided. In the case of a large-scale network, more nodes must be accommodated within a short period. In such cases, there is a need for a short SD duration but a larger number of superframes within a single multi-superframe. As an illustrative example, different network configurations and their respective application scenarios are shown in Table 1.

4 System Model

With the possibility of having multiple channels, enabling multi-channel mesh networks would be feasible in VLC scenarios. Having this in mind, we assume a mesh network as shown in Figure 3. To emulate real networking scenarios, we consider dynamic nodes that join and leave the network. A mesh network consists of a PAN coordinator (node PAN-C in Figure 3), which can transceive messages and beacons. Then there will be Fully Functional Nodes (FFN) that facilitate routing and send beacons for association and timing synchronization. Finally, the Reduced Functional Nodes (RFN) are capable of only receiving messages. Such a network is facilitated with the aid of routing using protocols like the RPL by which a point-to-many-points (P2MP) tree-like network can be devised.

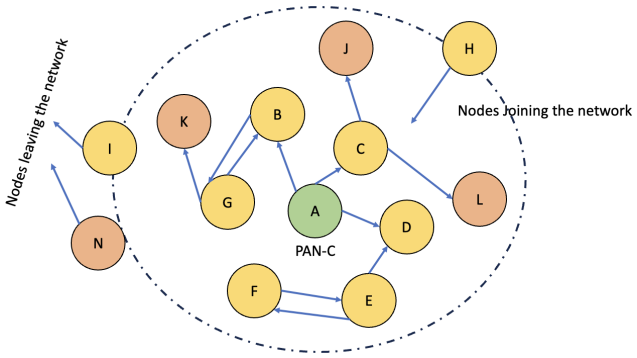
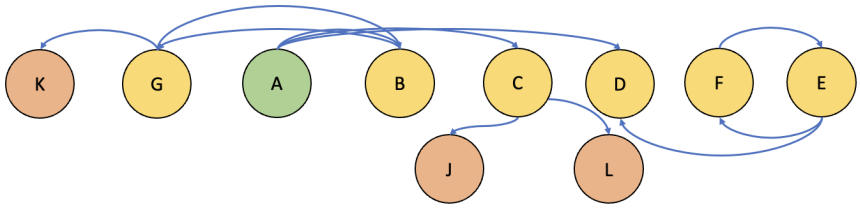


Figure 3 A mesh network comprising of the PAN-C, FFNs (green nodes) and RFNs (orange nodes) that dynamically join and leave the network.

Node association is done through the PAN coordinator. At the inception of the network formation, new FFNs advertise their respective superframe through periodic beacons. Association to an FFN, RFN or a PAN-C is done through an association request.

The nodes in the association process are assumed to be RPL-enabled routing nodes. The PAN-C acts as the sink in the Destination-Oriented Directed Acyclic Graph (DODAG). PAN-C is responsible for transmitting DODAG messages. In the RPL overlay network, all routers (FFNs) continuously broadcast DAG Information Object (DIO) messages to announce the DODAG. A node listens to the DIO messages when it joins the network through the association process. Upon receiving a DIO message from the FFN, the joining node adds the sender's DIO address to its parent list and calculates its rank based on the specified Objective Function (OF). The Objective Function for the DODAG can be QoS-defining factors such as Link Quality Indicator, Packet Delivery Ratio, or Power Consumption. Finally, the DIO message is updated with the newly computed ranks. The client node then selects its preferred parent from the list of FFNs as the default node through which inbound traffic is directed.

Figure 4 presents the mesh connection network for the network defined in the system model (Figure 3). An optimal schedule that utilizes the minimal number of time slots and channels can be defined using optimization methods like the Symphony [12] (adapted to IEEE 802.15.7 structure). Still, it must follow the mandate that the transmitting nodes do not overlap in time amongst themselves. By using Symphony, we provide a (near) optimal solution that uses 12 GTSs spanning over four channels and three timeslots. A transmission bitmap (Figure 4) will be created based on the transmissions of the mesh network and will be passed on to the underlying link layers using the RPL backbone periodically at every beacon interval.



Mesh connection

A -> B	A->C	A->D
G->K	G->B	B->G
C->J	C->L	F->E
E->F	E->D	

Optimized schedule for GTS allocation

	A	B	C	D	E	F	G	K	J	L
A	0	1	1	1	0	0	0	0	0	0
B	0	0	0	0	0	0	1	0	0	0
C	0	0	0	0	0	0	0	0	1	1
D	0	0	0	0	0	0	0	0	0	0
E	0	0	0	1	0	1	0	0	0	0
F	0	0	0	0	1	0	0	0	0	0
G	0	1	0	0	0	0	0	1	0	0

Tranmission Bitmap

Figure 4 mesh formation of the network, its optimized schedule and its respective transmission bitmap that will be transmitted through the RPL-enabled routing nodes.

The proposed solution in this work aids in tackling two major network issues caused by the static assignment of the network parameters at the inception of the network formation. The first problem is a requisite when there is *a need for a more guaranteed bandwidth than what is available*. More bandwidth will be provided if a smaller SO is defined at the beginning of the network definition. By setting a smaller SO more superframes can be affixed within the multi-superframe duration, further with CAP reduction the total number of guaranteed bandwidth to be serviced can also be drastically increased. However, in the case of a small SO with a large amount of bandwidth available, it could be a negative factor when *there is a need for less bandwidth compared to what is available*. The more suitable solution for these aforementioned problems is a tunable network architecture that can adjust its network parameters when the network demand changes.

5 DynaVLC architecture

The PAN-C establishes the multi-channel GTS allocation based on the number of channels, the number of GTS time slots and the total available GTS resources N_{CFP}. When the CAP reduction primitive is enabled the total number of GTS timeslots N_{TS} augments to $7 + N_{CR}$, where N_{CR} is the number of timeslots added through CAP reduction. In a system with C channels, the total resources available can be computed as $C \times N_{TS}$.

The duration of timeslot in the multi-superframe T_{MS} with N_η symbols that encompasses the size of the CAP (T_{CAP}) and the CFP created through CAP reduction T_{CFP} can be calculated as:

$$T_{MS} = \frac{N_\eta}{T_{CAP} + T_{CFP}}. \tag{6}$$

Let GTS_{min} be the minimum number of superframe slots a single GTS can extend over. We present this constraint such that there is a limit for the GTS not to span over multi-superframe duration for a maximum forward delay of D_{max}.

$$GTS_{min} = \left\lceil \frac{D_{max}}{T_{MS}} \right\rceil \tag{7}$$

For n timeslots with a burst rate b and a data rate D, the maximum forward delay D_{max} can be obtained as:

$$D_{max} = \frac{b \times BI}{D \times T_{data}} + (BI - n(T_{MS})). \tag{8}$$

Then, since the maximum number of the GTS varies based on the CAP reduction technique, with C number of channels spanning across the CFP timeslots, the max GTS can be defined as:

$$GTS_{max} = min\left(\left\lceil \frac{(T_{CAP}+T_{CFP})\left(1-\frac{T_{CAP}}{T_{MS}}\right)}{GTS_{min}} \right\rceil, C \times N_{CFP}\right). \tag{9}$$

Following the availability of the transmission bitmap from the optimal schedule through the RPL backbone, the amount of the required resources R is known to the PAN-C. Based on the requirement of resources, if needed more, the PAN-C adds/removes CAP reduction primitive and increments/decrements the value of MO in the subsequent beacon intervals as shown in Algorithm 1.

Algorithm 1 DynaVLC algorithm to dynamically tune the superframe to the network demand.

Input: BO, SO, MO
optimal schedule from the RPL backbone
Number of channels (C) and Number of GTS available (N_{CFP})
Initialization
repeat
  Schedule **R** = Required number of resources to accommodate the network
  Resource test: check $N_{CFP} \geq R$ in a multi-superframe

  Problem 1: Minimal resources and high demand
  while $N_{CFP} \leq R$ **do**
    CAP Reduction = ON;
    if resource test true **then**
      Print: DynaVLC is successful,
    else
      MO = MO + 1;
    end if
  end while

  Problem 2: abundant resources and less demand
  while $N_{CFP} \geq R$ **do**
    CAP Reduction = OFF;
    if Resource test true **then**
      Print: DynaVLC is successful,
    else
      MO = MO - 1;
    end if
  end while
until Every multi-superframe duration

In a multi-superframe duration (MD) of N superframes, and for a data rate of D over C channels, the maximum throughput for a single multi-superframe duration can be given as:

$$TH_{max} = \left(\frac{(N \times T_{MS}) - T_{idle}}{GTS_{max}} \right) \times D * C. \tag{10}$$

6 Numerical Analysis

To analyze the impact of DynaVLC, we consider an evolving network with the number of GTS transmissions increasing over time and analyze the delay. Let us consider a multi-superframe architecture with BO =6, MO = 1, SO = 1, such that there will be two superframes within a multi-superframe that repeats for every beacon interval. For this test let us consider three channels spanning over the 7 GTS in the classic IEEE 802.15.7 structure. In the classic VLC structure with 3 channels, there will be a total of 21 GTS slots, which will not be capable of accommodating more than 21 pairs of transmissions. However, when CAP reduction is added to the multi-superframe the number of available GTS increases to 63 individual GTS slots. However, in the case of static CAP reduction, after the 63 timeslots are filled, it waits for the entire CAP duration until the subsequent superframe starts allocating the GTSs.

In the case of DynaVLC, when the number of resources is minimal, initially the CAP reduction kicks in and we get almost the same performance as that of the static CAP reduction. When the entire CFP is full, the resource test R fails and the value of MO is increased adding another superframe to the multi-superframe. Now with CAP reduction on all of the superframes, we will have a total of 102 GTS slots for deterministic communication resulting in a decrease in delay by up to 45 %.

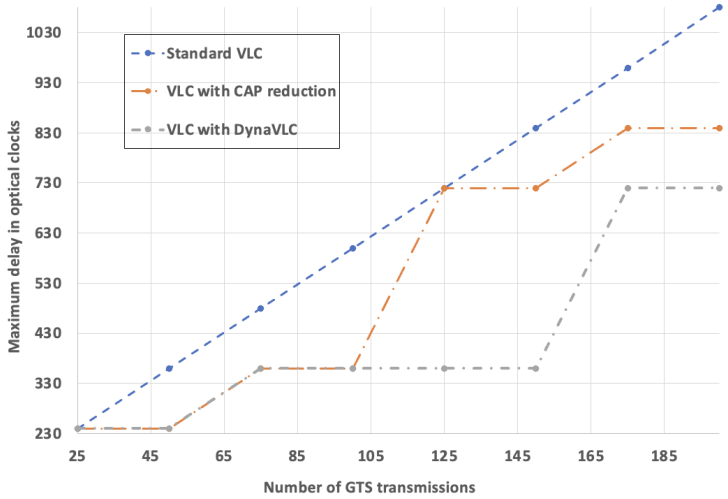


Figure 5 Impact of DynaVLC on the overall delay of the network - with the increment of MO as the number of GTS transmissions increase more superframes are added into the multi-superframe duration to accommodate the GTSs.

In the second part of our numerical analysis, we study the throughput of the network, comparing the static settings against the DynaVLC. Under static CAP reduction, we switch it "ON" at the beginning of the network, hence it has enough amount of GTS to accommodate the traffic. Yer, as the number of GTS increases, the non-allocated slots will have to wait for an entire CAP period to get served in the subsequent superframe. This results in a decrease in the network throughput, but still, it is higher than the standard VLC by 20–30 %.

In the case of DynaVLC, we initially have the CAP reduction setting "ON" to support the network demand, hence, it provides an identical throughput as the example with CAP reduction. However, as the number of GTS increases, the value of MO is incremented resulting in the addition of more superframes into the multisuperframe. Around 125 GTS requirement with the addition of more superframes into the MD, we witness an increase of throughput and it slowly reduces with the increase of the GTS slots. The throughput will eventually converge when the values of BO and MO become equal and all the GTS slots are occupied.

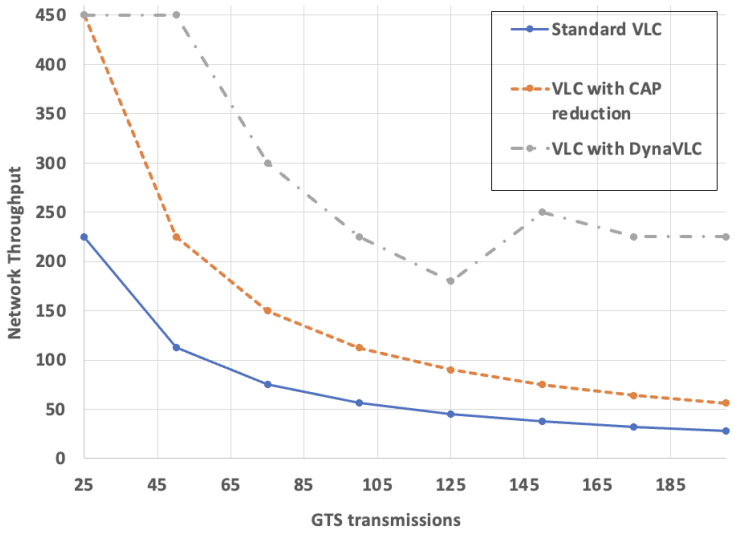


Figure 6 Impact of DynaVLC on the overall throughput of the network - with the increment of MO as the number of GTS transmissions increase more superframes are added into the multi-superframe duration to accommodate the GTSs.

7 Conclusion

In current VLC network deployments QoS defining MAC parameters such as MO, SO, BI are statically defined. This is an impediment to constantly evolving networks with varying workload conditions. To address the compromises of these static networks, in this research work, we propose a dynamic tuning mechanism called DynaVLC that can adjust the value of MO and CAP reduction to increase the resources available based on the changes in network demand. With DynaVLC, we were able to witness a decrease of 15-45% in delay when compared to the network in static settings, as well as an improvement of 20-30% in terms of the overall throughput. As a future work, we intend to create an open-source implementation of the IEEE 802.15.7 protocol adaptations here introduced with further enhancements towards the existing VLC architecture.

References

1 Mohammad Furqan Ali, Dushantha Nalin K Jayakody, and Yonghui Li. Recent trends in underwater visible light communication (UVLC) systems. *IEEE Access*, 10:22169–22225, 2022.

2 Lucas Almonacid, Pablo Palacios Játiva, Cesar A Azurdia-Meza, Diego Dujovne, Ismael Soto, Ali Dehghan Firoozabadi, and Miguel Gutierrez Gaitan. On the path loss performance of underwater visible light communication schemes evaluated in several water environments. In *2023 South American Conference On Visible Light Communications (SACVLC)*, pages 12–16. IEEE, 2023.

3 Monica Bhutani, Brejesh Lall, and Monika Agrawal. An efficient and adaptive superframe structure for IEEE 802.15.7-based real-time sensor networks. Research Square (Preprint), 2023.

4 Monica Bhutani, Brejesh Lall, and Monika Agrawal. A novel energy-efficient adaptive superframe structure for OWC-based real-time bio-sensor networks. Research Square (Preprint), 2023.

5 Wanshi Chen, Xingqin Lin, Juho Lee, Antti Toskala, Shu Sun, Carla Fabiana Chiasserini, and Lingjia Liu. 5G-advanced toward 6G: Past, present, and future. *IEEE Journal on Selected Areas in Communications*, 41(6):1592–1619, 2023.

6 Yuanming Ding, Yang Liu, and Jianxin Feng. Multi-priority MAC protocol for terahertz wireless local area network. In *2022 34th Chinese Control and Decision Conference (CCDC)*, pages 3427–3432. IEEE, 2022.

7 Prashant Dwivedy, Vipul Dixit, and Atul Kumar. A survey on visible light communication for 6G: Architecture, application and challenges. In *2023 International Conference on Computer, Electronics & Electrical Engineering & their Applications (IC2E3)*, pages 1–6. IEEE, 2023.

8 Sangrez Khan, A Naseem Alvi, M Awais Javed, and Safdar H Bouk. An enhanced superframe structure of IEEE 802.15. 4 standard for adaptive data requirement. *Computer Communications*, 169:59–70, 2021.

9 Harrison Kurunathan, R Indhumathi, Miguel Gutiérrez Gaitán, Carla Taramasco, and Eduardo Tovar. VLC-enabled monitoring in a healthcare setting: Overview and challenges. In *2023 South American Conference On Visible Light Communications (SACVLC)*, pages 135–140. IEEE, 2023.

10 Harrison Kurunathan, Ricardo Severino, Anis Koubaa, and Eduardo Tovar. IEEE 802.15. 4e in a nutshell: Survey and performance evaluation. *IEEE Communications Surveys & Tutorials*, 20(3):1989–2010, 2018.

11 Harrison Kurunathan, Ricardo Severino, Anis Koubaa, and Eduardo Tovar. DynaMO – Dynamic multisuperframe tuning for adaptive IEEE 802.15. 4e DSME Networks. *IEEE Access*, 7:122522–122535, 2019.

12 Harrison Kurunathan, Ricardo Severino, Anis Koubâa, and Eduardo Tovar. Symphony: routing aware scheduling for DSME networks. *ACM Sigbed Review*, 16(4):26–31, 2020.

13 Harrison Kurunathan, Ricardo Severino, and Eduardo Tovar. A comprehensive worst case bounds analysis of IEEE 802.15. 7. *Journal of Sensor and Actuator Networks*, 10(2):23, 2021.

14 Harrison Kurunathan, Ramiro Sámano-Robles, Miguel Gutiérrez Gaitán, R. Indhumathi, and Eduardo Tovar. Towards multi-channel GTS allocation in visible light communication. In *2023 South American Conference On Visible Light Communications (SACVLC)*, pages 130–134, 2023. doi:10.1109/SACVLC59022.2023.10347865.

15 Bih-Hwang Lee, Huai-Kuei Wu, and Neng-Chun Yu. A priority based algorithm for adaptive superframe adjustment and GTS allocation (PASAGA) in IEEE 802.15. 4 LR-WAN. In *2018 IEEE International Conference on Applied System Invention (ICASI)*, pages 318–320. IEEE, 2018.

16 Uma Shankar Pandey, Gulshan Soni, and Saroj Kumar Chandra. The impact of alteration of superframe duration on the consumption of energy in the IEEE 802.15. 4 MAC. In *2023 5th International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pages 254–260. IEEE, 2023.

17 Vu Van Huynh, Yeong Min Jang, et al. Priority MAC based on multi-parameter for IEEE 802.15. 7 VLC. In *ICTC 2011*, pages 257–260. IEEE, 2011.

History-Based Run-Time Requirement Enforcement of Non-Functional Properties on MPSoCs

Khalil Esper
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany

Jürgen Teich
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany

Abstract

Embedded system applications usually have requirements regarding non-functional properties of their execution like latency or power consumption. Enforcement of such requirements can be implemented by a reactive control loop, where an enforcer determines based on a system response (feedback) how to control the system, e.g., by selecting the number of active cores allocated to a program or by scaling their voltage/frequency mode. It is of a particular interest to design enforcement strategies for which it is possible to provide formal guarantees with respect to requirement violations, especially under a largely varying environmental input (workload) per execution. In this paper, we consider enforcement strategies that are modeled by a finite state machine (FSM) and the environment by a discrete-time Markov chain. Such a formalization enables the formal verification of temporal properties (verification goals) regarding the satisfaction of requirements of a given enforcement strategy.

In this paper, we propose *history-based* enforcement FSMs which compute a reaction not just on the current, but on a fixed history of K previously observed system responses. We then analyze the quality of such enforcement FSMs in terms of the probability of satisfying a given set of verification goals and compare them to enforcement FSMs that react solely on the current system response. As experimental results, we present three use cases while considering requirements on latency and power consumption. The results show that history-based enforcement FSMs outperform enforcement FSMs that only consider the current system response regarding the probability of satisfying a given set of verification goals.

2012 ACM Subject Classification Computer systems organization → Multicore architectures; Theory of computation → Linear logic; Theory of computation → Modal and temporal logics; Hardware → Finite state machines; Computer systems organization → Self-organizing autonomic computing; Theory of computation → Verification by model checking; Mathematics of computing → Probabilistic representations

Keywords and phrases Verification, Runtime Requirement Enforcement, History, Latency

Digital Object Identifier 10.4230/OASIcs.NG-RES.2024.4

Funding This work is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research-Foundation) – Project Number 146371743 – TRR 89 Invasive Computing.

1 Introduction

Embedded applications usually come with constraints on non-functional properties such as latency, power consumption, temperature, security, etc. A major uncertainty source that affects such properties is the varying workload of the input data[1]. Different run-time

[1] Other uncertainties such as caused by resource sharing can be handled systematically by techniques for isolating application programs dynamically at run-time such as invasive computing [1, 32] and therefore not considered here.

management methods exist for dynamic control of program executions. However, most of them have as disadvantages that they cannot provide formal guarantees regarding their capability to fulfill the given requirements.

Run-time requirement enforcement (RRE) techniques [33] have been proposed to enforce a set of non-functional properties of execution of a given application program within defined bounds. Such techniques dynamically adapt system configurations including, e.g., the voltage/frequency settings and/or the number of active cores in reaction to observed system responses. Based on that, *FSM-based RREs* [10–13, 30] have been proposed for formally specifying and verifying control strategies. Such approaches consider execution properties that can be modeled by *requirements* [31], i.e. expressions on non-functional properties such as permitted corridors on latency, power consumption, etc. Different verification goals can be specified and formally verified, e.g., the probability with which program executions satisfy a given set of requirements.

The FSM-based RRE approaches in [10–13, 30] define and use a binary *requirement response* vector that specifies for each given requirement whether it has been satisfied (1) or not (0) in the current execution. Based on such a system response, then determines the next state, respectively configuration to be applied during the next execution. However, it is a challenge to design enforcement FSMs that satisfy a set of verification goals with maximized probabilities, especially if the considered requirements are conflicting with each other like latency and power consumption. A potential for improvements is to let the enforcer consider not only the current, but also system responses from earlier execution iterations when deciding for the next configuration. In this regard, this paper proposes *history-based enforcement FSMs* that not only consider the current system response, but also a history of previous system responses for reaction.

This paper is organized as follows. Section 2 discusses the related work. In Section 3, we introduce the system model, formally specify history-based enforcement FSMs, and propose three examples of history-based enforcement FSMs that use a history of previous system responses for determining a reaction. Section 4 describes the evaluation of history-based enforcement FSMs for three different use case applications and compares between the proposed history-based enforcement FSMs and enforcement FSMs that do not consider previous responses for reaction. Finally, Section 5 concludes this work.

2 Related Work

Approaches based on heuristics [35], online learning [4, 23–25], or statistical regression [8, 15] are generally not able to provide any formal guarantees regarding the satisfaction or violation of non-functional properties of program executions. Finite state machines (FSMs) have been proposed to formally specify *functional* system properties [5, 14, 29]. Based on the concept of *Run-time Requirement Enforcement (RRE)* [33], FSMs have been proposed in [10–13, 30] for feedback-based enforcement of non-functional properties on MPSoCs.

Such FSM-based RREs utilize a *requirement response* vector that abstracts the system as a function that specifies for each requirement whether it has been fulfilled or violated in the current execution. Based on such a system response, the enforcer reacts by determining the configuration to be applied in the next execution iteration. However, all of the previous approaches only consider the current system response for reaction. In our work, we take into account a time window of K previous responses when deciding for the configuration for the next execution.

In control theory, the principle of *time-delayed feedback* [18, 36] has been proposed to increase the stability of a system. However, this concept has never been applied for controlling software systems. In addition, similar to [6, 16, 26, 27], approaches based on control theory can only give guarantees regarding the control stability or convergence, but not the satisfaction or violation of non-functional properties of program executions.

3 Method

In this section, we first present the considered system model and then formulate and propose three different history-based enforcement FSM strategies that take multiple previous system responses into account for taking a reaction.

3.1 System Model

Embedded systems, especially MPSoCs, often consider the execution of periodic applications, e.g., image, video, or periodic control applications. For each individual program execution, a set of non-functional requirements shall be respected, even under environmental changes, e.g., varying input. This input can be represented for each discrete execution k of an application by an *environment feature vector* $i(k) \in \mathcal{I}$, where $\mathcal{I}$ is called the *environment space* [11]. The program utilizes a number n of cores that can be dynamically changed as well as the their voltage/frequency setting m. Such a setting $\langle n, m \rangle$ is called a configuration c and the set of available configurations a *configuration space* C. FSM-based RREs [11] react based on a feedback from the system-under-control by adapting the configuration $c(k+1)$ for the $(k+1)$th execution accordingly. Figure 1 illustrates the considered system model which is described more closely in the following.

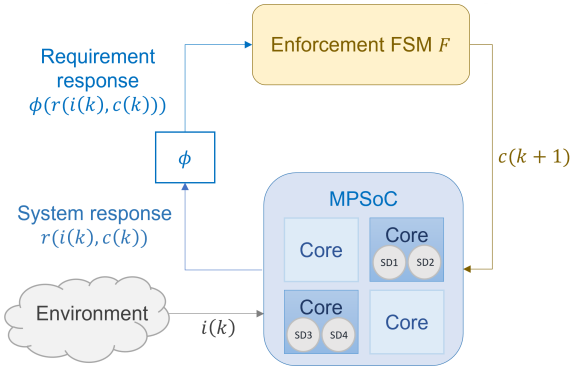


Figure 1 Illustration of a feedback-based RRE. A requirement response vector r is mapped to a binary requirement response vector ϕ that will be used by an enforcement FSM F to decide for the next configuration $c(k+1) \in C$. Reprinted from [11].

Depending on an input $i(k) \in \mathcal{I}$ and a system configuration $c(k) \in C$, let the k-th execution result into a set of H observable non-functional properties, e.g., latency and power consumption in case of $H = 2$ observable properties. The system-under-control can then be described by a *system response function* $r : \mathcal{I} \times C \to \mathbb{R}^H$ [11]. Thus, the *system response* $r(i(k), c(k)) = (o_1(k), \ldots, o_H(k))$ at execution k is a vector of H execution properties of interest, see Figure 1. Now, requirements on these properties, e.g., deadlines, must be fulfilled for each execution, where each property o_h, $h = 1, \ldots, H$ can be formulated using corridors from which the following two propositions φ_h^{LB} and φ_h^{UB} can be derived

$$\varphi_h^{LB}(o_h(k)) = (LB_h \leq o_h(k)) \tag{1}$$
$$\varphi_h^{UB}(o_h(k)) = (o_h(k) \leq UB_h) \tag{2}$$

where LB_h and UB_h refer to the lower and the upper bound, respectively, on the execution property o_h. The information regarding which proposition is satisfied and which is violated at the k-th execution is represented by a binary vector β named *requirement response*. It is obtained from the system response r using the requirement response function ϕ [11]

$$\beta(k) := \phi(o_1(k), \ldots, o_H(k)) = \big(\varphi^{LB}(o_1(k)), \varphi^{UB}(o_1(k)), \ldots, \varphi^{LB}(o_H(k)), \varphi^{UB}(o_H(k))\big) \in \{0,1\}^{2H}. \tag{3}$$

This binary requirement response vector $\beta(k)$ constitutes the input to the enforcement FSM F that determines the next configuration $c(k+1) \in C$ to enforce the given non-functional properties for the next execution.

An *enforcement FSM (F)* can be formally modeled by a deterministic finite state machine (Moore machine) which is described by a 6-tuple $(Z, z_0, B, \delta, C, \gamma)$ [11]:

- Z is a finite set of states.
- $z_0 \in Z$ is the initial state.
- B is the input alphabet.
- δ is the transition relation: $\delta \subseteq B \times Z \times Z$ with (β, z, z') representing a transition from z to z' under input β.
- C is the output alphabet, also called configuration space.
- γ is the output function that maps each state to an output (i.e., a configuration): $\gamma : Z \to C$.

Finally, in order to quantitatively compare different enforcement strategies, verification goals can be formulated in temporal logic [7]. The two types of temporal logic are linear temporal logic and branching time logic. Linear temporal logic (LTL) describes events over a single time path in the FSM. Branching time logic such as computation tree logic (CTL) quantifies the possible paths from a given state in the FSM. Different levels of strictness of requirement enforcement can be differentiated, see [34]. Accordingly, different verification goals can be defined. For example, the CTL formula $AG(\varphi_h)$ for strict enforcement indicates that φ holds for every path and at every state on the path. For loose enforcement, $AF(\varphi_h)$ specifies that for every possible path there exists a state at which φ holds, see [11].

Probabilistic verification goals, based on PCTL [2], can also be formulated to specify stochastic verification goals for loose enforcement. We utilize probabilistic verification goals that are based on *steady-state probabilities* in Markov chains as they are helpful for obtaining requirement satisfaction probabilities of long execution runs of an application regardless of the initial state. The operator S is used in PRISM [20] to reason about the steady-state probability of a model [3]. We define the verification goal $S_{=?}[\varphi]$ as the steady-state probability of being in a satisfying state for the requirement φ. Finally, we refer to the set of all considered verification goals by VG.

3.2 History-based Enforcement FSMs

In this work, we propose enforcement strategies that not only react based on the current response vector $\beta(k)$, but additionally on a *history* of system responses $(\beta(k-1), \ldots, \beta(k-K))$ belonging to the previous K execution iterations. Note that the case of $K = 0$ represents the case of enforcement FSMs that are not history-based, i.e., they only react on $\beta(k)$ and do not consider previous system responses for reaction.

In this section, we introduce exemplarily three multi-requirement history-based enforcement FSMs F_1, F_2, F_3 that consider current response $\beta(k)$ and the previous response $\beta(k-1)$ $(K = 1)$ to calculate a proper reaction. These FSMs execute on an MPSoC given with $n = 4$ available cores that can operate in $m = 20$ different power modes (voltage/frequency states). Thus, the size of the configuration space C available for enforcement is $|C| = 4 \cdot 20 = 80$. We also assume these configurations to be power-ascending so that the configuration c_j associated with $\langle n_j, m_j \rangle$ has a higher power consumption than that of configuration c_{j-1} where $0 \leq j < |C|$.

Let us consider the latency o_L and the power consumption o_P as properties of execution to be enforced, thus $H = 2$. For simplicity, we only utilize one-sided requirements so that the lower bounds are $LB_{o_L} = 0$ and $LB_{o_P} = 0$. Additionally, let the set of states Z be $|Z| = |C|$ such that the output function is a bijection $\gamma : Z \leftrightarrow C$ of sets Z and C. Thus, there is a one-to-one relation between enforcer states and configurations, such that each enforcer state $z \in Z$ uniquely outputs one configuration $c \in C$. Based on that, each enforcement FSM has as many states as $|Z| = |C| = 80$, thus, $Z = \{z_0, \cdots, z_{79}\}$, the input $\beta \in B = \{0,1\}^H = \{0,1\}^2$ with $\beta = \phi(r'(s,c)) = \phi(o_L, o_P) = ((o_L \leq UB_{o_L}), (o_P \leq UB_{o_P}))$, an assumed initial state $z_0 = 0$. Finally, we assume both the latency requirement $\varphi_L(k)$ and the power requirement $\varphi_P(k)$ are satisfied for executions $k < 0$.

3.2.1 Latency Violation-Oriented History-Based Enforcement FSM

This enforcement FSM decreases the current state by exactly one step in case of a violation of a power requirement in both the current execution $(\overline{\varphi_P(k)})$ and the previous one $(\overline{\varphi_P(k-1)})$ only when the latency requirement in both the current execution $(\varphi_L(k))$ and the previous one $(\varphi_L(k-1))$ is satisfied. It stays in the same configuration for the other cases when the latency requirement is satisfied in both executions $(\varphi_L(k))$ and $(\varphi_L(k-1))$. It increases by one step in case of a violation of an latency requirement in either the current execution $(\overline{\varphi_L(k)})$ or the previous one $(\overline{\varphi_L(k-1)})$. Finally, it increases by two steps in case of a violation of an latency requirement in both the current execution $(\overline{\varphi_L(k)})$ and the previous one $(\overline{\varphi_L(k-1)})$. A corresponding enforcement FSM $F_1 = (Z, z_0, I, \gamma, C, \delta_1)$ has the transition relation δ_1 shown in Table 1.

3.2.2 Power Violation-Oriented History-Based Enforcement FSM

This enforcement FSM increases the current state by exactly one step in case of a violation of a latency requirement in both the current execution $(\overline{\varphi_L(k)})$ and the previous one $(\overline{\varphi_L(k-1)})$ only when the power requirement in both the current execution $(\varphi_P(k))$ and the previous one $(\varphi_P(k-1))$ is satisfied. It stays in the same configuration for the other cases when the power requirement is satisfied in both executions $(\varphi_P(k))$ and $(\varphi_P(k-1))$. It decreases by one step in case of a violation of a power requirement in either the current execution $(\overline{\varphi_P(k)})$ or the previous one $(\overline{\varphi_P(k-1)})$. Finally, it decreases by two steps in case of a violation of a power requirement in both the current execution $(\overline{\varphi_P(k)})$ and the previous one $(\overline{\varphi_P(k-1)})$. A corresponding enforcement FSM $F_2 = (Z, z_0, I, \gamma, C, \delta_2)$ has the transition relation δ_2 shown in Table 2.

3.2.3 Multi-Requirement History-Based Enforcement FSM

This enforcement FSM does not favor any requirement when transitioning between enforcer states. A corresponding enforcement FSM $F_3 = (Z, z_0, I, \gamma, C, \delta_3)$ has the transition relation δ_3 shown in Table 3.

Table 1 The transition relation δ_1 of the latency-oriented history-based enforcement FSM F_1.

$z(k)$	$\beta(k-1)$		$\beta(k)$		$z(k+1)$
z_j	true	true	true	true	z_j
z_j	true	false	true	true	z_j
z_j	true	true	true	false	z_j
z_j	true	false	true	false	z_{j-1}
z_j	false	true	true	true	z_{j+1}
z_j	false	false	true	true	z_{j+1}
z_j	false	true	true	false	z_{j+1}
z_j	false	false	true	false	z_{j+1}
z_j	true	true	false	true	z_{j+1}
z_j	true	false	false	true	z_{j+1}
z_j	true	true	false	false	z_{j+1}
z_j	true	false	false	false	z_{j+1}
z_j	false	true	false	true	z_{j+2}
z_j	false	false	false	true	z_{j+2}
z_j	false	true	false	false	z_{j+2}
z_j	false	false	false	false	z_{j+2}

Table 2 The transition relation δ_2 of the power-oriented history-based enforcement FSM F_2.

$z(k)$	$\beta(k-1)$		$\beta(k)$		$z(k+1)$
z_j	true	true	true	true	z_j
z_j	true	false	true	true	z_{j-1}
z_j	true	true	true	false	z_{j-1}
z_j	true	false	true	false	z_{j-2}
z_j	false	true	true	true	z_j
z_j	false	false	true	true	z_{j-1}
z_j	false	true	true	false	z_{j-1}
z_j	false	false	true	false	z_{j-2}
z_j	true	true	false	true	z_j
z_j	true	false	false	true	z_{j-1}
z_j	true	true	false	false	z_{j-1}
z_j	true	false	false	false	z_{j-2}
z_j	false	true	false	true	z_{j+1}
z_j	false	false	false	true	z_j
z_j	false	true	false	false	z_{j-1}
z_j	false	false	false	false	z_{j-2}

4 Experimental Results

In this section, we introduce three applications for evaluating the proposed enforcement FSMs in Section 3.2. For comparison, we also perform a design space exploration (DSE) method from [13] to generate optimized enforcement FSMs for a given set of verification goals VG for each application, where these enforcement FSMs do not consider previous system responses for reaction. To perform the DSE, the NSGA-II [9] multi-objective evolutionary

Table 3 The transition relation δ_3 of the multi-requirement history-based enforcement FSM F_3.

$z(k)$	$\beta(k-1)$		$\beta(k)$		$z(k+1)$
z_j	true	true	true	true	z_j
z_j	true	false	true	true	z_{j-1}
z_j	true	true	true	false	z_{j-1}
z_j	true	false	true	false	z_{j-2}
z_j	false	true	true	true	z_{j+1}
z_j	false	false	true	true	z_j
z_j	false	true	true	false	z_j
z_j	false	false	true	false	z_{j-1}
z_j	true	true	false	true	z_{j+1}
z_j	true	false	false	true	z_j
z_j	true	true	false	false	z_j
z_j	true	false	false	false	z_{j-1}
z_j	false	true	false	true	z_{j+2}
z_j	false	false	false	true	z_{j+1}
z_j	false	true	false	false	z_{j+1}
z_j	false	false	false	false	z_j

algorithm provided by the optimization framework Opt4J [22] is used. Each run of the DSE features 100 iterations with a population size of 20 enforcement FSMs with a crossover probability of 0.9 and a mutation probability of 0.01. Each experiment was repeated three times to compensate for the randomness of the exploration.

4.1 Applications

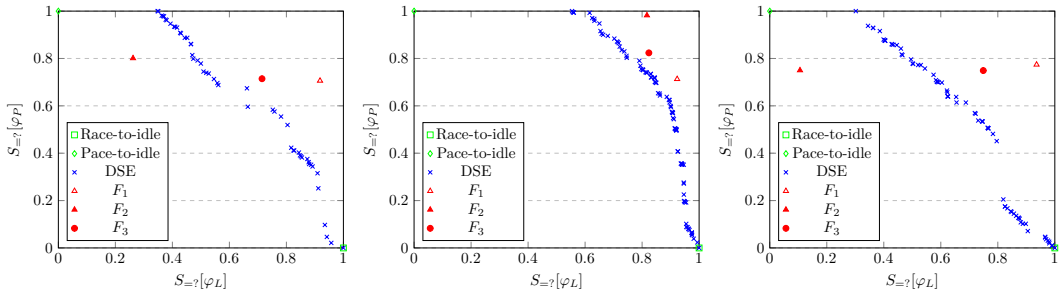
We consider three applications for evaluation. Each application is modeled by a graph of actors, where each actor processes an input $i(k)$ in each iteration k. The applications execute on a tiled many-core system that consists of a set of processing cores, peripherals like memories, and a network adapter, which are interconnected via a tile-local bus system. For this matter, a simulation framework called InvadeSIM [28], a many-core simulator for parallel applications is used.

4.1.1 Object Detection Application

An image processing application that detects a given object in each image frame by applying a scale-invariant feature transform (SIFT) matching algorithm. We use a driving car image sequence R of the KITTI-360 dataset [21] with $|R| = 100$ frames, a latency lower bound $LB_{o_L} = 0$ ms and an upper bound (deadline) $UB_{o_L} = 65$ W, an power lower bound $LB_{o_P} = 0$ mJ and an upper bound $UB_{o_P} = 5$ W.

4.1.2 String Search Application

This application stems from the ParMiBench benchmark suite [17] that searches in a given input text k with $i(k)$ lines for a given pattern. For this use case, we create a trace of $|R| = 100$ randomly generated texts, each having $i(k)$ lines. We use the bounds $LB_{o_L} = 0$ ms, $UB_{o_L} = 15$ ms, $LB_{o_P} = 0$ W and $UB_{o_P} = 1.5$ W.



(a) Object detection. **(b)** String search. **(c)** SHA.

Figure 2 Verification results for the proposed history-based enforcement FSMs F_1, F_2, F_3 for a history of $K = 1$, compared to DSE-optimized enforcement FSMs that do not consider the response history [13], and the heuristic techniques race-to-idle and pace-to-idle [19].

4.1.3 Secure Hash Application

Another application from the ParMiBench benchmark suite [17]. This security application computes the hash for the input k that consists of $i(k)$ messages. For this use case, we create a trace of $|R| = 100$ randomly generated inputs k, with $LB_{o_L} = 0$ ms, $UB_{o_L} = 9$ ms, $LB_{o_P} = 0$ W, and $UB_{o_P} = 3$ W.

4.2 Results

Figure 2 shows the verification results of the proposed history-based enforcers F_1, F_2, F_3 that consider the previous response $\beta(k-1)$ together with enforcement FSMs that are obtained from the DSE method in [13] and do not consider any previous response for reaction, as well as race-to-idle (i.e., running with the highest configuration c_{79}) and pace-to-idle (i.e., running with the slowest configuration c_0) [19].

We notice in Figure 2 that the history-based enforcement FSMs F_1 and F_2 are not dominated by any other enforcement FSM in all of the three applications. Also, the history-based enforcement FSM F_3 is not dominated in the case of string search application. This shows that reacting based on a history of previous system responses can enhance the probability of satisfying the considered verification goals. The reason is the larger design space of transition possibilities in the enforcement FSM.

Table 4 shows the average verification time, number of states, and transitions for 10 randomly-generated enforcement FSMs with different history options. Enforcement FSMs with $K = 0$ indicates that they only react on the current system response $\beta(k)$. History-based enforcement FSMs with $K = 1$ implies that they include the previous system response $\beta(k-1)$ for reaction as well as $\beta(k)$. Finally, history-based enforcement FSMs with $K = 2$ consider the system responses $\beta(k-2), \beta(k-1)$, and $\beta(k)$ for reaction. We notice that reacting based on previous system responses leads to a substantial increase in verification times. This is explained by the increase of number of states and transitions of the resulting enforcement FSM. We also notice that this increase is proportional to the length of the time window K of previously considered responses.

5 Conclusion

In this paper, we proposed to integrate a history of previous system responses into the design of enforcement strategies. The evaluation shows that such history-based enforcement FSMs have the potential to have higher probabilities of satisfying a given set of verification goals

Table 4 Average verification time, number of states, and transitions for 10 randomly-generated enforcement FSMs with different history options.

	$K = 0$			$K = 1$			$K = 2$		
Application	time (ms)	states	transitions	time (ms)	states	transitions	time (ms)	states	transitions
Object detection	133.0	262.7	710.7	357.0	1,193.3	3,176.3	6,460.1	4,064.3	10,786.6
String Search	416.1	1,299.4	5,166.7	24,786.9	10,354.9	41,301.9	840,498.1	41,477.8	166,058.5
SHA	179.3	453.9	1,513.5	2,894.3	2,800.1	9,303.8	63,172.1	9,726.3	32,336.3

than enforcement FSMs that do not consider any system response history. This offers system designers with a trade-off between complexity and performance. In the future, we aim to automatically optimize history-based enforcement FSMs for a given set of verification goals.

References

1 Nidhi Anantharajaiah, Tamim Asfour, Michael Bader, Lars Bauer, Jürgen Becker, Simon Bischof, Marcel Brand, Hans-Joachim Bungartz, Christian Eichler, Khalil Esper, Joachim Falk, Nael Fasfous, Felix Freiling, Andreas Fried, Michael Gerndt, Michael Glaß, Jeferson Gonzalez, Frank Hannig, Christian Heidorn, Jörg Henkel, Andreas Herkersdorf, Benedict Herzog, Jophin John, Timo Hönig, Felix Hundhausen, Heba Khdr, Tobias Langer, Oliver Lenke, Fabian Lesniak, Alexander Lindermayr, Alexandra Listl, Sebastian Maier, Nicole Megow, Marcel Mettler, Daniel Müller-Gritschneder, Hassan Nassar, Fabian Paus, Alexander Pöppl, Behnaz Pourmohseni, Jonas Rabenstein, Phillip Raffeck, Martin Rapp, Santiago Narváez Rivas, Mark Sagi, Franziska Schirrmacher, Ulf Schlichtmann, Florian Schmaus, Wolfgang Schröder-Preikschat, Tobias Schwarzer, Mohammed Bakr Sikal, Bertrand Simon, Gregor Snelting, Jan Spieck, Akshay Srivatsa, Walter Stechele, Jürgen Teich, Isaías A. Comprés Ureña, Ingrid Verbauwhede, Dominik Walter, Thomas Wild, Stefan Wildermann, Mario Wille, Michael Witterauf, and Li Zhang. *Invasive Computing*. FAU University Press, 2022.

2 Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. On the Logical Characterisation of Performability Properties. In *Automata, Languages and Programming, 27th International Colloquium, ICALP 2000, Geneva, Switzerland, July 9-15, 2000, Proceedings*, volume 1853 of *Lecture Notes in Computer Science*, pages 780–792. Springer, 2000.

3 Christel Baier, Joost-Pieter Katoen, and Holger Hermanns. Approximate symbolic model checking of continuous-time markov chains. In *CONCUR '99: Concurrency Theory, 10th International Conference, Eindhoven, The Netherlands, August 24-27, 1999, Proceedings*, volume 1664 of *Lecture Notes in Computer Science*, pages 146–161. Springer, 1999.

4 Dwaipayan Biswas, Vibishna Balagopal, Rishad A. Shafik, Bashir M. Al-Hashimi, and Geoff V. Merrett. Machine learning for run-time energy optimisation in many-core systems. In David Atienza and Giorgio Di Natale, editors, *Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017*, pages 1588–1592. IEEE, 2017.

5 Roderick Bloem, Bettina Könighofer, Robert Könighofer, and Chao Wang. Shield synthesis: Runtime Enforcement for Reactive Systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 9035 of *Lecture Notes in Computer Science*, pages 533–548. Springer, 2015.

6 Sophie Cerf, Raphaël Bleuse, Valentin Reis, Swann Perarnau, and Eric Rutten. Sustaining performance while reducing energy consumption: a control theory approach. In *Euro-Par 2021: Parallel Processing: 27th International Conference on Parallel and Distributed Computing, Lisbon, Portugal, September 1–3, 2021, Proceedings 27*, pages 334–349. Springer, 2021.

7 Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Logics of Programs, Workshop, Yorktown Heights, New York, USA, May 1981*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

8 Junio Cezar Ribeiro Da Silva, Lorena Leão, Vinicius Petrucci, Abdoulaye Gamatié, and Fernando Magno Quintão Pereira. Mapping computations in heterogeneous multicore systems with statistical regression on program inputs. *ACM Transactions on Embedded Computing Systems (TECS)*, 20(6):1–35, 2021.

9 Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.*, 6(2):182–197, 2002.

10 Khalil Esper, Jan Spieck, Pierre-Louis Sixdenier, Stefan Wildermann, and Jürgen Teich. RAVEN: reinforcement learning for generating verifiable run-time requirement enforcers for MPSoCs. In *Fourth Workshop on Next Generation Real-Time Embedded Systems, NG-RES 2023, January 18, 2023, Toulouse, France*, volume 108 of *OASIcs*, pages 7:1–7:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

11 Khalil Esper, Stefan Wildermann, and Jürgen Teich. Enforcement FSMs: specification and verification of non-functional properties of program executions on MPSoCs. In *MEMOCODE '21: 19th ACM-IEEE International Conference on Formal Methods and Models for System Design, Virtual Event, China, November 20–22, 2021*, pages 21–31. ACM, 2021.

12 Khalil Esper, Stefan Wildermann, and Jürgen Teich. Multi-requirement enforcement of non-functional properties on MPSoCs using enforcement FSMs – A case study. In *Third Workshop on Next Generation Real-Time Embedded Systems, NG-RES@HiPEAC 2022, June 22, 2022, Budapest, Hungary*, volume 98 of *OASIcs*, pages 2:1–2:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

13 Khalil Stefan Wildermann Esper and Jürgen Teich. Automatic synthesis of FSMs for enforcing non-functional requirements on MPSoCs using multi-objective evolutionary algorithms. *ACM Trans. Des. Autom. Electron. Syst.*, August 2023.

14 Yliès Falcone, Laurent Mounier, Jean-Claude Fernandez, and Jean-Luc Richier. Runtime enforcement monitors: composition, synthesis, and enforcement abilities. *Formal Methods in System Design*, 38(3):223–262, 2011.

15 Xinwei Fang, Sinem Getir Yaman, Radu Calinescu, Julie Wilson, and Colin Paterson. Predicting nonfunctional requirement violations in autonomous systems. *ACM Transactions on Autonomous and Adaptive Systems*, 2023.

16 Connor Imes, David HK Kim, Martina Maggio, and Henry Hoffmann. POET: a portable approach to minimizing energy under soft real-time constraints. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 75–86. IEEE Computer Society, 2015.

17 Syed Muhammad Zeeshan Iqbal, Yuchen Liang, and Håkan Grahn. Parmibench – An open-source benchmark for embedded multiprocessor systems. *IEEE Comput. Archit. Lett.*, 9(2):45–48, 2010.

18 Wolfram Just, Thomas Bernard, Matthias Ostheimer, Ekkehard Reibold, and Hartmut Benner. Mechanism of time-delayed feedback control. *Physical Review Letters*, 78(2):203, 1997.

19 David H. K. Kim, Connor Imes, and Henry Hoffmann. Racing and pacing to idle: Theoretical and empirical analysis of energy optimization heuristics. In *2015 IEEE 3rd International Conference on Cyber-Physical Systems, Networks, and Applications, CPSNA 2015, Kowloon, Hong Kong, China, August 19-21, 2015*, pages 78–85. IEEE Computer Society, 2015.

20 Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *Computer Aided Verification – 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer, 2011.

21 Yiyi Liao, Jun Xie, and Andreas Geiger. KITTI-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d. *CoRR*, abs/2109.13410, 2021.

22 Martin Lukasiewycz, Michael Glaß, Felix Reimann, and Jürgen Teich. Opt4j: a modular framework for meta-heuristic optimization. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1723–1730, 2011.

23 Sumit K. Mandal, Ganapati Bhat, Janardhan Rao Doppa, Partha Pratim Pande, and Ümit Y. Ogras. An energy-aware online learning framework for resource management in heterogeneous platforms. *ACM Trans. Design Autom. Electr. Syst.*, 25(3):28:1–28:26, 2020.

24 Sumit K. Mandal, Ganapati Bhat, Chetan Arvind Patil, Janardhan Rao Doppa, Partha Pratim Pande, and Ümit Y. Ogras. Dynamic resource management of heterogeneous mobile platforms via imitation learning. *IEEE Trans. Very Large Scale Integr. Syst.*, 27(12):2842–2854, 2019.

25 Maxime Mirka, Gilles Sassatelli, and Abdoulaye Gamatié. Online learning for dynamic control of openmp workloads. In *9th International Conference on Modern Circuits and Systems Technologies, MOCAST 2020, Bremen, Germany, September 7-9, 2020*, pages 1–6. IEEE, 2020.

26 Anway Mukherjee and Thidapat Chantem. Energy management of applications with varying resource usage on smartphones. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 37(11):2416–2427, 2018.

27 Thannirmalai Somu Muthukaruppan, Mihai Pricopi, Vanchinathan Venkataramani, Tulika Mitra, and Sanjay Vishin. Hierarchical power management for asymmetric multi-core in dark silicon era. In *The 50th Annual Design Automation Conference 2013, DAC '13, Austin, TX, USA, May 29 – June 07, 2013*, pages 174:1–174:9. ACM, 2013.

28 Sascha Roloff, Frank Hannig, and Jürgen Teich. *Modeling and Simulation of Invasive Applications and Architectures*. Computer Architecture and Design Methodologies. Springer, 2019.

29 Fred B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(1):30–50, 2000.

30 Jan Spieck, Pierre-Louis Sixdenier, Khalil Esper, Stefan Wildermann, and Jürgen Teich. Hybrid genetic reinforcement learning for generating run-time requirement enforcers. In *2023 21st ACM-IEEE International Symposium on Formal Methods and Models for System Design (MEMOCODE)*, pages 23–35, 2023.

31 Jürgen Teich, Michael Glaß, Sascha Roloff, Wolfgang Schröder-Preikschat, Gregor Snelting, Andreas Weichslgartner, and Stefan Wildermann. Language and Compilation of Parallel Programs for *-Predictable MPSoC Execution Using Invasive Computing. In *10th IEEE International Symposium on Embedded Multicore/Many-core Systems-on-Chip, MCSOC 2016, Lyon, France, September 21-23, 2016*, pages 313–320. IEEE Computer Society, 2016.

32 Jürgen Teich, Jörg Henkel, Andreas Herkersdorf, Doris Schmitt-Landsiedel, Wolfgang Schröder-Preikschat, and Gregor Snelting. Invasive computing: An overview. In *Multiprocessor System-on-Chip – Hardware Design and Tool Integration*, pages 241–268. Springer, 2011.

33 Jürgen Teich, Pouya Mahmoody, Behnaz Pourmohseni, Sascha Roloff, Wolfgang Schröder-Preikschat, and Stefan Wildermann. Run-Time Enforcement of Non-functional Program Properties on MPSoCs. In *A Journey of Embedded and Cyber-Physical Systems*, pages 125–149. Springer, 2021.

34 Jürgen Teich, Behnaz Pourmohseni, Oliver Keszöcze, Jan Spieck, and Stefan Wildermann. Run-Time Enforcement of Non-Functional Application Requirements in Heterogeneous Many-Core Systems. In *25th Asia and South Pacific Design Automation Conference, ASP-DAC 2020, Beijing, China, January 13-16, 2020*, pages 629–636. IEEE, 2020.

35 Xiaohang Wang, Amit Kumar Singh, Bing Li, Yang Yang, Hong Li, and Terrence S. T. Mak. Bubble budgeting: Throughput optimization for dynamic workloads by exploiting dark cores in many core systems. *IEEE Trans. Computers*, 67(2):178–192, 2018. `doi:10.1109/TC.2017.2735967`.

36 Dong Yue and Qing-Long Han. Delayed feedback control of uncertain systems with time-varying input delay. *Automatica*, 41(2):233–240, 2005.